Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Multi-type task offloading for wireless Internet of Things by federated deep reinforcement learning

Zhao Tong[a,*], Jiake Wang[a], Jing Mei[a], Kenli Li[b], Wenbin Li[c], Keqin Li[d]

[a] College of Information Science and Engineering, Hunan Normal University, Changsha, 410012, China
[b] College of Information Science and Engineering, Hunan University, Changsha, 410082, China
[c] College of Information Science and Engineering, Hunan Institute of Science and Technology, Yueyang 414006, China
[d] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Article history:
Received 4 November 2022
Received in revised form 27 March 2023
Accepted 1 April 2023
Available online 3 April 2023

Keywords:
Deep reinforcement learning
Federated learning
Internet of Things (IoT)
Mobile edge computing (MEC)
Multi-type task offloading
Service experience guarantee

ABSTRACT

With the popularity of Internet of Things (IoT) smart devices, the amount of data generated by these devices has grown rapidly. In these mobile edge computing (MEC) environments, it is not only important to save time and energy in offloading tasks, but also to protect user data. In this paper, due to the dynamics and complexity of the system, a multi-type task offloading based on a multi-capability federated deep Q-network (M2FD) algorithm is proposed to optimize the bi-objective performance. The algorithm consists of two parts, federated learning protects user privacy by transmitting model for training instead of data, and deep reinforcement learning trains model accuracy and identifies suitable offloading nodes with heterogeneous capabilities for multi-type tasks. In addition, under the constraints of the service experience guarantee (SEG) model, the tasks are offloaded with the goal of improving system utility while reducing system cost. Experiments show that the M2FD increases system utility, guarantees privacy, and reduces task response time and energy consumption.

© 2023 Elsevier B.V. All rights reserved.

1. Introduction

At present, due to the rapid development of Internet of Things (IoT) and wireless communication technologies in the 5th generation (5G) era, the number of various mobile smart devices has exploded [1]. Examples include virtual reality devices, smart cameras, healthcare devices, etc. The Cisco white paper indicates that by 2023, the global IoT devices will reach 14.7 billion devices, accounting for 50 percent of all connected devices [2]. Hundreds of millions of device connections can lead to congested communication channels and excessive bandwidth pressure. As the number of devices grows, the amount of task data will increase dramatically, and the content will be more complex [3]. Based on the above factors, robust computing power is needed to support the operation of the devices. Real-time feedback, low energy consumption and fast connection are also essential to ensure the user quality of experience (QoE). In addition, people are increasingly aware of privacy protection. For IoT devices, the privacy of data is critical [4]. Whether it is medical, industrial, political data or personal information should be protected.

To solve the above difficulties, it is necessary to divide the large amount of data into several tasks and use other computing resources to complete the tasks. Traditional cloud computing schedules all tasks to the cloud for completion, providing efficient processing for computational tasks. However, cloud computing is often unable to complete computing tasks in a specified time due to the high latency of task transmission. Therefore, mobile edge computing instead of cloud computing has been widely studied [5–7]. In contrast to the cloud, the edge is located close to the side of the data source, which makes the transmission latency and energy consumption well controlled. In addition, mobile edge computing (MEC) puts some tasks in local computing. Fewer data are transmitted, which secures the data to some extent. Hence, MEC not only meets the computation and communication needs of most tasks but also increases the efficiency and security of the system. However, there are many issues that need to be addressed in MEC.

(1) *The dynamically changing environment.* There are multiple types of tasks in a heterogeneous environment, and they are generated at uncertain times. In addition, the distribution of resources within the system is constantly changing. Therefore, in a dynamically changing environment, it is a challenge to make offloading decisions that will benefit the utility of the global system based only on the current network, device states, and task scenarios.
(2) *The complex optimization problem.* It is important to make the system fully utilize the computational resources to satisfy as many tasks as possible for execution. However, such

* Corresponding author.
E-mail address: tongzhao@hunnu.edu.cn (Z. Tong).

https://doi.org/10.1016/j.future.2023.04.004
0167-739X/© 2023 Elsevier B.V. All rights reserved.

offloading optimization is usually considered as an NP-hard problem. The optimal solution cannot be obtained in polynomial time [8]. Therefore, traditional offloading strategies have difficulty fully adapting to various environments in MEC, resulting in a sharp decrease in QoE for users.

(3) *The data privacy problem.* Due to large amounts of data are transmitted, there is a great risk of leakage. Traditional privacy protection technologies, such as physical layer security technologies [9], blockchain privacy technologies [10] and trusted assessment frameworks [11], are not effective in avoiding the problem. In addition, the general data protection regulation (GDPR) enacted by the European Union makes it illegal to trade in leaked data. This makes it urgent to address the problem of user privacy.

To solve the first two problems, deep reinforcement learning (DRL) is an effective approach. In the field of machine learning, reinforcement learning (RL) can learn from experiences. It has excellent decision-making ability. However, algorithms based on RL cannot solve the problem of high-dimensional data caused by variable environmental states. Such as, Q-learning, Sarsa, Policy Gradients. Deep learning (DL) driven by big data has an outstanding perception of dynamic environments [12]. This approach compensates for the shortcomings of RL. In addition, DL uses a back-propagation of errors algorithm to optimize the objectives. Combining DL and RL, DRL is widely applied in MEC to solve changing environments and complex optimization problems [13, 14]. Deep Q-network (DQN) is a classical DRL algorithm that uses a neural network instead of a table in the Q-learning algorithm. The high-dimensional state problem of agent interaction with a dynamic environment is well solved. A better offloading decision is also obtained to complete the task with low energy consumption within the deadline. For the third privacy problem, Federated Learning (FL) is an emerging solution. FL is proposed in [15] as a decentralized learning approach to protect user data privacy. It allows each device to use local data for model training. After this, the edge server will collect and aggregate all the models to obtain a global model. Therefore, the device transmits small models to the edge to avoid a large amount of private data transmission. While reducing the risk of privacy leakage, it also relieves the bandwidth pressure. In addition, distribution after model aggregation is beneficial to solve the problem of data islands.

In this paper, we integrate the DRL and FL into MEC by optimizing the latency and energy consumption of task offloading. By utilizing edge servers, the computation and communication pressure of end devices are relieved within the constraints of the service experience guarantee (SEG) model. In addition, the FL training method further improves the system efficiency and security. The bi-objective optimization of $\underline{m}$ulti-type task offloading based on $\underline{m}$ulti-capability $\underline{f}$ederated $\underline{D}$QN (M2FD) algorithm is proposed. The main contributions of this paper are as follows.

(1) We propose a two-layer offloading scenario that handles four types of tasks in different applications. In addition, heterogeneous processor computing powers are considered. In this IoT environment, the task model is introduced. We construct communication, computational models and SEG model, based on multi-type tasks and heterogeneous computing capabilities.

(2) We investigate a distributed model based on FL. FedAvg is used to aggregate the models of agents in the edge. The training approach not only enables the agents to accelerate the convergence of the model but also optimizes the task processing latency and energy consumption. In addition, the security and privacy of the user data are preserved.

(3) We develop a bi-objective optimization strategy for latency and energy consumption with the entropy method. Moreover, we consider the task failure rate to ensure the user QoE. Due to the large number of tasks and the dynamic network, it is difficult to obtain an optimal policy by traditional machine learning methods. M2FD is proposed to solve this problem.

(4) We compare the FL model in an edge environment with a centralized learning model in a distributed framework. The performance experiments include task size, task scenarios and task density. Under different conditions, the average system utility of the M2FD algorithm is improved by 15%, 15%, and 13%, respectively.

The remainder of this paper is organized as follows. Section 2 describes the related work. Section 3 presents the overview and models of the system. Section 4 formulates the model problem. Section 5 proposes the M2FD algorithm in detail. Section 6 evaluates and analyzes the proposed algorithm by experiments. Finally, Section 7 concludes this work.

2. Related work

2.1. Task offloading in mobile edge computing

To relieve the computation and energy consumption pressure on end devices, offloading task to edge computing is widely considered an effective solution [16–18]. However, in heterogeneous MEC environments, the overall efficiency of offloading decisions is influenced by many factors. Due to the limited computing power of a single MEC server, Zhou et al. [19] studied the task offloading problem based on collaborative MEC servers. The task offloading algorithm based on artificial bee colony, particle swarm optimization and genetic algorithm (ABC-PSO-GA) is proposed to solve the problem. The goal is to minimize the ratio of actual latency to maximum allowed latency. Heterogeneous server resources and fairness among all tasks are also considered. Tan et al. [20] investigated the problems of task offloading, computation, and communication resource allocation in a collaborative mobile edge computing network. A two-layer alternate approach combining the heuristic algorithm and the DRL algorithm is proposed. The optimization objective is to minimize the total user energy consumption under the latency constraint. Dong et al. [21] considered the competition for cache resources and proposed a joint optimization problem for content caching and computational offloading. It is transformed into two subproblems. By solving them and performing simulations, the results show that this approach reduces the average response time of a task by 20.52% compared to the baseline. In the above studies, the ratio of actual latency to maximum allowed latency, task latency, or user energy consumption was used as an independent optimization objective. However, task latency and energy consumption are not optimized as a bi-objective. In addition, they do not take into account the type of server resources and tasks in a heterogeneous environment. The QoE of users should also be guaranteed.

2.2. Deep reinforcement learning-based mobile edge computing

Artificial intelligence (AI) is flourishing, especially DRL, driven by algorithms, computing power and big data. An increasing number of studies are applying it to IoT and MEC fields [22–25]. Mohammed et al. [26] proposed an RL-based active sensor selection mechanism to solve the target localization problem in IoT, and a state–space reduction technique to optimize the placement and selection of computational nodes in the system. Chen et al. [27] studied a DRL-based algorithm to solve collaborative mobile computing offloading under a three-layer network

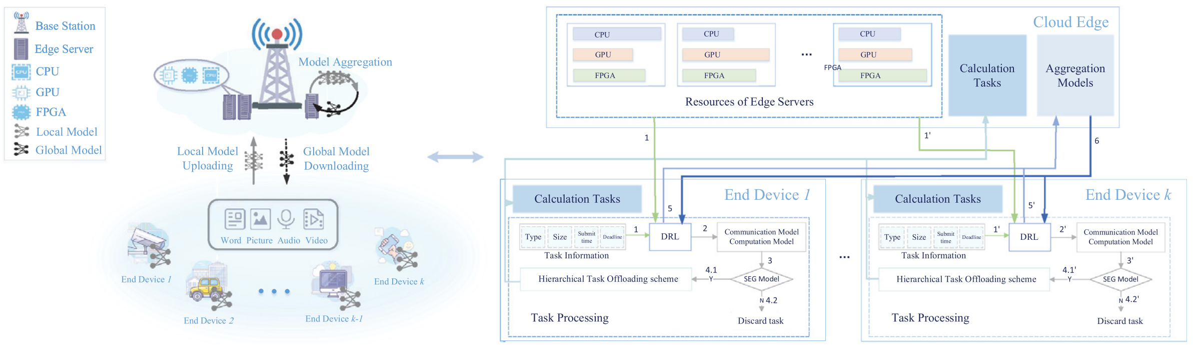


Fig. 1. The two-layer MEC system architecture and workflow.

model. The priority of the experience pool is set to select the most effective experience samples to complete the learning and training. Zhu et al. [28] proposed a DRL-based multi-agent offloading scheme considering the uncertainty in the environment. The objective is to determine the optimal offloading decision and to minimize the total task processing latency. In [29], the issue of multiple connections and task offloading in an MEC environment is considered. To solve this problem, a multi-agent deep reinforcement learning (MADRL) algorithm is proposed. It aims to improve the collaboration rate of edge devices and minimize the computational latency. Unfortunately, these works paid attention to minimizing the system cost. During model training, the risk of data leakage from end devices is high. As a result, there is an increased probability of threats to information security.

2.3. Federated learning-based mobile edge computing

FL proposes a solution to the user privacy issues by sharing the model rather than transmitting the data. Many studies in MEC are using FL extensively to avoid data leakage [30–32]. Ji et al. [33] investigated the edge assisted federated learning (EAFL) to reduce the computing pressure on lagging end devices. In addition, the task offloading rate is optimized to obtain the smallest learning latency of the system. Shinde et al. [34] proposed an FL-based distributed learning framework to solve the vehicle user computation offloading problem. In this framework, a genetic algorithm is used to minimize the overall latency and energy consumption. Wang et al. [35] considered the task data size variation with time and adjusted the dynamic resources. A support vector machine (SVM) based FL algorithm was proposed. Its objective is to minimize the delay and energy consumption of the offloading tasks. Although many studies of FL frameworks already exist, there are currently few works combining it with DRL. To address the challenge of high-dimensional data in MEC, DRL is an effective means.

In this paper, we present a multi-type task offloading model with limited heterogeneous computing resources. Based on this model for a MEC environment, an FL-based DRL bi-objective optimization algorithm is proposed. The user's QoE is guaranteed while minimizing overall system latency and energy consumption. In addition, user data privacy is also protected.

3. System overview and model

In this section, we describe an overview of the two-layer MEC system. In addition, we introduce all the models in this system.

3.1. System overview

An MEC network model is considered, consisting of the end-device layer and edge-server layer. On the one hand, K smart devices are located at the end-device layer, such as cars, cameras, and smartphones. Multi-type of tasks are generated by the devices due to the different application requirements on the devices. Such as text, image, audio, and video. Based on the FL training framework, each device trains and updates the local model using the privacy task data produced by the device. On the other hand, the edge-server layer contains a base station consisting of N servers. The servers take two main roles: (1) To reduce system cost, provide computing power support for ends' offloading tasks. (2) To protect data privacy, collect and aggregate local models of end devices. Due to the respective characteristics of above chips and adaptation to real MEC environment, three kinds of processing capabilities are integrated on the edge servers of the system. Due to the various characteristics of the individual chips on the market and their adaptability to the actual MEC environment, different processors, namely CPU, GPU, and FPGA, are integrated on the system's edge servers, which are designed to fit the reality and improve the efficiency of the system. CPU as the current general purpose processor is able to handle complex conditions and branches, and it has better capability for text processing. Chips such as the GPU and FPGA are also coming into the mainstream. GPU highlights the maximization of computational output in the chip design. It has obvious advantages in floating point matrix operations. Different functions and data types are accommodated by customizing the FPGA instructions. In addition, the wireless channel between the end devices and edge servers is responsible for transmitting tasks and model parameters.

The system architecture and workflow is illustrated in Fig. 1. First, the real-time task information and resource status of the edge servers are input to the local DRL model. Second, the local DRL model selects an offloading scheme for the task. Meanwhile, the system cost and utility of the scheme are calculated. Third, the system determines whether the scheme is executed over the deadline. Fourth, if the latency is higher than the deadline, the task fails and is discarded. Otherwise, an offloading scheme suitable for the task is obtained. Fifth, in a fixed training round, the edge server collects the local DRL models of the devices selected for training. They are aggregated into a global DRL model. Finally, all end devices receive the global model and update the local model. The main notions in the MEC model are summarized in Table 1.

3.2. System model

3.2.1. Task model

In the MEC environment, each device generates only one task at the current time slot. The end device k generates independent

Table 1
Summary of notions in the model.

Notion	Definition
h_k^n	The channel gain between device k and server n
r_k^n	The transmission rate between device k and server n
E_k^{loc}	The energy consumption of local computing in device k
$E_{k,n}^{off}$	The energy consumption of offloading to server n
E_k^n	The total energy consumption of the task generated by device k
E_{tot}	The total energy consumption of all tasks
M_k	The task generated by device k
R_{fai}	The task failure rate
T_k^{loc}	The offloading latency of local computing in device k
$T_{k,n}^{off}$	The offloading latency of offloading to server n
T_n^{wait}	The waiting time for the task to be calculated on server n
$T_{k,n}^{trt}$	The response time of the task
T_{avg}^{trt}	The average response time for all tasks
U_k	The system utility of the task
α_k^{ddl}	The deadline of the task M_k
α_k^{si}	The size of the task M_k
α_k^{sub}	The submission time of the task M_k
α_k^{tp}	The type of the task M_k
γ_k	The computing frequency of device k
γ_n^{pro}	The computing frequency of different processors on server n

tasks. At time slot t, the task attributes are defined as $M_k(t) = \left\{\alpha_k^{tp}(t), \alpha_k^{si}(t), \alpha_k^{sub}(t), \alpha_k^{ddl}(t)\right\}$. $\alpha_k^{tp}(t)$ represents the type of task, where 0 denotes text, 1 denotes image, 2 denotes audio, and 3 denotes video. $\alpha_k^{si}(t)$ represents the size of task. $\alpha_k^{sub}(t)$ and $\alpha_k^{ddl}(t)$ represent the submission time and deadline of the task, respectively. In addition, independent tasks follow 0–1 offloading. Optionally, it can be computed locally, or be offloaded to an edge server for computation. The offloading decision of device k generating a task at time slot t is denoted by $\partial_k(t)$. If $\partial_k(t) = 0$, it indicates that the task is computed locally. If $\partial_k(t) > 0$, it means that the task is offloaded to the edge server for computation.

3.2.2. Communication model

When $\partial_k(t) > 0$, the end device communicates with the edge server via wireless channels. The device needs to consume energy to send tasks. The model considers devices and servers communicating with each other via frequency division multiple access (FDMA) technology, with each device being allocated bandwidth B. Assume that $h_k^n(t)$ denotes the channel gain between device k to server n, which represents the fading and fading characteristics of the channel, defined as $h_k^n(t) = \varphi_k^n(t)\, g_k^n(t)$, where $\varphi_k^n(t)$ represents the small-scale channel fading power gain in time t. $g_k^n(t)$ denotes the large-scale channel fading power gain, which is determined by the distance between device k and server n and is negatively related to the distance [36,37]. Thus, according to the Shannon–Hartley formula, the channel transmission rate is denoted as

$$r_k^n(t) = B\log_2\left(1 + \frac{P_k h_k^n(t)}{N_0 B}\right), \tag{1}$$

where P_k indicates the transmit power of device k and N_0 is the noise density [38].

3.2.3. Computation model

There are two different computational sub-models for the two different computational locations of the task M_k by devise k.

Task computing locally. When $\partial_k(t) = 0$, the task M_k is executed locally. γ_k is used to denote the CPU frequency of local device k, and c_0 is the number of CPU cycles to calculate 1 bit of data. The computation latency of the task locally can be defined as

$$T_k^{loc}(t) = \frac{\alpha_k^{si}(t)\, c_0}{\gamma_k}. \tag{2}$$

$\eta_k(\gamma_k)^2$ represents the energy consumption for each compute node by device k, where η_k is the effective switched capacitance [16]. Therefore, the local computing energy consumption of a task is defined as

$$E_k^{loc}(t) = \eta_k(\gamma_k)^2 \alpha_k^{si}(t)\, c_0. \tag{3}$$

Task computing at the edge servers. After the task M_k is offloaded to one of the edge servers, it will be allocated a certain amount of computing resources. Since the task size is greatly larger than the computation result size, only the latency of the task transmission up-link is considered, without considering the computation result return. For different types of tasks, different processing speeds exist for heterogeneous processors. The sum of transmission latency and computation latency is defined as

$$T_{k,n}^{off}(t) = \frac{\alpha_k^{si}(t)\, c_n^{pro}}{\gamma_n^{pro}\left(\alpha_k^{tp}(t)\right)} + \frac{\alpha_k^{si}(t)}{r_k^n(t)}, \tag{4}$$

where c_n^{pro} represents the number of different processor cycles on server n to compute 1 bit of data, and $\gamma_n^{pro}\left(\alpha_k^{tp}(t)\right)$ denotes the frequency of different processors for processing various types of tasks in the server n. In addition, the task has a waiting time $T_n^{wait}(t)$ for computation when is offloaded to the edge server, which depends on the task queue of the computing server. If the queue is empty, $T_n^{wait}(t) = 0$, otherwise, $T_n^{wait}(t)$ is the sum of the computation times of all tasks in the queue. Then, the energy consumption in the transmission and computation process are considered mainly [39]. The total energy consumption is defined as

$$E_{k,n}^{off}(t) = P_k \frac{\alpha_k^{si}(t)}{r_k^n(t)} + \alpha_k^{si}(t)\, q_n, \tag{5}$$

where q_n is the energy consumption for computing 1 bit of data.

Therefore, for the task M_k, the task response time (TRT) and energy consumption are defined as

$$\begin{aligned} T_{k,n}^{trt}(t) =& \Gamma(\partial_k(t)) T_k^{loc}(t) \\ &+ (1 - \Gamma(\partial_k(t)))\left(T_{k,n}^{off}(t) + T_n^{wait}(t)\right), \end{aligned} \tag{6}$$

and

$$E_k^n(t) = \Gamma(\partial_k(t)) E_k^{loc}(t) + (1 - \Gamma(\partial_k(t)))\, E_{k,n}^{off}(t), \tag{7}$$

respectively, where $\Gamma(\partial_k(t))$ is an indicator function. If $\partial_k(t) = 0$, $\Gamma(\partial_k(t))$ equals 1, otherwise, it equals 0.

3.2.4. SEG model

To guarantee user QoE, the task is completed on the edge server needs to satisfy

$$\alpha_k^{sub}(t) + T_{k,n}^{trt}(t) \le \alpha_k^{ddl}(t). \tag{8}$$

When the TRT is within the deadline, i.e., Eq. (8) is valid, the task decision is executed. Otherwise, the task fails and is discarded.

4. Problem formulation

In this section, first, we introduce the four optimization indicators of the model. Next, we describe the specific optimization problem. Finally, the method of setting the weights in the optimization problem is analyzed.

4.1. Optimization indicators

With the two-layer MEC model, it is usually necessary to offload tasks to the edge server for computation owing to the limitation of end device resources. Therefore, the scheme of task offloading and resource allocation is formulated via the M2FD algorithm. To measure the algorithm performance, the average response time of the tasks T_{avg}^{trt}, the total energy consumption E_{tot}, the task failure rate R_{fai}, and the utility of the system U_k are considered, which can be described as

$$T_{avg}^{trt} = \frac{1}{V}\sum_{t=1}^{\mathrm{T}}\sum_{k=1}^{K} T_{k,n}^{trt}(t), \tag{9}$$

$$E_{tot} = \sum_{t=1}^{\mathrm{T}}\sum_{k=1}^{K} E_k^n(t), \tag{10}$$

$$R_{fai} = \frac{V_{fai}}{V}, \tag{11}$$

and

$$\begin{aligned} U_k(t) =& \beta_1(t)\frac{T_k^{loc}(t) - T_{k,n}^{trt}(t)}{T_k^{loc}(t)} \\ &+ \beta_2(t)\frac{E_k^{loc}(t) - E_k^n(t)}{E_k^{loc}(t)}, \end{aligned} \tag{12}$$

respectively. In Eq. (9), $V = \mathrm{T}K$ denotes the total number of tasks during the total system runtime T. In Eq. (11), V_{fai} represents the number of failed tasks. The $\beta_1(t)$ and $\beta_2(t)$ are two weight factors in Eq. (12). The relationship between the weight factors is $\beta_1(t) + \beta_2(t) = 1$ and $\beta_1(t), \beta_2(t) \in [0, 1]$. In the subsequent calculations, the $T_k^{loc}(t)$ and $T_{k,n}^{trt}(t)$ with the $E_k^{loc}(t)$ and $E_k^n(t)$, are normalized into the calculation.

4.2. Optimization problem

The system aim to maximize the system average utility for tasks that do not expire in the MEC environment. With this approach, the average TRT and the total energy consumption of the tasks are minimized. The QoE is also guaranteed under the constraints of the SEG model. The optimization problem is represented as

$$\max \frac{1}{V}\sum_{t=1}^{T}\sum_{k=1}^{K} U_k(t) \tag{13}$$

$$s.t. \quad y \le Y \tag{13a}$$

$$\beta_1(t), \beta_2(t) \in [0, 1] \tag{13b}$$

$$\partial_k(t) \in \{0, 1, \ldots, K\} \tag{13c}$$

$$\alpha_k^{sub}(t) + T_{k,n}^{trt}(t) \le \alpha_k^{ddl}(t). \tag{13d}$$

Constraint (13a) states that the currently allocated server computing node y is within the total number of nodes Y. Constraint (13b) denotes the dynamically changing weight factors with upper and lower bounds. Constraint (13c) means that the tasks select only one offloading location.

4.3. Analysis of optimization objective weights

To weigh the importance of optimization objectives in real MEC scenarios, it is important to set appropriate weight factors. Currently, some researchers use a subjective approach to set the weights of multiple optimization objectives. However, researchers cannot easily balance the importance of different weighting factors in a real-time changing environment. In addition, there are some limitations of fixed weights. Entropy has been widely used in engineering, sociology and economic fields [40]. The entropy weighting method determines objective weights based on the magnitude of the variability of the indicators. Assuming that there are d samples and p optimization objectives in the system, the calculation steps are as follows:

Step 1: Since the size of the optimization objectives is inconsistent, there is a necessity to normalize the objectives. In this model, normalize the data of each optimization objective by

$$y_{i,j} = \frac{x_{i,j} - \min\left\{x_{1,j}\cdots, x_{d,j}\right\}}{\max\left\{x_{1,j}\cdots, x_{d,j}\right\} - \min\left\{x_{1,j}\cdots, x_{d,j}\right\}}. \tag{14}$$

Step 2: For one optimization objective, the proportion of each sample to the total value is calculated, and the equation is expressed as

$$p_{i,j} = \frac{y_{i,j}}{\sum_{i=1}^{d} y_{i,j}} \; (i = 1, 2, \ldots, d; j = 1, 2, \ldots, p). \tag{15}$$

Step 3: The information entropy of the data is obtained according to

$$e_j = -\ln(d)^{-1}\sum_{i=1}^{d} p_{ij}\ln p_{ij}. \tag{16}$$

Step 4: The weight of each optimization objective is obtained by calculating the information redundancy from

$$\beta_j = \frac{D_j}{p - \sum_{j=1}^{p} D_j}\left(D_j = 1 - e_j\right). \tag{17}$$

In the model, the number of samples d is the number of computing nodes that can be selected. The number of optimization objectives p is 2, which are TRT and energy consumption, respectively. First, the system calculates the TRT and energy consumption values of the computing nodes that are available for offloading. One optimization objective corresponds to a set of $\left\{x_{1,j}\cdots, x_{d,j}\right\}$. Then, two weight values are obtained by calculating according to the above steps.

5. M2FD algorithm design

In this section, the M2FD algorithm is presented, and we introduce its theoretical background. In addition, the specific details of the M2FD are described.

5.1. Theoretical background of M2FD algorithm

5.1.1. The DQN algorithm

Currently, many issues arise in the study of traditional algorithms. These include the coupling relationships between computational and communication resources, and the problem of high-dimensional state data. These issues lead to the failure of traditional methods to obtain a better solution in effective time. RL belongs to the field of machine learning, which focuses on maximizing the cumulative reward and training the agent to take the most favorable action in different environments. The main components of RL are the agent, environment, state s, action a, and reward r. The agent selects an action to execute, causing the environment to change its state. The environment determines the strengths or weaknesses of the new state, and thus obtains a reward value, either positive or negative. During the interaction with the environment, the agent generates the optimal behavioral policy by continuously trial-and-error and maximizing the cumulative reward. However, it is difficult for RL to obtain the states corresponding to the current and next actions in the environment. Because RL is weak in perceiving the environment, it takes a lot of time to read a lot of environmental information. To solve the above problems and adapt to the large-scale network environment, it is a very effective decision to embed DL into RL. DL is a machine learning algorithm that relies on experience. It achieves optimization of the model by learning the deep nonlinear network structure and the essential features of the data set.

In combination with DL, DRL makes the system solve faster and with better performance.

DQN is an algorithm that belongs to DRL. The maximum expected gain that the algorithm can achieve by taking action a after observing a sequence of state spaces s under the strategy π is called the optimal action-value function, denoted as $Q^*(s, a)$. According to the Bellman equation, the formula for updating the action value function can be expressed as $Q^*(s,a) = r + \gamma \max_{a'} Q^*(s', a')$, where γ represents the degree of weakening of the future gain, namely the discount factor. In the DQN algorithm, a neural network is used to estimate the action-value function, which can be expressed as $Q(s, a; \theta) \approx Q^*(s, a)$. Therefore, the formula for calculating the *Q-target* network in DQN can be expressed as

$$Q^{target}(s, a; \theta) = r + \gamma \max_{a'} Q^{target}(s', a'; \theta'). \tag{18}$$

On the basis of Q-learning, DQN makes many improvements that enable the performance to be improved.

First, the problem of the high-dimensional state and action space is solved. The Q-learning uses a table to store the state and action space. However, with continuous high-dimensional states, the modifying and finding expenses of tables are extremely high. Therefore, DQN uses deep neural networks to fit this process. After the states are input into the neural network, a series of action values are computed. The action with the best result is selected using a greedy strategy. In addition, the model learns data features continuously and receives feedback for adjustment until the convergence.

Second, the target network is added to avoid the fluctuation problem during the training step. Two networks with identical structure but different parameters exist in the DQN. They are the *Q-evaluation* network, which updates parameters in real-time, and the *Q-target* network, which updates parameters in fixed iterations, respectively. In iteration i, the *Q-network* is trained by minimizing the loss function. The mean square error loss can be expressed as

$$L(\theta) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right)^2\right], \tag{19}$$

where the parameter θ_{i-1} of the previous iteration is fixed. After several iterations, the parameters of the *Q-evaluation* network are all copied to the *Q-target* network. This method makes the algorithm update stably and the error is reduced.

Third, an experience pool is used to improve training performance. The experiences are stored in the experience pool after they are obtained from explorations or actions. An experience can be represented by (s, a, r, s'). The experiences are randomly selected to update the network, which makes the correlation between experiences decrease. Meanwhile, each experience can be learned repeatedly, increasing the experience utilization. In addition, the diversity of experiences within a batch is increased and more environmental features are learned.

Finally, an ε-greedy strategy is used to improve the exploration rate. This strategy allows the system to have a certain probability of randomly selecting an action, denoted as

$$a = \begin{cases} \arg\max_a Q(s, a; \theta), \varepsilon \le e \le 1 \\ random, 0 \le e < \varepsilon \end{cases}, \tag{20}$$

where e $(0 \le e \le 1)$ is a random value. The ε value decreases with the increasing iterations, which ensures the accuracy of the model after convergence.

Based on the above DQN algorithm properties, the optimization problem is modeled as a Markov decision process (MDP). This is used to solve the task offloading and resource allocation problems under the two-layer MEC model. The pseudocode of the DQN algorithm is shown in Algorithm 1.

Algorithm 1: The DQN algorithm

```
   Input: Environment state $s$
   Output: Offloading action $a$
1  Initialize experience pool capacity $C_{ep}$;
2  Initialize Q-evaluation and Q-target network parameters;
3  for episode = 1, $E$ do
4      Initialize $s_1$ from environment;
5      for $t$ = 1, T do
6          With probability $\varepsilon$, choose a random action $a_t$;
            otherwise, choose $a_t = \arg\max_a Q(s, a; \theta)$;
7          Execute action $a_t$ and obtain reward $r_t$ from
            Q-evaluation;
8          Transfer the environment state to $s_{t+1}$;
9          Add experience quadruple $(s_t, a_t, r_t, s_{t+1})$ to
            experience pool;
10         Sample random mini-batch of transitions
            $(s_j, a_j, r_j, s_{j+1})$ from experience pool;
11         Calculate the value of the Q-target using
12             $y_j = \begin{cases} r_j, \text{for terminal at step } j+1 \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta'), \text{otherwise} \end{cases}$;
13         Perform a gradient descent step on
            $(y_j - Q(s_j, a_j, \theta))^2$;
14         Every $\upsilon$ steps, clone Q-evaluation parameters to
            Q-target;
15     end
16 end
17 return
```

5.1.2. Federated learning framework

Classical machine learning algorithms are trained with many samples to obtain a model with good performance. Meanwhile, task information from different devices is stored centrally. An example is the traditional DRL-based centralized framework. However, the security of the exposed data suffers from a huge threat. In addition, in some special environments, such as finance, healthcare and government, data are not allowed to be shared with the outside world. This results in data that cannot be stored centrally and a centralized approach to DRL model training is not achievable. To fit a variety of environments and protect device data privacy, FL is deployed. In the MEC system, the goal of FL is to achieve joint modeling of many devices and enhance the effectiveness of the model based on ensuring data privacy and security. First, FL applies techniques such as differential privacy, homomorphic encryption, and multiparty secure computing. Through these encryption mechanisms for parameter exchange, the risk of data leakage is greatly reduced. Second, for unreliable mobile devices, FL cancels some training rounds of the device to avoid reducing the overall model efficiency. Finally, the devices upload model parameters to the server, effectively reducing the bandwidth burden and communication costs.

In summary, we propose an M2FD algorithm. The algorithm employs an FL-based model training approach to secure the data. Meanwhile, the problem of multi-type tasks and heterogeneous processor resources is transformed into an MDP. The objective is to increase the system utility while reducing the task failure rate. In addition, M2FD uses the DQN algorithm in DRL. The DQN algorithm is used locally to generate offloading decisions. Each end device submits a task request and trains the model locally. The global model is aggregated and distributed in fixed rounds until the model converges. The better offloading scheme is obtained.

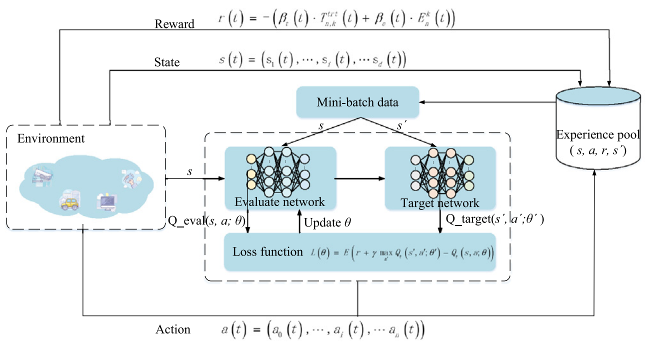

Reward
$r(t) = -\left(\beta_t(t)\cdot T_{n,k}^{trt}(t) + \beta_e(t)\cdot E_n^k(t)\right)$
State
$s(t) = \left(s_1(t),\cdots,s_i(t),\cdots s_d(t)\right)$
Mini-batch data
s
s′
Environment
Experience pool
(s, a, r, s′)
s
Evaluate network
Target network
Q_eval(s, a; θ)
Update θ
Q_target(s′, a′; θ′)
Loss function $L(\theta) = E\left(r + \gamma \max_{a'} Q_t(s', a'; \theta') - Q_t(s, a; \theta)\right)$
Action
$a(t) = \left(a_0(t),\cdots,a_i(t),\cdots a_n(t)\right)$

Fig. 2. The process of local training.

5.2. Training process of DQN model based on the FL

5.2.1. Process of local training

The DRL agent based on the DQN algorithm is set at each end device. For each device, the details of the DQN algorithm are shown in Fig. 2. The system optimization problem is translated into an MDP presented as follows, which includes the state space, action space and reward function.

State space. At the beginning of each time slot t, the device sends a request for a computational task to the local model. Meanwhile, the status information of all available computing nodes on the server and locally is collected. The system cost of all available nodes and the type of task are defined as the state space. Thus, the state space located in time slot t can be described as

$$s(t) = \left(s_0(t), \ldots, s_i(t), \ldots, s_n(t), \alpha_k^{tp}(t)\right), \\ s_i(t) = \begin{cases} \beta_1(t)\, T_k^{loc}(t) + \beta_2(t)\, E_k^{loc}(t), i = 0 \\ \beta_1(t)\, T_{i,k}^{trt}(t) + \beta_2(t)\, E_k^{i}(t), i \neq 0 \end{cases}. \tag{21}$$

Action space. At time t, the system decides the task offloading location based on the task information and resource allocation. Local and the edge of the computing nodes can be selected location of offloading action. The action space is all optional nodes. Therefore, the action space can be represented as

$$a(t) = \left(a_0(t), \ldots, a_i(t), \ldots a_n(t)\right), \\ a_i(t) \in \begin{cases} \{0, 1\}, i = 0 \\ \{0, 1, 2, 3\}, i \neq 0 \end{cases}, \tag{22}$$

where a_0 represents whether the task is computed locally, and a_1 to a_n represent the task is offloaded to the computing node of the edge server. When $i \neq 0$, $a_i(t) = 1$ means that the task is computed at the ith server integrated CPU node, $a_i(t) = 2$ represents that it is computed at the GPU node, and $a_i(t) = 3$ indicates that it is computed at the FPGA node. Assuming that the nth server CPU computing node is selected and the action space is denoted as $a(t) = (0, 0, \ldots, 0, 1)$.

Reward function. After performing the action $a(t)$ in the current state $s(t)$, the reward function $r(t)$ is used to calculate the reward $r(t+1)$ for the next time. The function defines the reward obtained by leaving the state $s(t)$, namely $s(t)$ corresponds to $r(t+1)$. To reduce the system cost and increase the system utility, the negative of the weighted sum of the TRT and energy consumption is used as the reward, which is denoted as

$$r(t) = -\left(\beta_1(t)\, T_{k,n}^{trt}(t) + \beta_2(t)\, E_k^n(t)\right), r(t) \in [-1, 0]. \tag{23}$$

5.2.2. Process of model aggregation and distribution

After the K devices are trained for ϕ rounds, the local models $W_1(t), W_2(t), \ldots, W_K(t)$ are sent to the node responsible for aggregation at the edge at time t. Next, the aggregation node receives and updates the models. In this model, the heterogeneous computing nodes in the MEC are considered, mainly the differences in the node computation rates in the devices. The state information in the input space of the DQN model is affected by the heterogeneity of the devices, i.e., $s(t)$ of Eq. (21). In practice, the weighted sum of the response time and energy consumption required by the computing nodes differs significantly due to the large difference in computing power between local and edge-side servers. Therefore, the impact of local differences in node computing power during model aggregation is small. The new global model $W_g(t+1)$ is constructed by using the federated averaging, which can be represented as

$$W_g(t+1) = \frac{1}{K}\sum_{k=1}^{K} W_k(t). \tag{24}$$

Finally, the global model parameters are distributed to the all devices, which can be expressed as

$$W_k(t+1) = W_g(t+1). \tag{25}$$

The above three steps are one aggregation process, and the model is trained iteratively according to this process until it converges. The M2FD algorithm for the FL training process is summarized in Algorithm 2.

6. Performance evaluation

In this section, we first introduce the platform and environment for the experiments. Then, we make a suitable choice of hyperparameters for the experiments. Finally, we compare and analyze the performance in different environments for FL-based training and traditional distributed-based training.

6.1. Experimental settings

The Python and TensorFlow are employed in the implementation of the M2FD algorithm. In addition, we use a Cloudsim to simulate the process of task offloading and resource allocation. The Cloudsim platform supports a customized network topology, computing node resources, and task offloading policy. In the experiment, the arrival time of the tasks is based on a Poisson distribution within a certain range; the sizes of the tasks are distributed normally within a certain range [41]. The main experimental parameters are described in Table 2, where the γ_n^{pro} array represents the computation frequencies of the four task types corresponding to the processor in the order of text, image, audio, and video [38].

6.2. Hyperparametric experiment

In the M2FD algorithm, there are many important hyperparameters that determine the final performance of the algorithm. These include the activation function, batch size and discount factor.

The activation function mainly performs a non-linear transformation of the data, which solves the problem of the lack of classification ability of linear models. When complex data are input, it can better represent the non-linear and complex function mapping between input and output. This makes the neural network more powerful. The most frequently used activation functions are softplus, sigmoid, tanh, and relu, which can be denoted as

$$softplus(x) = \log\left(1 + e^x\right), \tag{26}$$

Algorithm 2: The M2FD algorithm

```
Input: Task set and device states
Output: Task average response time, total energy consumption and failure rate
1  Edge server side: Initialize the global model with random parameter values $W_g(0)$ at time $t = 0$;
2  End device side: Download $W_g(0)$ from the edge server and let $W_k(0) = W_g(0)$, $(k = 1, 2, \cdots, K)$;
3  for $t$ = 1, $T$ do
4      End device side:
5      for each end device $k \in K$ in parallel do
6          Download $W(t)$ from edge;
7          Let $W_k(t) = W_g(t)$;
8          for each task generated by device $k$ do
9              Select a suitable computing node for the task based on $W_k(t)$ with Algorithm 1;
10             if $\alpha_k^{sub}(t) + T_{k,n}^{trt}(t) \leq \alpha_k^{ddl}(t)$ is satisfied at the selected computing node then
11                 Execute or offloading the task;
12                 Calculate the task response time and energy consumption;
13             else
14                 The task failure;
15             end
16             Train the local model $W_k(t)$;
17         end
18         Upload the trained model parameters $W_k(t+1)$ to the edge;
19     end
20     Edge server side:
21     Receive local models of all devices;
22     Use the federated averaging to construct global model $W_g(t+1)$ according to $W_g(t+1) = \frac{1}{K}\sum_{k=1}^{K} W_k(t)$;
23     Distribute the global model;
24 end
25 return
```

Table 2
Experimental simulation parameters.

Parameters	Value
B (MHz)	2.0
N_0 (dBm/Hz)	−174
P_k (W)	0.5
T (s)	10^4
γ_k (GHz)	Unif(4,5)
γ_n^{cpu} (GHz)	[Unif(9,10), Unif(7,9), Unif(6,8), Unif(6,7)]
γ_n^{fpga} (GHz)	[Unif(8,10), Unif(8,10), Unif(8,10), Unif(7,8)]
γ_n^{gpu} (GHz)	[Unif(8,9), Unif(9,10), Unif(7,9), Unif(7,8)]
η_k	10^{-28}

$$sigmoid(x) = \frac{1}{1+e^{-x}}, \tag{27}$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{28}$$

and

$$relu(x) = \begin{cases} x, x \geq 0 \\ 0, x < 0 \end{cases}, \tag{29}$$

respectively. The softplus, sigmoid, and tanh functions contain exponential operations, and the derivation involves division when performing backward error propagation. This causes the values in the input layer to become increasingly smaller as they approach the output layer, causing the gradient to vanish. In contrast, the relu is better suited for backward propagation, eliminating the problem. We compare four convergences, as shown in Fig. 3. Fig. 3(a) is the loss value for each episode. Softplus, sigmoid and tanh oscillations are larger than relu during the iterations. Starting from the 1750th episode, the model converges, and the loss value tends to 0. Fig. 3(b) is the total loss value until the current episode. In general, the total loss of relu is smaller than the other three activation functions from the 750th episode, and the subsequent convergence is at the 125th episode. Therefore, setting the activation function to relu is the most appropriate in this experiment.

The batch size is an important parameter that indicates the number of data in each batch. Each episode is fed with batch size samples for training. Moreover, the average loss value of them is calculated to update the model parameters. The batch size determines the direction of gradient descent. In addition, as the batch size increases, the direction of descent is determined more accurately, and the training oscillations caused are lower. The batch size is set to 16, 32, 64 and 128 in the M2FD algorithm, respectively, as shown in Fig. 4. Fig. 4(a) indicates the trend of loss for each episode, and the oscillation of batch size from 16 to 128 gradually decreases. This is because within a certain range, the smaller the batch size is, the larger the error in the gradient valuation, resulting in a larger oscillation. Fig. 4(b) shows the trend of the total loss value until the current episode. In

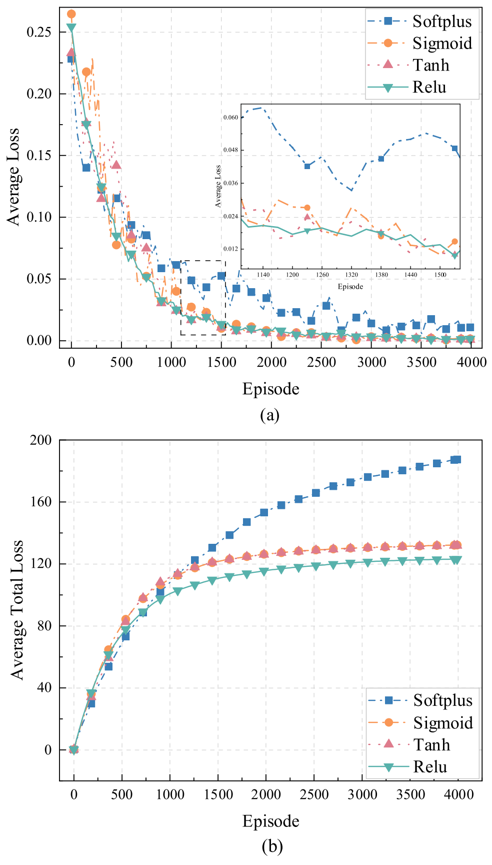


Fig. 3. Convergence of different activation functions. (a) Trend of the average loss. (b) Trend of total loss.

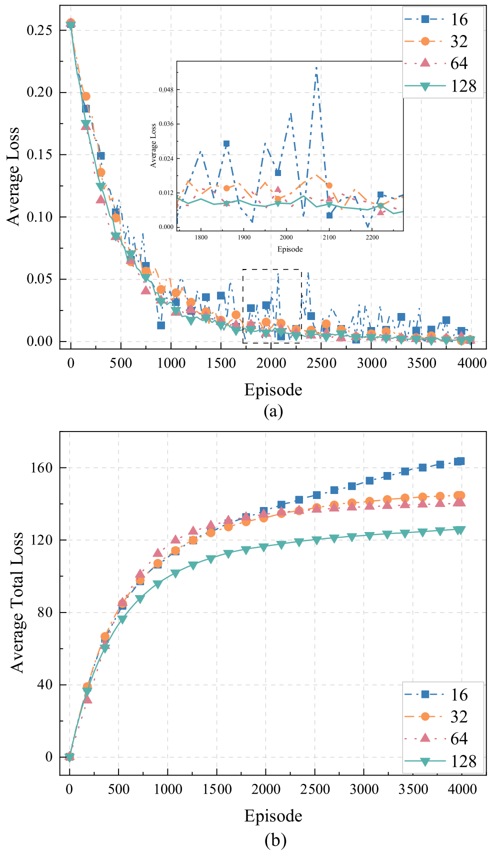


Fig. 4. Convergence of different batch sizes. (a) Trend of the average loss. (b) Trend of total loss.

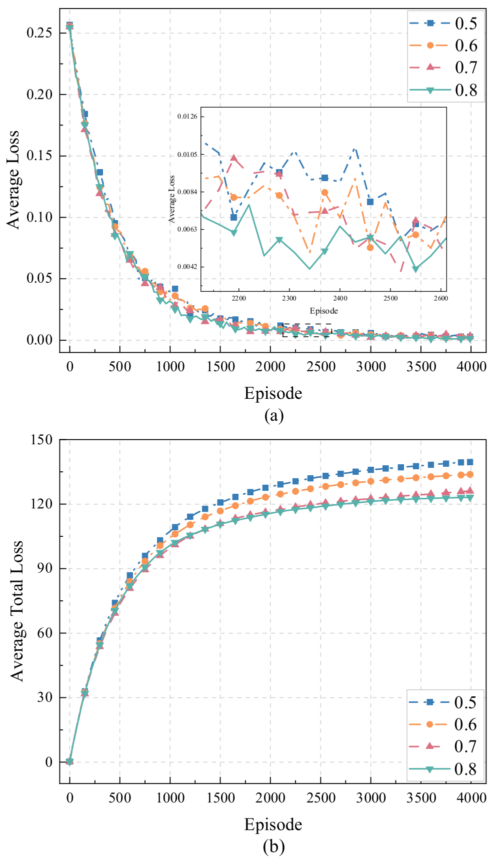


Fig. 5. Convergence of different discount factors. (a) Trend of the average loss. (b) Trend of total loss.

episode 375, the total loss of batch size 128 is smaller than that of the other three batch sizes, and the gap subsequently remains larger. Therefore, batch size 128 is used as the experimental hyperparameter in the performance experiments.

The main effect of the discount factor γ is to reduce the influence of future rewards on current decision-making. Since future rewards contain uncertainty, rewards are expected to be received in the present rather than the future. In the extreme case, $\gamma = 0$ means that the system only focuses on the current state; $\gamma = 1$ implies that the system considers the present and the future equally important. Choosing the right discount factor can avoid the reward value from tending to infinity. In the experiments, a common range of discount factors from 0.5 to 0.8 is observed for convergence trends, as shown in Fig. 5. The loss calculated for each episode is presented in Fig. 5(a). The oscillations of the model are obviously reduced after the choice of activation function and batch size. However, different discount factors affect the final convergence value of the model. The convergence value reaches approximately 0.005 for a discount factor of 0.8, while 0.5, 0.6, and 0.7 converge to approximately 0.009, 0.008, and 0.007, respectively. Fig. 5(b) demonstrates the total loss as of the current episode. At the beginning of the model training, the discount factors 0.7 and 0.8 are better than the other two values. At the 1500th episode, the total loss of the discount factor 0.8 starts to be smaller than the total loss of 0.7. Therefore, the appropriate value of the discount factor is 0.8 in the experiment.

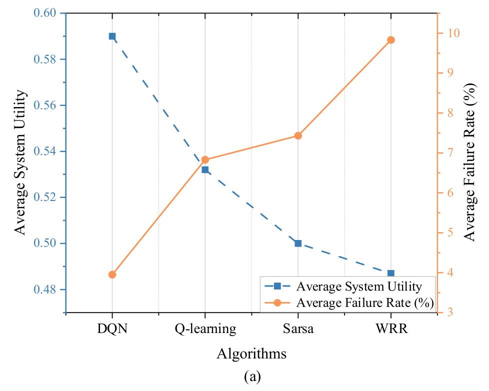
The main hyperparameters in the experiment are summarized in Table 3.

Table 3
Main hyperparameters.

Hyperparameters	Value
The activation function	Relu
The batch size	128
The discount factor γ	0.8
The experience pool capacity C_{ep}	100
The greedy probability ε	0.5
The layers of neural network	4
The learning rate α	1e−3
The loss function	Mean-square error

6.3. Performance experiments

The performance experiments first compare the performance of the DQN, Q-leaning, Sarsa and weight round robin (WRR) algorithms under a centralized model. DQN is the RL algorithm used in M2FD. Q-learning and Sarsa are also RL algorithms, but the main difference between them and DQN is that they use a Q-table to calculate Q values instead of a neural network. WRR is a classic task offloading algorithm. Then, to evaluate the performance of the proposed M2FD algorithm, the centralized model training-based algorithm is used for comparison. The model differs from M2FD mainly in the training method. It is trained by using a DRL agent that collects data from all devices at each time slot for training. The centralized algorithm is based on the DQN algorithm in DRL to find an appropriate system policy. A suitable offloading node is obtained after inputting information about the task. However, the centralized algorithm has a high amount of



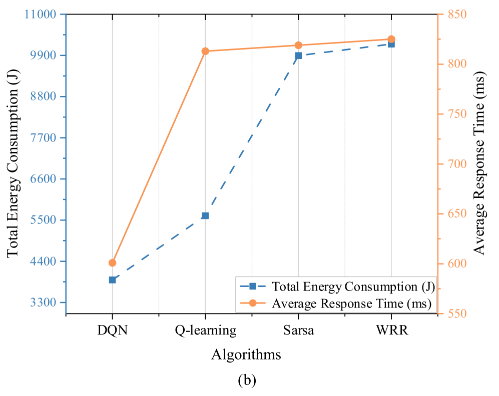


Fig. 6. Comparison of performance indicators for different algorithms. (a) Average system utility and task failure rate. (b) Total energy consumption and average response time.

data transmission and high cost of communication between the devices and the edge. In addition, the data are exposed to the transmission process, which may lead to privacy disclosures. In contrast, the FL-based training approach uses the data from the device to train the model locally and sends the model to the edge for aggregation.

6.3.1. Validation of the DQN model

In this experiment, four algorithms are compared for each performance indicator. The task size of the device in the experiment is scenario 2 in Table 4, and the task density is 30 tasks generated in 100 s. The results of the experiments are presented in Fig. 6.

Figs. 6(a)–(b) demonstrate that the WRR algorithm is weaker than the other three algorithms for all types of performance indicators. This is because the algorithm only ranks the task queues by weighting them, making the offloading decision highly uncertain. Q-learning and Sarsa showed better performance than WRR but weaker performance than DQN, because DQN pre-processes a larger number of tasks and uses network model training methods such as experience replay and target networks. Moreover, the DQN algorithm is able to combine neural networks to analyze the MEC environment and the task states to obtain a good offloading decision.

6.3.2. Task size

In this experiment, the two models are compared for each performance indicator as the task size increases under the SEG constraint, including the average system utility, average task failure rate, total energy consumption, and average task response time. The sizes of the various types of tasks in the experiment are shown in Table 4, with a task density of 30 tasks generated in 100 s. There are 3 devices in the system. The ratios of task types and task sizes generated by each device were consistent. The experimental results are displayed in Fig. 7.

Table 4
Different scenarios for each type of task size (kb).

Scenario	Text	Image	Audio	Video
I	256	512	768	1024
II	512	1024	1536	2048
III	768	1536	2304	3072
IV	1024	2048	3072	4096

Figs. 7(a)–(d) represent the trends of the four indicators when the task sizes of the four types are from small to large. At different task sizes, the system utility, energy consumption, task response time, and task failure rate are optimized by 15%, 33%, 5%, and 8% on average, respectively. In Fig. 7(a), the average system utility is in a fluctuating state with no significant upward or downward trend when the task size increases. This indicates that the task size has no obvious effect on the system utility value. However, the total energy consumption and average response time increase with increasing task size in Figs. 7(b)–(c). This is because the larger the task size is, the longer the processing time at the computing node and the transmission time in the channel. The slower the tasks in the queue of each computing node are computed, the longer the waiting time for new arrivals. Similarly, the larger the size is, the higher the energy consumption of the task being processed for transmission and computation. In Fig. 7(d), the average failure rate of tasks rises. The reason is that the resources of edge computing nodes are fixed, and the computational resources consumed by a single task increase. This leads to more tasks being discarded due to not having enough resources to process them before the deadline. With the same task types and sizes scaled for each device, the M2FD model is slightly better trained than the centralized training model. This is because during the training process of the M2FD model, the state of the computing nodes and the task waiting queue information at the edge are synchronized to the locals, and the local training agent dynamically adjusts the weights between the two objectives in real time using the entropy weighting method. This makes the system more efficient.

6.3.3. Task density

In this experiment, as the task density increases, the two algorithms are compared through four performance indicators. The density of tasks in the experiment is 20, 25, 30, 35, 40, and 45 tasks generated in 100 s. There are 3 devices in the system. The task size is scenario II in Table 4. The ratio of task types and task sizes generated for each device is consistent. The results are shown in Fig. 8.

An increase in task density is an increase in the number of tasks generated by the device in 100 s, which results in more total tasks and a greater number of model trainings. The system utility, energy consumption, task response time, and task failure rate are optimized by 15%, 25%, 2%, and 6% on average, respectively, at different task densities. In Fig. 8(a), the different model characteristics of each device are learned in the M2FD algorithm. The centralized model is not better able to allocate computing nodes and resources based on environmental information. This results in a minor upward trend in its task average system utility and is slightly better than that of the centralized model. In addition, as the task density increases, the total energy consumption for task transfer and computation increases accordingly. Thus, the total energy consumption shows an increasing trend in Fig. 8(b). As more energy is consumed, the entropy weighting method adjusts the weight of the model optimization objective according to the environment. In addition, due to the consistent size range of the

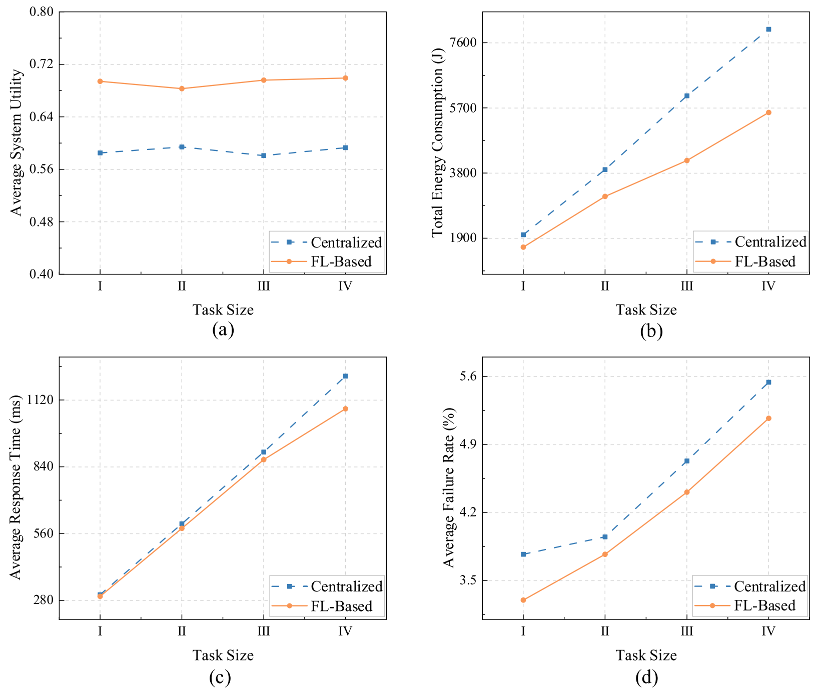


Fig. 7. Comparison of performance indicators for different task sizes. (a) Average system utility. (b) Total energy consumption. (c) Average response time. (d) Task failure rate.

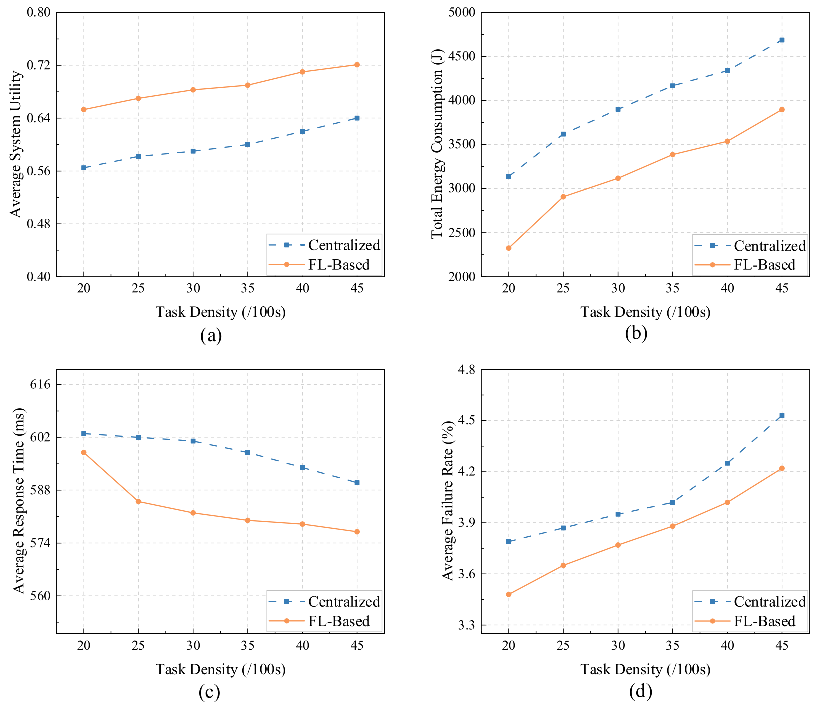


Fig. 8. Comparison of performance indicators for different task densities. (a) Average system utility. (b) Total energy consumption. (c) Average response time. (d) Task failure rate.

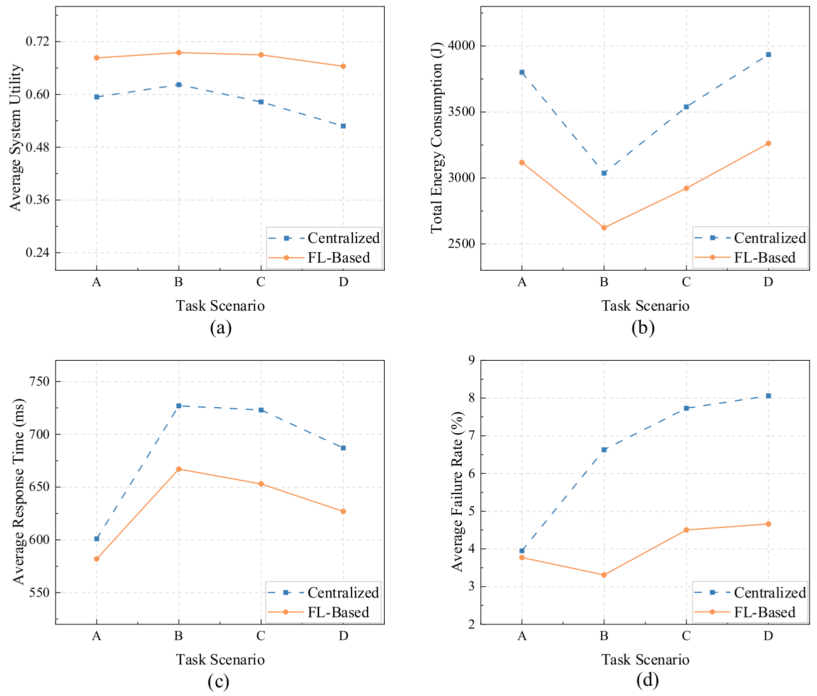


Fig. 9. Comparison of performance indicators for different task distributions. (a) Average system utility. (b) Total energy consumption. (c) Average response time. (d) Task failure rate.

Table 5
Distribution of the sizes and types of tasks in each device.

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

generated tasks, a minor downward trend in the average TRT is indicated in Fig. 8(c), with a time difference of less than 1 s. In Fig. 8(d), the task failure rate gradually increases. This is because the increase in task density leads to an increase in the number of tasks that the system needs to process at the same time. However, the overall resources of the system are fixed. This leads to a decrease in the resources available for each task and an increase in the task failure rate.

6.3.4. *Task scenario*

In this experiment, we simulated a data environment with a non-independent identical distribution in FL. The four cases of task data distribution are shown in Table 5. Trends in average system utility, total energy consumption, task failure rate, and response time are shown. The task density is 30 tasks generated in 100 s. The task size is scenario II in Table 4. The experimental results are shown in Fig. 9.

The variation in task scenarios illustrates the difference in task type and size for each scenario. The system utility, energy consumption, task response time, and task failure rate are optimized by 17%, 19%, 8%, and 62% on average, respectively, under different task scenarios. Fig. 9(a) represents the average system utility under different task data distribution scenarios. Compared to A, the average utility of both algorithms decreases under the scenarios B, C, and D. This indicates that under non-independently and identically distributed task scenarios B, C, and D, aggregating the model using only the federated average algorithm results in a partial loss of model properties. This leads to a decrease in system utility. Figs. 9(b) and 9(c) show the values of energy consumption and response time for different data distribution scenarios. Under the scenario B, the total energy consumption of the system is the lowest, but the average response time of the tasks is the highest. This is followed by an increase in energy consumption and a decrease in response time in scenarios C and D. The response time varies in the range of 1 s. The reason is that the sum of the weights of response time and energy consumption in the model optimization objective is 1. The task size and type are changing in different task scenarios, and the optimization of both are mutually exclusive. Fig. 9(d) shows the different distribution scenarios of the task failure rate. The difference in task failure rates between the two models is small when both the task size and type proportion are consistent. However, there is an overall increasing trend. This indicates that when the task size is constant, the larger the task, the higher the task failure rate. And the models are less efficient for non-independently and identically distributed data than for independently and identically distributed data.

In all experiments, the FL-based M2FD algorithm only slightly outperformed the algorithm in the other framework. The experiments show that the FL-based M2FD algorithm slightly outperforms the algorithms of other frameworks. The result performs better in the MEC environment where the number of user terminals is small. And the optimization effect of independent and identically distributed task data is better than that of non-independent and identically distributed data. In summary, the M2FD algorithm optimizes the system utility, response time, energy consumption, and statistics of the task failure rate in MEC environments considering different types of task and edge-side integrated processors by combining the DQN algorithm and entropy weight method. In addition, FL does not transfer the task data but chooses to transfer the model during the training of the model. Therefore, the M2FD algorithm is able to protect user privacy and create a data security environment. It provides a good choice for industries with special security requirements.

7. Conclusion

In this paper, a multi-type task offloading and resource allocation problem is considered in the two-layer MEC model. To obtain the higher system utility and lower cost while preserving user data privacy, the M2FD algorithm is proposed. In addition, to avoid the influence of subjectivity, the weight values of the optimization objectives are calculated using the entropy weight method. The optimization problem is modeled as an MDP. In the FL training framework, the intelligent DRL method is employed to assign the appropriate computing node for the task. The experimental results show that the M2FD algorithm can improve the system utility and reduce the system cost while safeguarding the user data privacy. In the future, it is promising to investigate blockchain-enabled edge computing architectures. Combining FL, blockchain, and more efficient DRL methods to address user needs in a more complex MEC environment.

CRediT authorship contribution statement

Zhao Tong: Idea, Survey, Optimization, Reviewing. **Jiake Wang:** Survey, Implement, Experiments, Writing – original draft. **Jing Mei:** Optimization, Reviewing. **Kenli Li:** Optimization, Reviewing. **Wenbin Li:** Suggestions, Reviewing. **Keqin Li:** Suggestions, Reviewing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

Acknowledgments

The authors thank the editors and reviewers for their insightful comment and valuable suggestions. This work was supported by the Program of National Natural Science Foundation of China (Grant No. 62072174, 61502165), Distinguished Youth Science Foundation of Hunan Province, China (Grant No. 2023JJ10030), National Natural Science Foundation of Hunan Province, China (Grant No. 2022JJ40278, 2020JJ5370), Scientific Research Fund of Hunan Provincial Education Department, China (Grant No. 22A0026). All authors approved the version of the manuscript to be published.

References

[1] R. Lohiya, A. Thakkar, Application domains, evaluation data sets, and research challenges of IoT: A Systematic Review, IEEE Internet Things J. 8 (11) (2020) 8774–8798.

[2] Cisco Annual Internet Report (2018–2023) White Paper, Cisco, San Jose, CA, USA, 2020.

[3] B.B. Sinha, R. Dhanalakshmi, Recent advancements and challenges of Internet of Things in smart agriculture: A survey, Future Gener. Comput. Syst. 126 (2022) 169–184.

[4] A. Uprety, D.B. Rawat, Reinforcement learning for IoT security: A comprehensive survey, IEEE Internet Things J. 8 (11) (2020) 8693–8706.

[5] Q. Li, S. Wang, A. Zhou, X. Ma, F. Yang, A.X. Liu, QoS driven task offloading with statistical guarantee in mobile edge computing, IEEE Trans. Mob. Comput. 21 (1) (2020) 278–290.

[6] Z. Ma, S. Zhang, Z. Chen, T. Han, Z. Qian, M. Xiao, N. Chen, J. Wu, S. Lu, Towards revenue-driven multi-user online task offloading in edge computing, IEEE Trans. Parallel Distrib. Syst. 33 (5) (2021) 1185–1198.

[7] Y. Miao, G. Wu, M. Li, A. Ghoneim, M. Al-Rakhami, M.S. Hossain, Intelligent task prediction and computation offloading based on mobile-edge cloud computing, Future Gener. Comput. Syst. 102 (2020) 925–931.

[8] T. Zhu, T. Shi, J. Li, Z. Cai, X. Zhou, Task scheduling in deadline-aware mobile edge computing systems, IEEE Internet Things J. 6 (3) (2018) 4854–4866.

[9] J.-B. Wang, H. Yang, M. Cheng, J.-Y. Wang, M. Lin, J. Wang, Joint optimization of offloading and resources allocation in secure mobile edge computing systems, IEEE Trans. Veh. Technol. 69 (8) (2020) 8843–8854.

[10] H. Xu, W. Huang, Y. Zhou, D. Yang, M. Li, Z. Han, Edge computing resource allocation for unmanned aerial vehicle assisted mobile network with blockchain applications, IEEE Trans. Wirel. Commun. 20 (5) (2021) 3107–3121.

[11] X. Deng, J. Liu, L. Wang, Z. Zhao, A trust evaluation system based on reputation data in mobile edge computing network, Peer-To-Peer. Netw. Appl. 13 (5) (2020) 1744–1755.

[12] R.A. Khalil, N. Saeed, M. Masood, Y.M. Fard, M.-S. Alouini, T.Y. Al-Naffouri, Deep learning in the industrial internet of things: Potentials, challenges, and emerging applications, IEEE Internet Things J. 8 (14) (2021) 11016–11040.

[13] X. Qiu, W. Zhang, W. Chen, Z. Zheng, Distributed and collective deep reinforcement learning for computation offloading: A practical perspective, IEEE Trans. Parallel Distrib. Syst. 32 (5) (2020) 1085–1101.

[14] F. Jiang, K. Wang, L. Dong, C. Pan, K. Yang, Stacked autoencoder-based deep reinforcement learning for online resource scheduling in large-scale MEC networks, IEEE Internet Things J. 7 (10) (2020) 9278–9290.

[15] J. Konečnỳ, H.B. McMahan, F.X. Yu, P. Richtárik, A.T. Suresh, D. Bacon, Federated learning: Strategies for improving communication efficiency, 2016, arXiv preprint arXiv:1610.05492.

[16] Z. Tong, X. Deng, F. Ye, S. Basodi, Y. Pan, Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment, Inform. Sci. 537 (2020) 116–131.

[17] Y. Guo, R. Zhao, S. Lai, L. Fan, X. Lei, G.K. Karagiannidis, Distributed machine learning for multiuser mobile edge computing systems, IEEE J. Sel. Top. Signal Process. (2022) http://dx.doi.org/10.1109/JSTSP.2022.3140660.

[18] Z. Tong, J. Cai, J. Mei, K. Li, K. Li, Dynamic energy-saving offloading strategy guided by Lyapunov optimization for IoT devices, IEEE Internet Things J. (2022) http://dx.doi.org/10.1109/JIOT.2022.3168968.

[19] J. Zhou, X. Zhang, Fairness-aware task offloading and resource allocation in cooperative mobile edge computing, IEEE Internet Things J. 9 (5) (2022) 3812–3824.

[20] L. Tan, Z. Kuang, L. Zhao, A. Liu, Energy-efficient joint task offloading and resource allocation in OFDMA-based collaborative edge computing, IEEE Trans. Wirel. Commun. 21 (3) (2021) 1960–1972.

[21] Y. Dong, S. Guo, Q. Wang, S. Yu, Y. Yang, Content caching-enhanced computation offloading in mobile edge service networks, IEEE Trans. Veh. Technol. 71 (1) (2021) 872–886.

[22] G. Yang, L. Hou, X. He, D. He, S. Chan, M. Guizani, Offloading time optimization via Markov decision process in mobile-edge computing, IEEE Internet Things J. 8 (4) (2020) 2483–2493.

[23] A. Asheralieva, D. Niyato, Bayesian reinforcement learning and bayesian deep learning for blockchains with mobile edge computing, IEEE Trans. Cognit. Commun. Netw. 7 (1) (2020) 319–335.

[24] L. Lei, Y. Tan, K. Zheng, S. Liu, K. Zhang, X. Shen, Deep reinforcement learning for autonomous internet of things: Model, applications and challenges, IEEE Commun. Surv. Tutor. 22 (3) (2020) 1722–1760.

[25] W. Chen, X. Qiu, T. Cai, H.-N. Dai, Z. Zheng, Y. Zhang, Deep reinforcement learning for Internet of Things: A comprehensive survey, IEEE Commun. Surv. Tutor. 23 (3) (2021) 1659–1692.

[26] M. Shurrab, S. Singh, R. Mizouni, H. Otrok, IoT sensor selection for target localization: A reinforcement learning based approach, Ad Hoc Netw. 134 (2022) 102927.

[27] S. Chen, J. Chen, Y. Miao, Q. Wang, C. Zhao, Deep reinforcement learning-based cloud-edge collaborative mobile computation offloading in industrial networks, IEEE Trans. Signal Inf. Process. Netw. 8 (2022) 364–375.

[28] X. Zhu, Y. Luo, A. Liu, M.Z.A. Bhuiyan, S. Zhang, Multiagent deep reinforcement learning for vehicular computation offloading in IoT, IEEE Internet Things J. 8 (12) (2020) 9763–9773.

[29] Z. Cao, P. Zhou, R. Li, S. Huang, D. Wu, Multiagent deep reinforcement learning for joint multichannel access and task offloading of mobile-edge computing in industry 4.0, IEEE Internet Things J. 7 (7) (2020) 6201–6213.

[30] R. Yu, P. Li, Toward resource-efficient federated learning in mobile edge computing, IEEE Netw. 35 (1) (2021) 148–155.

[31] Y. Lu, X. Huang, Y. Dai, S. Maharjan, Y. Zhang, Differentially private asynchronous federated learning for mobile edge computing in urban informatics, IEEE Trans. Ind. Inform. 16 (3) (2020) 2134–2143.

[32] C. Feng, Z. Zhao, Y. Wang, T. Quek, M. Peng, On the design of federated learning in the mobile edge computing systems, IEEE Trans. Commun. 69 (9) (2021) 5902–5916.
[33] Z. Ji, L. Chen, N. Zhao, Y. Chen, G. Wei, F.R. Yu, Computation offloading for edge-assisted federated learning, IEEE Trans. Veh. Technol. 70 (9) (2021) 9330–9344.
[34] S.S. Shinde, A. Bozorgchenani, D. Tarchi, Q. Ni, On the design of federated learning in latency and energy constrained computation offloading operations in vehicular edge computing systems, IEEE Trans. Veh. Technol. 71 (2) (2021) 2041–2057.
[35] S. Wang, M. Chen, C. Yin, W. Saad, C.S. Hong, S. Cui, H.V. Poor, Federated learning for task and resource allocation in wireless high-altitude balloon networks, IEEE Internet Things J. 8 (24) (2021) 17460–17475.
[36] C. Guo, L. Liang, G.Y. Li, Resource allocation for vehicular communications with low latency and high reliability, IEEE Trans. Wirel. Commun. 18 (8) (2019) 3887–3902.
[37] J. Huang, S. Li, Y. Chen, Revenue-optimal task scheduling and resource management for IoT batch jobs in mobile edge computing, Peer-To-Peer Netw. Appl. 13 (5) (2020) 1776–1787.
[38] Y. Mao, J. Zhang, S. Song, K.B. Letaief, Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems, IEEE Trans. Wirel. Commun. 16 (9) (2017) 5994–6009.
[39] W. Hu, G. Cao, Quality-aware traffic offloading in wireless networks, IEEE Trans. Mob. Comput. 16 (11) (2017) 3182–3195.
[40] Q. Ding, Y.-M. Wang, Intuitionistic fuzzy TOPSIS multi-attribute decision making method based on revised scoring function and entropy weight method, J. Intell. Fuzzy Systems 36 (1) (2019) 625–635.
[41] J. de Lope, D. Maravall, Y. Quinonez, Self-organizing techniques to improve the decentralized multi-task distribution in multi-robot systems, Neurocomputing 163 (2015) 47–55.

Zhao Tong received the Ph.D. degree in computer science from Hunan University, Changsha, China in 2014. He was a visiting scholar at the Georgia State University from 2017 to 2018. He is currently an associate professor at the College of Information Science and Engineering of Hunan Normal University, 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-TPDS, Information Sciences, FGCS, NCA, and JPDC, PDCAT, etc. He is a senior member of the China Computer Federation (CCF) and a Member of the IEEE.

Jiake Wang received the B.E. degree from Hunan Institute of Science and Technology, Yueyang, China in 2021. She is currently pursuing a M.S. degree in the College of Information Science and Engineering of 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. in computer science from Hunan University, China, in 2015. She is currently an assistant professor in the College of Information Science and Engineering at Hunan Normal University. Her research interests include parallel and distributed computing, cloud computing, etc. 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*, *Journal of Supercomputing*.

Kenli Li received the Ph.D. degree in computer science from Huazhong University of Science and Technology, in 2003. He was a visiting scholar at University of Illinois at Urbana-Champaign from 2004 to 2005. He is currently a full professor of computer science and technology at Hunan University and deputy director of National Supercomputing Center in Changsha. His major research includes 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*, *Artificial Intelligence*. He is a senior member of IEEE and an outstanding member of CCF.

Wenbin Li received the B.S. degree in computer science and technology from Hunan Normal University, Changsha, China, in 2003, the M.S. degree in computer applications technology from Changsha University of Science and Technology, Changsha, China in 2006, and the Ph.D. degree in control engineering from Central South University, Changsha, China, in 2020. His research interests include industrial process control, evolutionary computation, and multi-objective optimization.

Keqin Li is a SUNY Distinguished Professor of computer science. His current research interests include parallel computing and high-performance computing, distributed 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–physical systems. He has published over 510 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Services Computing*, *IEEE Transactions on Sustainable Computing*. He is an IEEE Fellow.