Journal of Computers Vol. 36 No. 3, June 2025, pp. 173-190
https://doi.org/10.63367/19911599202506360 3012

Research on Computing Task Offloading Mechanism Based on Cloud-Edge Collaboration

Ze-Yun Tong[1] and Jun Wang[1,2*]

[1] College of Communication and Information Engineering, Nanjing University of Post and Telecommunications, Nanjing, 210003, Jiangsu, China

{wang_jun, 1022010114}@njupt.edu.cn

[2] Jiangsu Key Laboratory of Intelligent Information Processing and Communication Technology (Nanjing University of Posts and Telecommunications), Nanjing, 210003, Jiangsu, China

Received 6 June 2024; Revised 24 October 2024; Accepted 16 December 2024

Abstract. Mobile edge computing (MEC) as an emerging and prospective computing paradigm, can significantly enhance computation capability and save energy of smart mobile devices (SMDs) by offloading computation-intensive tasks from resource-constrained SMDs to the resource-rich cloud. Most edge computing offloading algorithms only consider latency and energy consumption, while neglecting the load balancing on edge servers. Therefore, in this paper a load balancing model is introduced. The load balance model is based on the utilization of edge servers. At the same time, in order to achieve load balancing goals, cloud-edge collaboration is needed. In addition, the traditional particle swarm optimization algorithm cannot be applied to discrete solution spaces, an improved multi-dimensional discrete particle swarm optimization (MDPSO) algorithm is proposed to solve multi-objective optimization problems based on our load balancing model. At the same time, the improved algorithm MDPSO can reduce the degree of manual intervention in the selection process of learning factors', while accelerating the convergence speed of the algorithm. Finally, a simulation platform was built to verify the algorithm, and the simulation results demonstrate that the improved algorithm effectively reduces network latency and energy consumption, achieving global optimization while maintaining load balance on edge servers.

Keywords: edge computing, cloud-edge collaborate, task offloading, load balancing, particle swarm optimization

1 Introduction

The intelligent Internet of Things (IoT) is bolstering billions of smart devices connected to heterogeneous networks. This expansive network supports a wide range of computation-intensive applications, ranging from augmented reality (AR) to industrial robots and intelligent robotic cars [1, 2]. As these applications become increasingly complex and demand real-time processing, the limitations of resource-constrained IoT devices become more pronounced. Typically, these devices rely on high-speed real-time signal processing and high-rate data exchange [3]. However, due to their limited resources, they are unable to support computation-intensive tasks independently and need assistance from other devices with richer resources. Therefore, there is an urgent need for effective solutions to enhance computational capabilities without overburdening the IoT devices themselves.

Traditional cloud-centric architectures have been one of the primary solutions for providing computing and storage support. IoT devices collect data and transmit it to a central cloud server, which performs the most important computing tasks and returns the results to the IoT devices for display to the user. However, as IoT devices evolve into smart mobile devices (SMDs) and applications incorporate artificial intelligence, this traditional architecture is no longer suitable for latency-sensitive applications. The main constraints are the transmission latency of inter-cloud communication and the large overhead of I/O operations. Therefore, a new low-latency architecture is needed to meet the requirements of intelligent IoT.

As an emerging technology, edge computing is envisioned as a promising architecture for offloading tasks from IoT devices [4]. Edge computing brings computation and data storage closer to where it is needed, improving response times and saving bandwidth. Recent studies have proposed various task offloading strategies in

* Corresponding Author

edge computing environments. For instance, Yang et al. [5] formulate a long-term multi-objective problem that jointly optimizes the task execution latency and the sensing performance of multiple vehicles. Zhao [6] presented a green task offloading framework integrating energy harvesting technology. By applying Lyapunov optimization theory, he transformed the original system power consumption problem into a weighted sum problem of system power consumption and task queue stability. Although Yang and Zhao's research have all proposed various task offloading strategies in edge computing environments, they primarily focus on optimizing latency and energy consumption, while overlooking the potential resource waste and bottlenecks caused by uneven load distribution among edge servers. Ma et al. [7] proposed a distributed approach that directly confronts the multi-variable non-convex challenge by decoupling variables and deriving constraint-based bounds to guide offloading and allocation decisions. Ma's approach reduces the time complexity of the algorithm and improves task computation latency in vehicular networks while balancing load distribution, but it overlooks energy consumption. In contrast to these methods, our proposed solution integrates load balancing directly into the offloading decision-making process. By incorporating storage utilization and computational resources into the offloading strategy, we aim to distribute tasks more evenly across edge servers from the outset. This proactive approach not only optimizes latency and energy consumption but also improves system performance by preventing server overloads and ensuring efficient resource utilization.

In this work, we focus on the challenge of load balancing in edge computing environments to enhance the overall system performance. We develop a cloud-edge collaborative framework that optimizes task offloading decisions using an enhanced Particle Swarm Optimization algorithm. The main contribution of this paper includes three aspects:

1. A network model for edge computing is proposed. By decomposing the edge computing process into various modules and modeling them sequentially, the mathematical model seeks to align with the practical edge computing networks.
2. In order to achieve load balancing between edge servers, we introduce storage utilization as an important factor for offloading decisions.
3. An enhanced particle swarm optimization algorithm is proposed, which can be used to solve the optimization problems with discrete solution spaces. Additionally, a method is proposed to automatically change algorithm parameters based on the number of iterations, boosting the convergence speed of the algorithm, and minimizing the level of human interaction.

The remainder of this paper is organized as follows: In Section 2, we review related work on task offloading and load balancing in edge computing. Section 3 presents the proposed network model and problem formulation. In Section 4, we describe the enhanced Particle Swarm Optimization algorithm in detail. Section 5 discusses simulation results and performance analysis. Finally, Section 6 concludes the paper and discusses future work.

2 Related Work

With the in-depth study of edge computing in recent years, different layered architectures have emerged. Based on previous studies, these designs may be roughly categorized into three types [8]: "user-edge" architecture, "user-edge-cloud" architecture, and "edge-cloud" architecture. Among these, the "edge-cloud" architecture, also known as cloud-edge collaboration, can be defined as the offloading of computational tasks between the central cloud server and edge servers. Extensive research has been conducted in the past few years on computation offloading in the edge computing environment. These approaches aim to meet the diverse needs of various services.

Offloading is considered one of the solutions to support mobile applications' capabilities by moving computations from the mobile application to resource-rich servers. The design objective is to make efficient offloading decisions to improve network performance. Existing task offloading schemes can be categorized into three groups based on their characteristics [8]. The task allocation diagram is presented in Fig. 1.

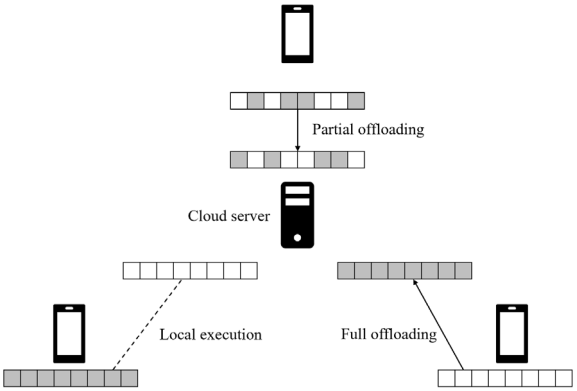


Fig. 1. Schematic diagram of task offloading type

1. Full offloading: The whole computation is offloaded and processed by the MEC server.
2. Partial offloading: A part of the computation is processed locally while the rest is offloaded to the MEC server.
3. Local execution: The whole computation is done locally at the UE. The offloading to the MEC is not performed, for example, due to unavailability of the MEC computation resource.

The existing research in this field can be broadly categorized into three primary optimization objectives [9]:

1. Low Latency: The primary aim is to minimize the overall completion time of computing tasks. When a computing task is executed locally, the latency is the execution time of the task. However, when offloading tasks to edge or central cloud servers, the latency includes: i) upload latency, ii) computation time, iii) download latency, and iv) communication latency. To minimize the communication latency and balance the workload, Guo et al. [10] propose an approach that combines K-means clustering and mixed-integer quadratic programming for the placement problem of the edge and the cloud. Fan et al. [11] seek to optimize the latency by modeling the latency minimization problem as a Markov decision process (MDP), and propose a Q-learning algorithm to solve this optimization problem.
2. Energy Efficiency: The total energy consumption consists of the energy consumption of computation and data transmission. In fog networks, Hazra et al. [12] present an Energy-Efficient Task Offloading (EETO) mechanism that uses Lyapunov optimization techniques to address the energy-performance trade-off in handling real-time IoT applications, thus reducing queueing time and overall energy consumption. Optimizing only latency or only energy consumption may potentially result in suboptimal performance. Therefore, many studies try to optimize the energy consumption and latency at the same time. Chen et al. [13] propose a robust computation offloading strategy with failure recovery (RoFFR) in an intermittently connected cloudlet system aiming to reduce energy consumption and the task completion time. In paper [14], Li et al. focus on maximizing the energy efficiency of the system through joint optimization of computation time allocation, energy consumption, local computing capacity and task offloading by proposing a tabu search based algorithm.
3. Load balancing: Load balancing for edge servers is also a critical concern. Various load balancing techniques are presented [15-17], with a specific focus on situations where the number of servers is limited. Hui et al. [15] address load balancing through a workload migration scheme. This scheme involves making offloading decisions based on the current workload of the server and the available resources of the target server. It facilitates the transfer of workloads from overloaded servers to lightly loaded servers, thereby improving the load balance and enhancing network resource utilization. Zhang et al. [16] propose a decentralized learning scheme for load balancing in mobile edge computing systems. This scheme leverages a non-cooperative random congestion game with limited information, allowing users to independently select their offloading strategies to enhance their effectiveness and system performance. Gao et al. [17] introduce a load balancing-aware task offloading approach in the mobile edge computing environment. It employs a Markov Decision Process and deep reinforcement learning to generate offloading strategies, and then uses a Particle Swarm Optimization algorithm to optimally balance the load across edge servers. However, these studies primarily solve the load balancing problem after offloading decisions have been made. That is to say, only when one of the servers is highly loaded, the load balancing algorithms will be

invoked. This reactive approach may not effectively prevent overload situations and can lead to increased latency and decreased performance.

Load imbalance among edge servers can lead to several issues, such as increased latency, reduced system throughput, and inefficient resource utilization. Overloaded servers may become bottlenecks, causing delays and negatively impacting user experience, while underutilized servers represent wasted computational resources. Despite the advancements in task offloading and load balancing techniques, integrating load balancing directly into the offloading decision process remains a significant challenge. Most existing research does not adequately address load balancing during the offloading decision-making process, treating it as a separate or subsequent problem. The key research problem addressed in this paper is how to simultaneously make offloading decisions and achieve load balancing among edge servers in a proactive manner, ensuring optimal resource utilization and improved system performance.

Our proposed solution involves developing an integrated optimization scheme that considers load balancing as a crucial factor during the offloading decision-making process, rather than as a subsequent step. By incorporating storage utilization and computational resources into the offloading strategy, we aim to distribute tasks more evenly across edge servers from the outset. This approach not only improves system performance and reduces latency but also maximizes resource utilization and enhances the overall quality of service for users, highlighting the significance of solving the load imbalance problem in edge computing environments.

3 Joint Optimization Modeling of Task Offload and Load Balancing Based on Cloud Servers' Collaboration

3.1 System Model

Fig. 2 shows the structure of our model, it includes three parts: the cloud data center layer, the edge server layer, and the mobile device layer. The top layer is the central cloud layer, it contains a high-performance physical server. The intermediate layer is the edge layer, which consists of numerous edge servers represented by the set $S = \{s_1, s_2, \ldots, s_m\}$. The bottom layer is the mobile device layer, also known as the user layer, consisting of various types of IoT mobile terminals, sensors or smart devices represented by the set $U = \{u_1, u_2, \ldots, u_n\}$.

In this model, tasks are offloaded using a 0/1 scheme, meaning that each task can only have two options: either fully offloaded or it should remain on the local device. However, in the case of full offloading, the task may be offloaded to an edge server or to offload to a central cloud server.

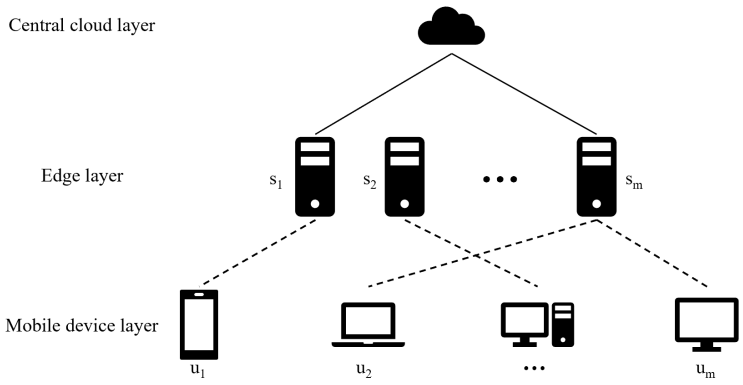


Fig. 2. Cloud-edge collaborative task offloading system model

3.2 Task Model

In this model, it is assumed that each user device u_i has a set of tasks $T_i^{local} = \{T_1, T_2, ..., T_{ki}\}$, which represent the tasks that have not yet been computed on the mobile device u_i, where k_i indicates the number of remaining tasks for device u_i. For any task $T_j (1 \leq j \leq k_i)$, the key properties of the task can be expressed by a triplet:

$$T_j = \left\{ m_j, c_j, d_j^{\max} \right\}, \tag{1}$$

where m_j represents the storage resources required by the task, the first task in each device's task list is the first task for offloading. c_j denotes the number of central processing unit (CPU) cycles necessary to execute this computational activity. d_j^{max} indicates the maximum latency of the task, also known as the deadline for the task.

As most tasks are generally executed sequentially, that means that the first task in each device's task list is the first task for offloading. Let's denote this task as $T_{i,1}^{local}$. We design a binary decision variable $x_{i,j}$ to denote whether the first sequential task $T_{i,1}^{local}$ of user device u_i should be offloaded to edge server s_j. If $x_{i,j} = 1$, it implies that task will be offloaded to the servers, whereas $x_{i,j} = 0$ represents that task will not be offloaded.

For any mobile device u_i, we introduce two binary variables $x_{i,0}$ and $x_{i,m+1}$. $x_{i,0}$ denotes whether the task $T_{i,1}^{local}$ should be directly computed on the local device, and $x_{i,m+1}$ denotes whether the task should be offloaded to the central cloud server. Therefore, the constraints can be expressed as follows:

$$\sum_{j=0}^{m+1} x_{i,j} = 1, \tag{2}$$

3.3 Communication Model

In our edge computing model, it is assumed that fiber optical communication networks are used for communication between the edge servers and the central cloud server, while wireless communication employed orthogonal frequency division multiplexing (OFDM) for data transmission between mobile end devices and the edge servers [18]. However, the use of OFDM introduces some level of noise and interference, which can impair the receipt of signals, leading to a reduction in the quality of communication. This degradation can be represented by the signal-to-interference-plus-noise ratio (SINR). Let $\boldsymbol{h}$ represent the signal gain matrix from the users to the server inside the coverage region, and $\boldsymbol{p}_i$ represent the output power matrix of the users. Therefore, the data transmission rate of the wireless channel can be calculated as:

$$s_{i,j}^k = \frac{\left|\boldsymbol{h}\right|_{i,j}^k \boldsymbol{p}_i}{\sum_{m \in U \setminus u_i} \boldsymbol{p}_m \left|\boldsymbol{h}\right|_{m,j}^k + \sigma^2}, \tag{3}$$

According to Shannon formula, the transmission rate of information can be obtained as follows:

$$r_{i,j} = \omega \log_2 \left(1 + \sum\nolimits_k s_{i,j}^k\right), \tag{4}$$

where ω represents the channel bandwidth, σ^2 is the noise power, and k denotes the number of subchannels in OFDM.

3.4 Computation Model

The computation offloading model includes modeling and simulating the latency for various offloading scenarios and making the offloading decisions.

In this model, there are three choices for the local device when processing tasks: tasks are executed locally on

the local device, offloading tasks to edge servers, or offloading tasks to central cloud servers for processing. The workflow of task offloading is showed in Fig. 3.

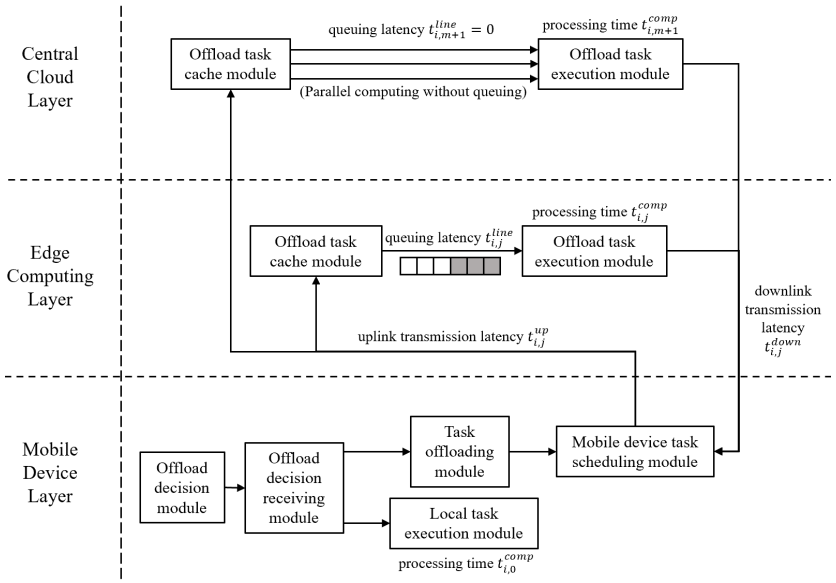


Fig. 3. Flowchart of task offloading

1) Computational tasks are processed locally

When the user device u_i chooses to process the first sequential task $T_{i,1}$ on the local device, the decision variable $x_{i,0}$ is set to 1. Since the computation task is processed locally, there is no data communication latency. Thus, the overall latency is equal to task's execution time:

$$t_{i,0}^{local}=\frac{c_i}{f_i^{local}}. \tag{5}$$

Where c_i represents the size of the data that needs to be computed in one task and f_i^{local} means the computing power of the CPU of the user equipment u_i, that is, the frequency of the CPU.

2) Offloading tasks to edge servers

When the user device u_i decides to offload the first-order task $T_{i,1}$ to the edge server s_j $(1 \le j \le m)$ for processing, the decision variable $x_{i,j}$ is set to 1. The offloading is done through four steps, as indicated in Fig. 3: i) The user device u_i transfers the code and input data of task $T_{i,1}$ to the edge server s_j through wireless network. ii) Task $T_{i,1}$ waits in the queue of edge server s_j until all previous tasks are completed. iii) Edge server s_j executes the task. iv) The server returns the computation results to the user device.

When the task is offloaded, the uplink data transmission latency can be expressed as:

$$t_{i,j}^{up}=\frac{m_i}{r_{i,j}}, \tag{6}$$

The queuing delay together with the processing time can be expressed as:

$$t_{i,j}^{comp}=\frac{\sum_{k=1}^{n}x_{k,j}c_k}{f_j^{edge}}, \tag{7}$$

where f_j^{edge} represents the computing power of the CPU of the edge server s_j, where $\Sigma_{k=1}^{n} x_{k,j} c_k$ means the total number of processing cycles of all computing tasks offloaded to the edge server.

For the downlink transmission latency of the fourth portion of the process, this element of the latency is often ignored because the result data size of most jobs is very small. When the computation tasks are offloaded to the edge server, the total latency is as follows:

$$t_{i,j}^{edge} = t_{i,j}^{up} + t_{i,j}^{comp}. \tag{8}$$

3) Offloading tasks to central cloud server

When the user device u_i decides to offload the first task $T_{i,1}$ to the centralized cloud server for processing, the decision variable $x_{i,m+1}$ is set to 1. However, the communication between the user device and the centralized cloud server comprises not only wireless network transmission but also data transmission through optical fiber network. Therefore, the uplink data transmission latency is given as:

$$t_{i,m+1}^{up} = \frac{m_i}{r_{i,j}} + n\frac{m_i}{c} + \tau, \tag{9}$$

where n represents the optical amplification times, c represents the optical fiber transmission rate, and τ is the uplink transmission latency of the long-distance optical backbone network [16].

The processing time of the task in the central cloud server is expressed as follows:

$$t_{i,m+1}^{comp} = \frac{c_i}{f^{center}}, \tag{10}$$

where f^{center} represents the computing power of the central cloud server. Consequently, the total latency is expressed as:

$$t_{i,m+1}^{center} = t_{i,m+1}^{up} + t_{i,m+1}^{comp}. \tag{11}$$

3.5 Energy Model

In edge computing, most mobile devices have limited battery capacity. Therefore, in order to extend the lifetime of mobile devices, it is vital to improve the energy efficiency.

When the user device u_i decides to process the computation task locally, γ_i^{local} indicates the power consumption required by the user device's CPU per cycle. Then the energy consumption of the local computation task can be expressed as:

$$e_i^{local} = c_i \cdot \gamma_i^{local}. \tag{12}$$

When the user device u_i decides to offload the computational task to a cloud server, the energy consumption during task offloading can be expressed as:

$$e_{i,j}^{up} = p_i \cdot t_{i,j}^{up}. \tag{13}$$

where the p_i denotes the transmission power of the user devise u_i .

3.6 Load Balancing

In edge computing, many factors, such as CPU utilization, storage space utilization, and network bandwidth will affect task offloading and resource allocation decisions. To prevent the task offloading algorithm from offloading most of computation tasks to some or even one of the servers, load balancing among edge servers is needed. Among the critical factors affecting load balance, available storage space stands out prominently. Because in general, more complex computation tasks require more storage space. Therefore, the storage space utilization R_i^{use} of the edge cloud server is used as the metric defining the network load.

$$R_j^{use} = \frac{\sum_{i=1}^{n} x_{i,j} \cdot m_i}{R_j^{storage}}. \tag{14}$$

where $R_j^{storage}$ represents the available storage space of edge cloud server s_j. Obviously, the sum of the storage space occupied by all tasks allocated to cloud server s_j should not exceed the server's storage space. Therefore, there is the following constraint:

$$\sum_{i=1}^{n} x_{i,j} \cdot m_i \le R_j^{storage}. \tag{15}$$

Formula (14) reflects the storage space utilization ratio of each cloud server. To describe the load balancing degree of all cloud servers in the edge computing network, we further introduce the balance index μ, which is represented as (16):

$$\mu = \frac{\sum_{i \in S} \left(R_i^{use} - \overline{R} \right)^2}{m}. \tag{16}$$

where m is the total number of cloud servers in the network, and $\overline{R}$ is the average storage space utilization of all cloud servers, defined as:

$$\overline{R} = \frac{\sum_{i \in S} R_j^{use}}{m}. \tag{17}$$

Obviously, the closer the balance index μ is to zero, the more balanced the load is.

3.7 Optimization Problem Formulation

Based on the foregoing research, the mathematical model and constraint conditions for the combined optimization of task offloading and load balancing with cloud-edge collaboration have been established. In this section, we formulate our problem as a multi-objective optimization problem.

The execution latency of each task is limited by the variable d^{max}. Therefore, together with the binary decision variable $x_{i,i}$, the latency constraint of each task may be represented as:

$$x_{i,0} \cdot t_{i,0}^{local} + x_{i,1} \cdot t_{i,1}^{edge} + \cdots + x_{i,m} \cdot t_{i,m}^{edge} + x_{i,m+1} \cdot t_{i,m+1}^{center} \le d_i^{max}. \tag{18}$$

According to the above analysis, our research objective is to minimize the total latency and the energy consumption of the user's device, and achieve the load balancing of the cloud servers. Therefore, in the scenario of multi-users, multiple edge servers, and a single central cloud server, the optimization problem is formulated as (19):

$$
\begin{aligned}
&\min \sum_{i=1}^{N} X_i \cdot t_i^{delay} \\
&\min \sum_{i=1}^{N} E_i^{local} \\
&\min \mu \\
s.t.\ \ & C1: X_i \cdot t_i^{delay} \le d_i^{\max}, \forall i \in U \\
& C2: \sum_{i=1}^{n} x_{i,j} \cdot m_i \le R_j^{storage}, \forall j \in S \\
& C3: X \cdot \left(1 \quad \cdots \quad 1\right)_{1\times i}^{T} = \left(1 \quad \cdots \quad 1\right)_{1\times i}^{T} \\
& C4: x_{i,j} \in \{0,1\}, \forall i \in U, j \in S \\
& C5: m_i > 0, c_i > 0, d_i^{\max} > 0, \forall i \in U
\end{aligned}
\tag{19}
$$

where C1 represents that the task offloading latency for users should not exceed the specified time limit, which is the vectorized representation of formula (18); C2 represents that the storage requirements of all tasks offloaded to the cloud server should not exceed its designated storage capacity; C3 represents that each computational task can be scheduled to only one device for processing; C4 denotes the offloading decision constraint of the user device; and C5 ensures that all task resources and other parameters should be non-negative values.

4 Solution

From formula (19), we observe that the optimization problem involves both task offloading and load balancing. Due to the inclusion of both integer constraints and continuous variables, this optimization problem falls into the category of mixed-integer nonlinear programming. Furthermore, since the optimization problem in this study has three different optimization targets, making it is a multi-objective optimization problem. To address this problem, we propose an improved multi-dimensional discrete particle swarm optimization (MDPSO) algorithm to solve this problem.

4.1 Analytic Hierarchy Process and Multi-objective Optimization

The linear weighting method is one of the multi-objective optimization methodologies based on aggregation or decomposition. It converts a multi-objective problem into a single-objective problem by assigning different weights to each objective. In the multi-objective optimization problem (19), we define three optimization objectives: latency ($target_{time}$), energy consumption ($target_{energy}$), and load balancing ($target_{load}$). The multi-objective problem can be converted into a single-objective problem as follows:

$$
\min Z = \alpha_1 \cdot target_{time} + \alpha_2 \cdot target_{energy} + \alpha_3 \cdot target_{load}. \tag{20}
$$

In the formula, α_1, α_2 and α_3 represent the weights assigned to the latency optimization $target_{time}$, energy consumption optimization $target_{energy}$ and load balancing optimization $target_{load}$, respectively. These weights satisfy the relation $\alpha_1 + \alpha_2 + \alpha_3 = 1$.

This study applies the Analytic Hierarchy Process (AHP) to determine the weights of the three indicators. AHP is multiple criteria decision-making approach. It systematically structures complex decision problems into hierarchical frameworks and determines the weights of each factor by generating a judgment matrix. This process ultimately leads to optimal solutions.

First, we construct the judgment matrix, also known as a positive reciprocal matrix. The matrix entries measure the relative importance of the issues and are selected from the Saaty set [20] (Table 1). These entries are arranged such that $a_{ij} = 1/a_{ji}$ and $a_{ii} = 1$.

Table 1. Saaty's relative indices

Relative importance of issues i and j	Value of a_{ij}
i and j of equal importance	1
i slightly more important than j	3
i more important than j	5
i a lot more important than j	7
i overwhelmingly more important than j	9

In this scenario, load balancing primarily impacts servers' performance and has a slight effect on users' experience. Therefore, when comparing latency or energy consumption with load balancing, we assign a relative importance value of 3. On the other hand, latency is the most directly perceived metric by users and significantly influences QoS. Therefore, when comparing latency to energy consumption, we assign a relative importance value between 1 and 3. We utilized the extended Saaty set for refinement, and consequently, latency is assigned a rating of 2 in comparison to energy consumption. The positive reciprocal matrix is represented in formula (21). The first, second, and third rows correspond to latency, energy usage, and load balancing, respectively.

$$\boldsymbol{A} = \begin{pmatrix} 1 & 2 & 3 \\ \frac{1}{2} & 1 & 3 \\ \frac{1}{3} & \frac{1}{3} & 1 \end{pmatrix}, \tag{21}$$

After generating the judgment matrix, a consistency check is necessary. The consistency index (CI) formula is as follows:

$$CI = \frac{\lambda_{\max} - n}{n-1} = \frac{3.0536 - 3}{3-1} = 0.0268, \tag{22}$$

where λ_{max} represents the principal eigenvalue and n is the order of the matrix. According to the average random consistency index (RI), the matching value is $RI = 0.52$. Next, we can calculate the consistency ratio (CR) as follows:

$$CR = \frac{CI}{RI} = \frac{0.0268}{0.52} = 0.0516. \tag{23}$$

Therefore, $CR = 0.0516 < 0.1$, it can be concluded that the consistency of the judgment matrix is acceptable.

Finally, the eigenvalue approach is used to calculate the weight. The eigenvector corresponding to the greatest eigenvalue of the judgment matrix is [0.8257, 0.5201, 0.2184], which, after normalization, yields [0.5278, 0.3325, 0.1396]. In this paper, the weights of the three are selected as $\alpha_1 = 0.5278$, $\alpha_2 = 0.3325$, $\alpha_3 = 0.1396$.

In this paper, the 0-1 transformation is applied to normalize the data. Since all three objective functions are cost-type indicators, suggesting that smaller values are preferable, the 0-1 transformation is as follows:

$$\widehat{target_i} = \frac{target_i^{max} - target_i}{target_i^{max} - target_i^{min}}. \tag{24}$$

where $target_i^{\max}$ and $target_i^{\min}$ are the maximum and minimum value of $target_i$, and $\widehat{target_i}$ is the normalized value following the 0-1 transformation. After the normalization, the optimal value of $target_i$ will change to 1, and the worst value of $target_i$ will change to 0.

Therefore, the single-objective optimization problem with linearly weighted optimization objectives is as follows:

$$\begin{aligned} &\max \sum_{i=1}^{3} \alpha_i \cdot \widehat{target_i}, \\ &s.t.\ C1-C5. \end{aligned} \tag{25}$$

4.2 Solution Based on Improved Particle Swarm Optimization

Particle swarm optimization (PSO) is widely employed in solving single optimization problems. Inspired by the foraging behavior of birds, the PSO method simulates how a flock gradually converges towards an optimal goal through individual and group experiences, thereby obtaining desirable outcomes. The conventional PSO approach employs the following formula to update the position and velocity of particles:

$$v_i^d = \omega v_i^d + c_1 r_1 \left(p_i^d - x_i^d \right) + c_2 r_2 \left(g^d - x_i^d \right), \tag{26}$$

$$x_i^d = x_i^d + \alpha v_i^d . \tag{27}$$

Due to the discrete solution space inherent in the optimization problem addressed in this paper, the traditional PSO approach cannot be directly applied. Therefore, an improved binary discrete particle swarm approach is adopted, where the position of a particle can only take values of 0 or 1, while the velocity v_{ii} remains continuous.

To achieve this, the sigmoid function is employed to map the values of each dimension of the velocity v_{ij} to a continuous space ranging from 0 to 1, denoted as $s(v_{ij})$. These mapped values represent the chance of the relevant position v_{ij} taking the value of 1. Finally, the position update algorithm is appropriately adjusted to ensure the mapping from continuous space to discrete space. The velocity compression formula and particle position update formula are expressed as follows:

$$s(v_{ij}) = \frac{1}{1+e^{-v_{ij}}}, \tag{28}$$

$$x_{ij} = \begin{cases} 1, & r < s(v_{ij}) \\ 0, & otherwise \end{cases}. \tag{29}$$

where r is a random number uniformly distributed between 0 and 1.

The discrete particle swarm approach handles the problem of location updates when particles have only two possible values in the state space. However, for the optimization problem (19), apart from satisfying the binary variable constraints, there exists an additional constraint C4 among the variables. Therefore, a more refined particle position update formula is used as shown in equation (30):

$$x_{ij} = \begin{cases} 0, s(v_{ij}) \le r_0 \\ 1, r_0 < s(v_{ij}) \le r_1 \\ \quad \vdots \\ m+1, r_m < s(v_{ij}) \end{cases}. \tag{30}$$

In equation (30), r_1, r_2, …, r_m represent random values in the range [0, 1] that are monotonically increasing relationship. By utilizing the position update formula (30), the solution space is reduced from $n \times (m + 2)$ dimensions to n dimensions, while preserving the real range of the solution space.

In the conventional particle swarm optimization algorithms, the selection of the learning factors (c_1 and c_2) is often chosen manually, which increases the uncertainty in the algorithm's solution finding process [18]. To address this issue, this study introduces an intelligent technique for selecting the learning factors, allowing them to be automatically adjusted during the iteration process of the algorithm.

The learning factor c_1 determines the influence of one's own experience on the next step of motion, while c_2 determines the influence of group social experience on motion. Therefore, throughout the full iterative process, c_1 gradually decreases from a higher value, while c_2 gradually increases from a smaller value. This method enhances the algorithm boost the global search ability of particles in the early stage of the algorithm, while achieving a more refined local search in the later stage.

In order to increase convergence speed and obtain the ideal value, the expression of the improved learning factor is:

$$c_1 = 2\sin^2\left[\frac{\pi}{2}\left(1-\frac{t}{T_{\max}}\right)\right], \tag{31}$$

$$c_2 = 2\sin^2\left[\frac{\pi t}{2T_{\max}}\right]. \tag{32}$$

where t indicates the current iteration number of the algorithm, and T_{max} represents the maximum number of iterations. The above two formulas are monotonic decreasing and increasing functions respectively, consistent with the changing trend outlined in the previous analysis.

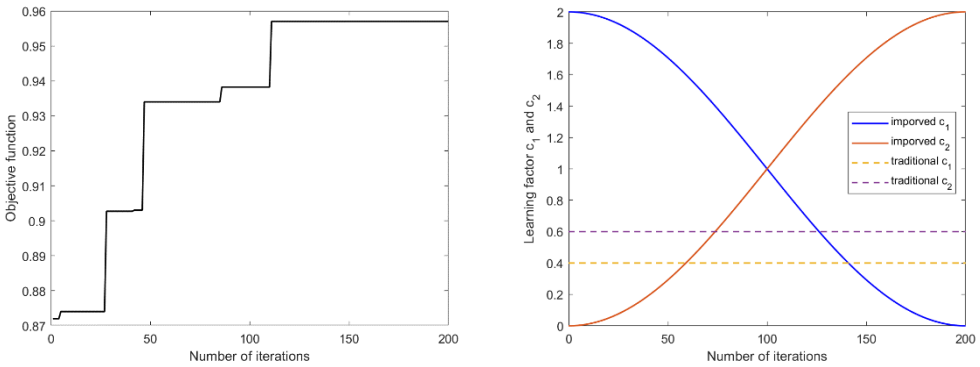
In this paper, the improved particle swarm optimization algorithm is employed to solve the optimization problem. Therefore, the pseudo code of the improved particle swarm optimization algorithm as follow:

```
  program improved_multi_dimensional_discrete_pso:
  { Initialize the particle position of the particle swarm };
  { Calculate the optimal value Pbest_i and global optimal value Gbest of
each particle in the initial case }
  begin
    while (number of iterations or stop conditions have not been reached)
       for (each particle)
          { Update the speed of particles according to the speed update
formula (26) };
          { Determine the boundary according to the updated particle ve-
locity };
          { Update the position of particles according to formulas (28)
and (30) };
          if (solution given by particles is feasible)
             { Calculating objective function Formula (25) };
             { Update the optimal value Pbest_i & global optimal value
Gbest of particle };
          else do
             { Calculating penalty function };
          end.
       end.
    end.
end.
```

5 Simulation and Analysis

5.1 Experimental Simulation Results

By setting specific parameters in the modified particle swarm optimization algorithm, the objective function evolution diagram is illustrated in Fig. 4.



(a) Objective function evolution curve (b) Learning factor curve

Fig. 4. MDPSO algorithm simulation experiment results

Fig. 4(a) shows that the modified particle swarm algorithm continuously explores multiple solutions while keeping track of the best answer. During the iterative process, the algorithm initially converged to a value of 0.872. As iterations progressed, it finally reached the peak value of 0.9584 at the 111th iteration. This demonstrates that the algorithm effectively balances exploration and exploitation, allowing it to escape local optima and find a better global solution over time. Fig. 4(b) shows the changing trend of the improved learning factors c_1 and c_2. Unlike the traditional method, where these factors remain constant, they gradually change as the number of iterations increases. Specifically, the cognitive learning factor c_1 decreases while the social learning factor c_2 increases, which enhances the algorithm's ability to converge by relying more on the collective experience of the swarm in later iterations.

To evaluate the performance of our proposed MDPSO algorithm, we compare it with three different schemes with three key metrics: latency, energy consumption and load balancing. The three baseline methods are as follows:

1. ACCO [19]: Through an approximation collaboration algorithm, ACCO determines the device's offloading decision by a greedy strategy in three stages, aiming to minimize the energy consumption.
2. JORDDPG [6]: By applying Lyapunov optimization theory, JORDDPG transforms the original system power consumption problem into a weighted sum problem of system power consumption and task queue stability, effectively balancing resource management and energy efficiency optimization for edge computing.
3. RoFFR [13]: Chen et al. propose a robust computation offloading strategy with failure recovery (RoFFR) in an intermittently connected cloudlet system, aiming to reduce energy consumption and task completion time.

Table 3 provides the specific values of the three objectives of the enhanced particle swarm system and three different baselines.

Table 2. Comparison of different schemes

Scheme	Latency (s)	Energy consumption (J)	Load balancing
ACCO	6.6467	131.4545	0.61
JORDDPG	9.5732	47.0638	0.58
RoFFR	6.2493	68.5062	0.33
MDPSO	7.1825	58.3908	0.23

From the findings in Table 2, it is evident that our proposed optimization algorithm MDPSO has the best performance according to these three metrics. Compared with the other three baseline methods, MDPSO algorithm achieved the best load balancing performance, with an improvement of about 30% compared to RoFFR algorithm. In addition, although the MDPSO algorithm did not achieve the best latency and energy consumption indicators, the difference compared to the RoFFR algorithm with the best latency is 12%, and the difference compared to the JORDDPG algorithm with the best energy consumption is 19%. This trade-off is due to our al-

gorithm incorporating load balancing metrics and increasing priority in the multi-objective optimization process. Therefore, these results demonstrate the effectiveness of MDPSO in achieving a balanced optimization across multiple objectives.

To further validate the performance of our algorithm, we conducted additional experiments comparing MDPSO with other existing methods in terms of scalability and robustness under varying network conditions.

5.2 Algorithm Comparison

This work establishes a simulation environment using MATLAB and compares the proposed method with four different schemes. From the comparative experiment in Table 3, the RoFFR algorithm has the best performance among the three baselines we selected. Therefore, we chose the RoFFR algorithm and two other basic scenarios for further comparison. The selected comparison algorithms are as follows:

Scheme 1: local computing scheme. All tasks are calculated locally by the user device.

Scheme 2: offload-all scheme. All tasks are offloaded to the centralized cloud server for computation.

Scheme 3: the RoFFR algorithm.

Scheme 4: the improved particle swarm optimization technique MDPSO presented in this paper. We conducted extensive simulations to assess the performance of these schemes under different network sizes and resource configurations.

Without loss of generality, Table 3 shows the basic characteristics of each device (including mobile user devices and cloud servers), as well as the parameter settings of the computing tasks of each user device. The simulation parameters are selected to reflect realistic settings in IoT environments, ensuring that the experimental results are representative of practical scenarios.

Table 3. Parameters settings

Parameter	Description	Value
m_i	Task data size	300-800 KB
c_i	Number of CPU cycles required for the task	100-1000 Megacycles
d^{max}	Maximum tolerated task latency	10-100 seconds
f_i^{local}	Local device CPU base frequency	740-860 MHz
γ_i^{local}	Local device CPU power consumption	0.3-1.3 J/Gigacycles
p_i	Local device uploads transmit power	280-520 mW
r_i	Local device data upload rate	9.5-10.5 MB/s
f_j^{edge}	Edge server CPU's frequency	3.8-4.2 GHz
$R_j^{storage}$	Edge server's storage space	500-1500 KB [19]
f^{center}	Central cloud server CPU's frequency	6 GHz

Fig. 5 depicts the relationship between the number of network users and the total time latency of all computational tasks. From Fig. 5, it can be noticed that, among the four ways, the total time latency of all computational tasks increases as the number of network users grows. This is expected, as more users generate more computational tasks, leading to increased processing times and potential network congestion. Additionally, the modified particle swarm technique described in this paper achieves the minimum latency when the number of network users is less than 35. However, if the number of users exceeds 35, the latency caused by the modified particle swarm algorithm is slightly higher than the strategy of keeping all tasks local. This is because as the number of users increase, the quantity of computational activities also increases. If these jobs are offloaded to servers, there will not only be a delay during task transmission, but they will also have to wait in the server's queue until previous tasks are all completed. In contrast, local computation avoids transmission delays but suffers from limited processing power. Therefore, keeping all tasks at the local place may be slightly faster than offloading them to servers. However, it's important to note that the MDPSO algorithm still provides a better balance between latency and energy consumption, as will be discussed next.

Fig. 6 displays the relationship between the energy consumption of user devices and the number of network tasks. It is worth noting that keeping all computing operations local results in the highest energy consumption. This is due to the limited computational efficiency of mobile devices, leading to higher energy usage per task. However, the energy consumption required by the modified particle swarm method is slightly higher than the technique of offloading all work to a central cloud server. This is because when all tasks are offloaded to the

server, local user devices only need to contribute energy during the task transmission phase. They continuously demand energy until the tasks are completed while processing activities locally. Offloading reduces computational energy but may increase communication energy consumption. Therefore, the modified particle swarm method sacrifices a portion of energy consumption to obtain lower latency, resulting in a superior total objective function value. Overall, the MDPSO algorithm achieves a good compromise between energy consumption and latency, making it suitable for applications where both factors are critical.

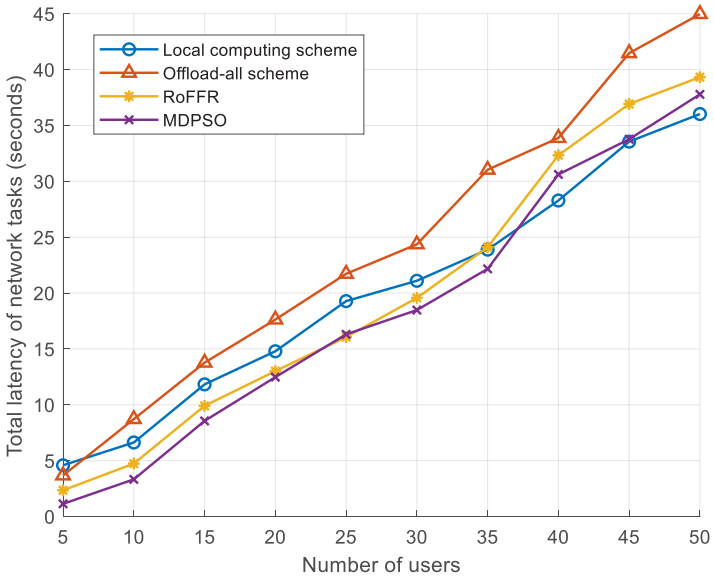


Fig. 5. Graph of the relationship between the total latency of network tasks and the number of users

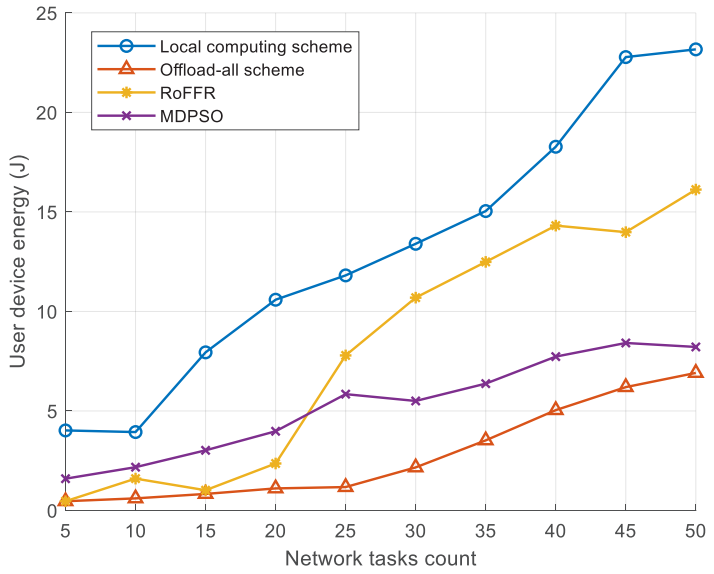


Fig. 6. Diagram of the relationship between user device energy consumption and task count

To analyze the impact of network size on energy consumption, we observe that as the number of tasks increases, the energy consumption for all schemes rises, but the rate of increase differs. The MDPSO algorithm demonstrates a moderate growth rate, indicating its scalability and efficiency in larger networks.

Fig. 7 depicts the relationship between the load of the network and the number of edge servers, with 3, 5, and 7 edge servers respectively. It can be noted that the load balancing metric of the edge servers shows a rising trend as the number of tasks in the network grows. This is because an imbalance in load distribution can occur when an

increasing number of tasks are scheduled to edge servers with superior performance, while the number of edge servers remains unchanged. Additionally, as the number of edge servers increases, the load measurements of edge servers tend to become more balanced. This is because with a growing number of edge servers, there are more opportunities to distribute computing tasks, leading to a tendency to schedule tasks to edge servers with lower loads. This helps to achieve overall load balancing among edge servers in the network. Our MDPSO algorithm effectively utilizes additional edge servers to distribute tasks more evenly, thereby improving load balancing and preventing potential bottlenecks in the network.

Similarly, Fig. 8 depicts the relationship between load balancing of the edge servers and the cache space. As the cache space rises, the load balancing exhibits a decreasing trend. This indicates that larger cache spaces allow for more tasks to be handled by individual edge servers, which can lead to imbalances if not properly managed. The MDPSO algorithm accounts for this by considering storage utilization in the offloading decisions, ensuring that tasks are allocated in a way that maintains load balance across the network.

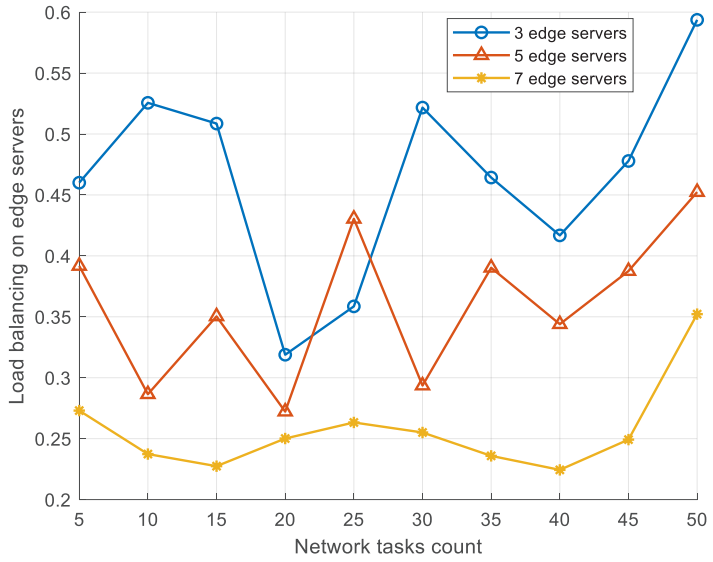


Fig. 7. Load balancing and task count relationship diagram for edge servers

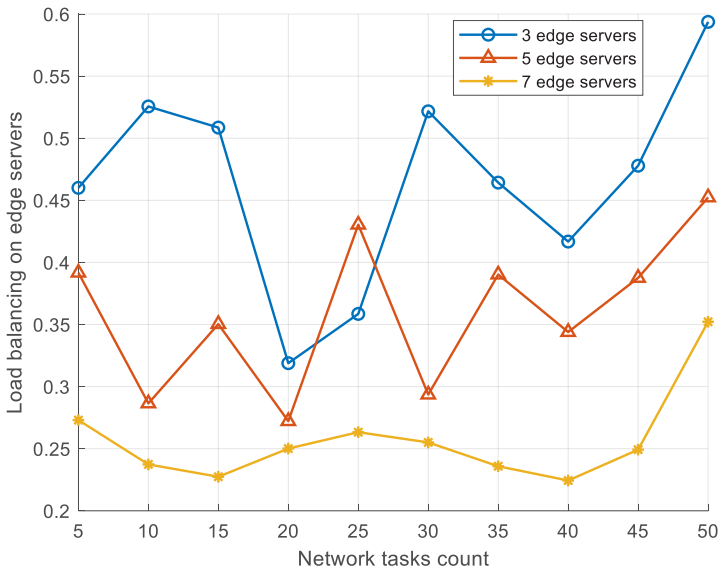


Fig. 8. Load balancing and cache space relationship diagram of edge servers

6 Conclusion

This paper studies the computing task offloading mechanism based on cloud-edge collaboration. Firstly, we present a mathematical model of an edge computing network. It is formulated as a multi-objective optimization problem with multi-constraint, which is then solved by an improved particle swarm optimization algorithm.

For edge computing networks, this paper offers a system model with multiple users, numerous edge servers, and a single centralized cloud server. Based on current research, we propose a new load balancing metric for edge servers that leverages server parameters. This metric is incorporated into an optimization problem as one of multiple objectives, aiming to achieve better offloading decisions alongside latency and energy consumption metrics. For the multi-objective optimization problem, we first transform it into a single-objective optimization problem by using standard 0-1 transformation and the linear weighting method. Then an enhanced particle swarm optimization approach is proposed to solve the problem. Traditional particle swarm optimization algorithms involve manual configuration of several parameters. This paper provides a method to automatically adjust the parameters based on the number of algorithm iterations, which results in faster convergence of the algorithm. To verify the proposed improved particle swarm optimization algorithm, simulation tests and analysis are conducted in Section 5. In the scenario of several users and a single edge server, the modified particle swarm optimization method suggested in this paper beats the RoFFR algorithm in terms of latency and energy usage. When the number of edge servers increases, the modified particle swarm optimization algorithm also displays good optimization outcomes in terms of latency and energy consumption compared to the other three approaches. At the same time, the edge servers maintain load balance.

However, there are some limitations to the proposed scheme. Firstly, the computational complexity of the algorithm increases with the number of users and tasks, which may limit its scalability in large-scale networks or real-time applications. Secondly, the model assumes stable network conditions and does not fully consider network dynamics such as varying bandwidth, latency fluctuations, or node failures, which can affect the performance of the offloading decisions. Future work will focus on optimizing the algorithm for better scalability and robustness and extending the model to account for more dynamic network scenarios. Additionally, real-world experiments will be conducted to validate the practical applicability of the proposed method.

7 Acknowledgement

This work was supported by Key R&D Plan of Jiangsu Province (No. BE2020084-5).

References

[1] W. Shi, H. Sun, J. Cao, Q. Zhang, W. Liu. Edge Computing—An Emerging Computing Model for the Internet of Everything Era, Journal of Computer Research and Development 54(5)(2017) 907-924. https://doi.org/10.7544/issn1000-1239.2017.20160941

[2] Y. Chen, B. Liu, W. Lin, H. Chen, Survey of Cloud-edge Collaboration, Computer Science 48(3)(2021) 259-268. https://doi.org/10.11896/jsjkx.201000109

[3] X. Liu, N. Ansari, Toward green IoT: Energy solutions and key challenges, IEEE Communications Magazine 57(3)(2019) 104-110. https://doi.org/10.1109/MCOM.2019.1800175

[4] K. Wang, W. Chen, J. Li, Y. Yang, L. Hanzo, Joint task offloading and caching for massive MIMO-aided multi-tier computing networks, IEEE Transactions on Communications 70(3)(2022) 1820-1833. https://doi.org/10.1109/TCOMM.2022.3142162

[5] H. Yang, Z. Feng, Z. Wei, Q. Zhang, X. Yuan, T.Q.S. Quek, Intelligent computation offloading for joint communication and sensing-based vehicular networks, IEEE Transactions on Wireless Communications 23(4)(2023) 3600-3616. https://doi.org/10.1109/TWC.2023.3309688

[6] J. Wang, H. Zhao, Collaborative offloading computing scheme based on energy harvesting technology, Chinese Journal on Internet of Things 8(2)(2024) 91-102. https://doi.org/10.11959/j.issn.2096-3750.2024.00361

[7] G. Ma, M. Hu, X. Wang, H. Li, Y. Bian, K. Zhu, D. Wu, Joint Partial Offloading and Resource Allocation for Vehicular Federated Learning Tasks, IEEE Transactions on Intelligent Transportation Systems 25(8)(2024) 8444-8459. https://doi.org/10.1109/TITS.2024.3393529

[8] S. Zhang, N. Yi, Y. Ma, A survey of computation offloading with task types, IEEE Transactions on Intelligent Transportation Systems 25(8)(2024) 8313-8333. https://doi.org/10.1109/TITS.2024.3410896

[9] R. Xie, X. Lian, Q. Jia, T. Huang, Y. Liu, Survey on computation offloading in mobile edge computing, Journal on communications 39(11)(2018) 138-155. https://doi.org/10.11959/j.issn.1000-436x.2018215
[10] Y. Guo, S. Wang, A. Zhou, J. Xu, J. Yuan, C.-H. Hsu, User allocation-aware edge cloud placement in mobile edge computing, Software: Practice and Experience 50(5)(2020) 489-502. https://doi.org/10.1002/spe.2685
[11] Y. Fan, S. Yuan, Y. Cai, R. Chen, Deep Reinforcement Learning-based Collaborative Computation Offloading Scheme in VehicularEdge Computing, Computer Science 48(5)(2021) 270-276. https://doi.org/10.11896/jsjkx.201000005
[12] A. Hazra, M. Adhikari, T. Amgoth, S.N. Srirama, Joint computation offloading and scheduling optimization of IoT applications in fog networks, IEEE Transactions on Network Science and Engineering 7(4)(2020) 3266-3278. https://doi.org/10.1109/TNSE.2020.3021792
[13] M. Chen, S. Guo, K. Liu, X. Liao, B. Xiao, Robust computation offloading and resource scheduling in cloudlet-based mobile cloud computing, IEEE Transactions on Mobile Computing 20(5)(2021) 2025-2040. https://doi.org/10.1109/TMC.2020.2973993
[14] C. Li, M. Song, L. Zhang, W. Chen, Y. Luo, Offloading Optimization and Time Allocation for Multiuser Wireless Energy Transfer Based Mobile Edge Computing System, Mobile Networks and Applications 27(5)(2022) 1783-1791. https://doi.org/10.1007/s11036-019-01396-3
[15] M. Hui, J. Chen, Y. Zhou, B. He, K. Wu, L. Yang, Server deployment and load balancing in stochastic mobile edge computing networks, IEEE Communications Letters 26(5)(2022) 1194-1198. https://doi.org/10.1109/LCOMM.2022.3151467
[16] F. Zhang, M.M. Wang, Stochastic congestion game for load balancing in mobile-edge computing, IEEE Internet of Things Journal 8(2)(2021) 778-790. https://doi.org/10.1109/JIOT.2020.3008009
[17] Y. Gao, Z. Li, Load balancing aware task offloading in mobile edge computing, in: Proc. 2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design, 2022. https://doi.org/10.1109/CSCWD54268.2022.9776263.
[18] D. Zhang, X. Li, J. Zhang, T. Zhang, C. Gong, New Method of Task Offloading in Mobile Edge Computing for Vehicles Based on Simulated Annealing Mechanism, Journal of Electronics & Information Technology 44(9)(2022) 3220-3230. https://doi.org/10.11999/JEIT210102
[19] H. Guo, J. Liu, Collaborative computation offloading for multiaccess edge computing over fiber–wireless networks, IEEE Transactions on Vehicular Technology 67(5)(2018) 4514-4526. https://doi.org/10.1109/TVT.2018.2790421
[20] H.A. Donegan, F.J. Dodd, T.B.M. McMaster, A new approach to AHP decision-making, Journal of the Royal Statistical Society 41(3)(1992) 295-302. https://doi.org/10.2307/2348551