Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

SRA-E-ABCO: terminal task offloading for cloud-edge-end environments

Shun Jiao[1], Haiyan Wang[1,2*] and Jian Luo[1]

Abstract

The rapid development of the Internet technology along with the emergence of intelligent applications has put forward higher requirements for task offloading. In Cloud-Edge-End (CEE) environments, offloading computing tasks of terminal devices to edge and cloud servers can effectively reduce system delay and alleviate network congestion. Designing a reliable task offloading strategy in CEE environments to meet users' requirements is a challenging issue. To design an effective offloading strategy, a Service Reliability Analysis and Elite-Artificial Bee Colony Offloading model (SRA-E-ABCO) is presented for cloud-edge-end environments. Specifically, a Service Reliability Analysis (SRA) method is proposed to assist in predicting the offloading necessity of terminal tasks and analyzing the attributes of terminal devices and edge nodes. An Elite Artificial Bee Colony Offloading (E-ABCO) method is also proposed, which optimizes the offloading strategy by combining elite populations with improved fitness formulas, position update formulas, and population initialization methods. Simulation results on real datasets validate the efficient performance of the proposed scheme that not only reduces task offloading delay but also optimize system overhead in comparison to baseline schemes.

Keywords Cloud-edge-end, Terminal device, Service reliability, Bee colony algorithm, Task offloading

Introduction

With the rapid advancement of Internet technology, there has been a significant increase in the number of User Equipments such as smartphones, tablets, and portable devices. According to a report published by Cisco [1], it is predicted that the number of User Equipments connected to the network has reached 13.1 billion by the end of 2023. Although the performance of User Equipments in computing power, battery life, and memory capacity has been improving continuously, it cannot meet the computational demands of latency-sensitive and computing intensive applications. As a result, there arises a necessity to offload computing tasks from User Equipments to the cloud center or edge servers located nearby to minimize system delay.

Cloud-edge-end (CEE) converged computing is a prominent computing paradigm resulting from the advancement of big data [2]. It encompasses various computing forms to create a unified architecture that incorporates end devices, edge nodes, and the cloud computing center. This architecture efficiently handles computationally demanding and time-sensitive tasks by offloading them from low-capacity and energy-constrained end devices to the edge of the network. To minimize latency and energy consumption on end devices, computation offloading strategies are employed in the cloud-edge-end scenario. Currently, researchers primarily focus on addressing the mismatch between application demands and end device capabilities in CEE environments. The core concept involves deploying edge servers in the neighborhood to terminal devices and offloading users' tasks to these servers, which possess computational and storage resources for processing. However, in the CEE scenario, the lack of

*Correspondence:
Haiyan Wang
wanghy@njupt.edu.cn
[1] School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210003, China
[2] Jiangsu Key Laboratory of Big Data Security and Intelligent Processing, Nanjing 210023, China

an effective task offloading strategy not only hinders the benefits of CEE collaboration but also compromises service reliability. Therefore, it is crucial to investigate task offloading methods that prioritize service reliability in CEE scenarios, with the key aspect being the formulation of effective offloading strategies.

However, there are still some problems in traditional task offloading methods. First, these methods lack reliable predictions for determining whether tasks need to be offloaded from the terminal device. Second, they fail to consider the correlation between edge nodes, resulting in low offloading reliability. Finally, existing methods do not thoroughly analyze the relationship between historical processing records of nodes and task types, leading to inefficient offloading strategies. For instance, traditional heuristic algorithms only consider node processing capacity and load, disregarding important indicators such as energy consumption and reliability. These problems will lead to inefficient offloading results.

To address these issues, this paper proposes an efficient and low-latency task offloading strategy. This strategy combines the analysis of offloading reliability and the optimization of bee colony algorithms to minimize energy consumption and reduce system delay. The main contributions of this paper are as follows:

1. A service reliability analysis (SRA) method for cloud edge-end environments is proposed, which first predicts whether an end task is offloaded or not by the attributes of the end device using the Logistic Regression model (LR) and then uses the historical processing data of each edge node, combined with the task attributes to construct a correlation matching matrix. By analyzing the matrix, a set of offloading decision vectors is obtained to provide relevant parameters for subsequent terminal task offloading decisions.
2. An offloading decision formulation method (Elite-Artificial Bee Colony Offloading, E-ABCO) is proposed, which continuously optimizes the offloading decision scheme by improving the fitness formula and the position update formula and improves the offloading efficiency and success of the terminal task offloading by means of cross-honey search.
3. A series of comparison experiments are con ducted on public datasets in *Kaggle*,[1] comparing the offloading schemes formulated by the proposed algorithm with those of local offloading, all offloading, greedy algorithm-based offloading, deep neural network model-based offloading, and directional bee colony algorithm offloading, and verifying the validity of the model proposed in this paper.

[1] Kaggle's official website: https://www.kaggle.com/datasets/sachin26240/vehicularfogcomputing

Related work

Task offloading

Gao et al. [3] proposed a cloud-edge cooperative architecture based on Lyapunov optimization theory, which reduces the problem to a constrained optimization problem by establishing a dynamic queueing model of cloud computing servers and edge computing servers and combining the system power function to form a drift plus penalty function framework. Li et al. [4] proposed a multi-objective strategy based on Biogeographic Optimization (BBO) algorithm and constructs a time-energy model and a cost model for task offloading, based on which the BBO algorithm is introduced into task offloading for MEC to solve the multi-objective optimization problem. Zhang et al. [5] proposed a multi-user, multi-computing task offloading scheme based on mobile cloud computing and mobile edge computing that minimizes energy consumption and latency through differential evolutionary algorithms and provides optimal computational task offloading decisions. Liu et al. [6] proposed a DVR-minimizing computational offloading scheme with task migration and merging with the goal of minimizing the system deadline rate (DVR), constructs an overall directed acyclic graph (DAG) for all current dependent tasks, develops a migration-enabled multi-prioritized task ranking algorithm that creatively introduces multiple task prioritization metrics and determines the optimal order in which tasks should be executed, and, finally, develops a learning algorithm based on the Deep Deterministic Policy Gradient (DDPG) for finding the optimal offloading strategy. Ai et al. [7] designed a smart collaborative framework scheme, establishes a theoretical model including a Hierarchical Spatial–Temporal Monitoring (HSTM) module and a fine-grained resource scheduling (FRS) module, applies a hybrid deep learning algorithm to the monitoring module from the Spatial–Temporal dimension, and in addition, adopts a hybrid game and improves the queuing theory to improve the offloading efficiency of the FRS module. Li et al. [8] utilized mobile edge computing and deep learning to decompose a convolutional neural network-based encoder-decoder model and deploy the encoder on the edge server to extract features of the task and derive user tolerance limits at the edge server by using a linear regression model in order to further improve the quality of user experience. Liu et al. [9] calculated the optimal task allocation ratio by a mathematical analysis method with the objective of minimizing the processing delay, and then uses the Lagrangian Dyadic (LD) method to obtain the optimal task offloading and resource allocation policy. Lv et al.

[10] studied the mobile edge computing task offloading problem under dependency and service caching constraints and proposed a table-based task offloading algorithm to optimize the maximum completion time and energy consumption by predicting the impact of offloading decisions. Ko et al. [11] proposed a belief-based task offloading algorithm (BTOA), in which the vehicle selects the target edge cloud and subchannels based on its beliefs and observes its current resource and channel conditions, and based on the observed information, the vehicle ultimately determines the most appropriate edge cloud and subchannels. Gao et al. [12] proposed a hierarchical computational partitioning strategy for DNNs, which divides the tasks of each MD into subtasks, Second, we develop a latency prediction model for DNNs to characterize the computational latency of the MDs and each subtask on the server. Third, we design a slot model and a dynamic pricing strategy for the server to efficiently schedule the offloaded subtasks. Fourth, we jointly optimize the design of task partitioning and offloading to minimize the cost of each MD in terms of computation latency, energy consumption, and price paid to the server. Yilin et al. [13] carried out in-depth research on the research progress of computational offloading in mobile edge computing, summarizes and generalizes two types of traditional task methods and intelligent methods based on online learning, and analyzes and compares the traditional computational offloading based on heuristic algorithms from the minimization of latency time, minimization of energy consumption, and trade-off between time and energy consumption with three different optimization objectives.

Li et al. [14] proposed a new multi-objective strategy based on the biogeography-based optimization (BBO) algorithm. In this strategy, a time-energy consumption model and a cost model are constructed for task offloading firstly. Based on these models, the BBO algorithm is introduced into task offloading for MEC to solve the problem of multi-objective optimization. Gao et al. [15] proposed a cloud-edge collaboration architecture, then by establishing the dynamic queue model of cloud computing server and edge computing server, and combining with the system power function to form a drift plus penalty function framework, the problem is reduced to a constrained optimization problem.

Service reliability

Tang et al. [16] proposed a heuristic greedy reliability and cost-aware job scheduling (RCJS) algorithm, which mainly addresses the problem of high service cost due to multiple replications caused by reliability enhancement techniques. Li et al. [17] investigated the problem of providing reliable VNF service provisioning in Mobile Edge Cloud (MEC) networks, proposes a new VNF service reliability problem and develops an effective online algorithm through primary and backup VNF instance placement and primitive pairwise updating techniques, and experimentally demonstrates the algorithm's promising nature. Li et al. [18] investigated the problem of providing reliable VNF services in mobile edge computing environments by providing primary and backup VNF instances to satisfy the reliability requirements of the users, and two efficient online scheduling algorithms are developed for the problem under two different backup scenarios, onsite (local) and offsite (remote), by using both primal and dual update techniques. Zhang et al. [19] proposed a new logistic function based Service Reliability Probability (SRP) estimation model which does not specify the distribution of resource requirements, investigates the Average SRP Maximization Problem (ASRPMP) in VM-based Edge Computing Servers (ECSs) by jointly optimizing the Service Quality Ratios (SQRs) and computational resource allocations, and proposes an Alternative Optimization Algorithm (AOA) transformed into a Resource Allocation Problem (RAP) and a Service Quality Control Problem (SQCP) by decomposing the problem. Yu et al. [20] proposed a new Hierarchical Intelligent Memory Fault Prediction (HiMFP) framework that predicts UCEs at multiple levels of a memory system and correlates it with memory recovery techniques that utilize CE addresses at multiple levels of memory (especially at the bit level) and constructs machine learning models based on spatial and temporal CE information. Luo et al. [21] introduced a fail-safe software design for a layered protection and validation perspective based on root cause analysis of more than 800 cloud reliability problems using an example of a hyperscale platform software, Intel Media Driver. Hu et al. [22] obtained the closed network reliability as a polynomial expression of link reliability using the Hop-State Algorithm (HSA) based on the Markov model. Jia et al. [23] formulated the service function chain scheduling problem in 5G networks supporting Network Functions Virtualization (NFV) as mixed integer nonlinear programming with the objective of maximizing the number of requests that satisfy the delay and reliability constraints, proposes an efficient algorithm to determine the redundancy of the VNFs while minimizing the reducing the delay, and using state-of-the-art reinforcement learning techniques to learn SFC scheduling policies to improve the success rate of service function chain requests.

Optimization algorithms

Fu et al. [24] proposed a genetic hybrid algorithm based on particle bee colony optimization, which divides the particle population of each generation, and uses the

swallowing mechanism and cross mutation of the genetic algorithm to change the position of the particles in the subpopulation, so as to expand the search range of the solution space; and then merges the subpopulations, which ensures the diversity of the particles in the populations, and reduces the probability of the algorithm falling into the local optimal solution; finally, the feedback mechanism will feedback the flight experience and the accompanying flight experience of the particles to the next generation particle population, so as to ensure that the particle population can always move towards the optimal solution. Finally, using the feedback mechanism, the flight experience of the particles themselves and the accompanying flight experience are fed back to the next generation of particle populations, to ensure that the particle populations can always move forward in the direction of the optimal solution. Rizvi et al. [25] proposed an algorithm named MFGA (Modified Fuzzy Adaptive Genetic Algorithm) to minimize makespan and improve resource utilization under deadline and budget constraints, and a fuzzy logic controller is designed to control the crossover rate and the mutation rate to prevent the MFGA from getting stuck in local optima. Senthil et al. [26] proposed a task allocation algorithm based on Cat Swarm Optimization and BAT algorithm. The BAT algorithm helps the CSO algorithm to get rid of the preconvergence problem. Gai et al. [27] proposed a heuristic elastic PSO algorithm, which adopts the A* algorithm to provide global guidance for path planning of large-scale grids. The elastic PSO algorithm utilizes contraction operations to determine the globally optimal paths formed by locally optimal nodes, so that the particles can converge quickly, and the particle diversity is ensured by the rebound operation in the iterative process.

To sum up, existing research on task offloading pays more attention to whether tasks can be decomposed into subtasks and how many tasks should be offloaded. Considering the performance limitations of the terminal devices, how to offload their tasks to an appropriate server with abundant resources is a challenging problem. Especially, how to predict the offloading necessity of terminal tasks in advance and how to formulate an effective offload strategy according to the attributes of terminal devices and edge nodes are also challenging issues. This is exactly the research work of this paper.

System model

This section describes the general task offloading architecture for CEE collaboration. As shown in Fig. 1, the CEE system architecture includes the cloud center layer on top, the edge server layer in the middle, and the end layer at the bottom.

(1) Cloud center layer: The cloud center layer comprises a cloud computing center consisting of centralized cloud servers located at a significant distance from the local equipment. These cloud servers possess abundant computing resources and offer rapid task response times. However, due to the distance factor, there exists a certain round-trip transmission delay for the transmission of computing results.
(2) Edge Layer: The Edge Layer is an edge cluster consisting of dispersed fixed-point edge servers close to the local equipment; edge server computing power and resources are weaker than cloud servers, but the advantage in transmission delay is obvious. There is an overlap in the communication coverage of edge servers within the coverage area of a single base station; one edge server is set up next to each base station, the number of which is all n. The set of edge servers $Edge = \{E_1, E_2, \ldots, E_n\}$.
(3) End layer: The end layer consists of many local terminal devices, which run computationally intensive applications or delay-sensitive applications and are limited in computational and storage resources due to the constraints of hardware, computing power, energy consumption, and other factors. The same end device may be within the communication range of different mobile edge servers. The set of end devices $UE = \{U_1, U_2, \ldots, U_m\}$

Under the CEE collaborative computation offloading system architecture, the local terminal device can choose to keep the computation task running at the local device, offload it to the edge server in the edge layer for execution, and offload it to the central cloud for execution.

We assume that a server can process several different types of tasks simultaneously. The main scenario in this paper has multiple edge nodes, a cloud center, and a macro base station, while terminal devices are overlaid around the macro base station, and users can offload tasks by uploading them to the macro base station. The macro station can receive multiple task requests at the same time because the macro station is a transit station, which facilitates users to offload their tasks to edge nodes and the cloud center.

Problem formulation

The system has a total of M terminal devices, each of which initiates a service request, and all tasks can be offloaded and scheduled. Each request T is described by three parameters (L,C,D), L denotes the amount of data (in bits) required to process the terminal task, C denotes the number of CPU cycles required to process each unit of data for that terminal task, and D denotes the maximum tolerable delay for

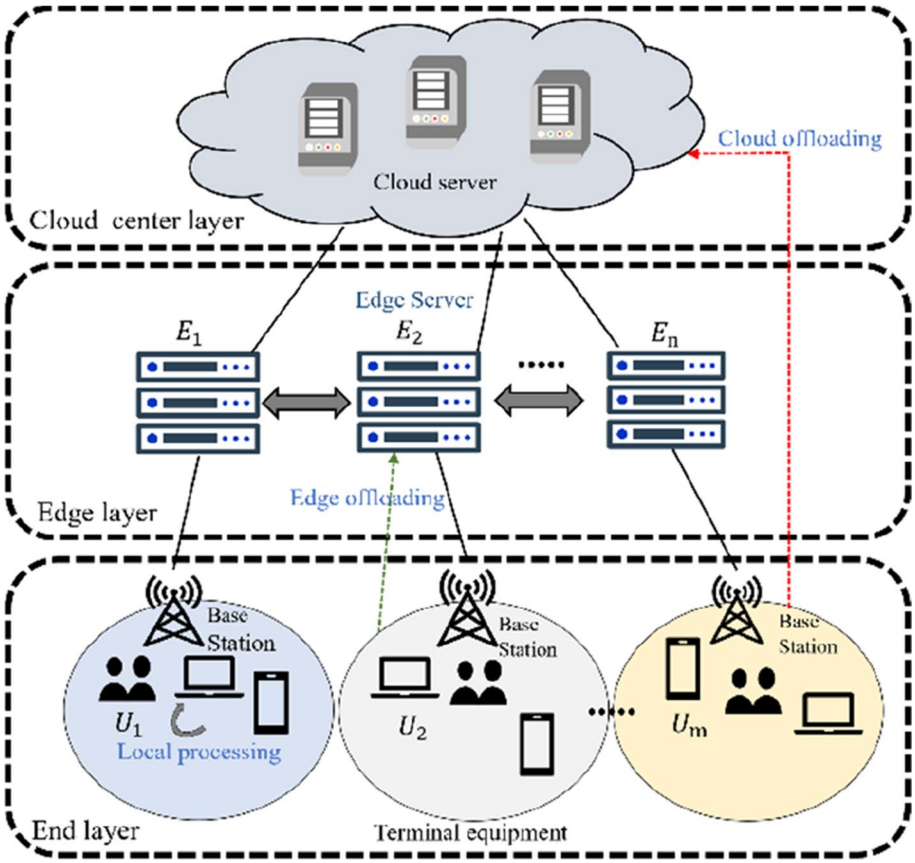

Fig. 1 System model

that task. Each node is described by the two parameters (f^{ser}, m^{ser}), f^{ser} denotes the processing power of the node and m^{ser} denotes the remaining memory of the node.

$$Task_i = (L_i, C_i, D_i)$$
$$Edge_i = (f_i^{ser}, m_i^{ser})$$

In this paper, we focus on the case where the terminal devices have limited computational resources and each terminal task cannot be further divided, each terminal device is equipped with only one antenna and can only transmit one terminal task at a time; all users share the system transmission bandwidth equally; therefore, the rate at which the terminal device transmits the task T to the edge node is denoted as:

$$r_{i,j} = \frac{W}{n} log_2\left(1 + \frac{pg_i^j}{\frac{W}{n}N_0}\right) \tag{1}$$

where the total transmission bandwidth is W and p denotes the transmission power of the end-device offloading task T to the edge server. g_i^j is the channel gain of the end-device and the edge server. N_0 is the noise power spectral density. Each task can choose to be executed locally or offloaded to the edge server or the cloud center for execution, therefore, we introduce an offloading decision variable $V = \{v_{i,j}\}$ to represent the task offloading situation. Set the number of cloud center and n edge servers to {0,1, ..., n}:

$$v_{i,j} = \begin{cases} 1 & \text{Perform task offloading} \\ 0 & \text{Task local execution} \end{cases} \tag{2}$$

If $v_{i,j}$ is 1 and j is 0, it is offloaded to the cloud center. If $v_{i,j}$ is 1 and j is not 0, it is offloaded to an edge server.

All edge devices are randomly and uniformly distributed next to the base station, and the distance obeys a uniform distribution $L \sim Unif([l_{min}, l_{max}])$, the total frequency of the edge server is F_i, the frequency assigned to the edge device U_i is f_i^{ser}, and the local CPU frequency of the user U_i is f_i^{user}, and the CPU frequency of the cloud center is f_{cloud}.

End Devices U_i Tasks initiated T_i at the execution time at the edge server is expressed as $t_{i,j}^{edge}$:

$$t_{i,j}^{edge} = \frac{C_i}{f_i^{ser}} \tag{3}$$

The local execution time of task T_i is expressed as t_i^{local}:

$$t_i^{local} = \frac{C_i}{f_i^{user}} \tag{4}$$

The cloud execution time of task T_i is expressed as $t_{i,0}^{cloud}$:

$$t_{i,0}^{cloud} = \frac{C_i}{f_{cloud}} \tag{5}$$

The transfer time of the task T_i offloaded to the edge server is expressed as $t_{i,j}^{tran}$:

$$t_{i,j}^{tran} = \frac{L_i}{r_{i,j}} \tag{6}$$

The transfer time of the task T_i offloaded to the edge server is expressed as $t_{i,0}^{tran}$:

$$t_{i,0}^{tran} = \frac{L_i}{r_{i,0}} \tag{7}$$

The transmission energy consumption of task T_i offloaded to the edge server is expressed as $e_{i,j}^{tran}$:

$$e_{i,j}^{tran} = t_{i,j}^{tran} p \tag{8}$$

The transmission energy consumption of task T_i offloaded to the cloud center is expressed as $e_{i,0}^{tran}$:

$$e_{i,0}^{tran} = t_{i,0}^{tran} p \tag{9}$$

The local execution energy consumption of task T_i is represented as e_i^{local}:

$$e_i^{local} = c_i f_i^{user2} C_i \tag{10}$$

Of which c_i is the terminal device U_i's effective capacitance factor, which depends on the chip architecture of the CPU.

The edge execution energy consumption of task T_i is expressed as $e_{i,j}^{edge}$:

$$e_{i,j}^{edge} = k_i f_i^{ser2} C_i \tag{11}$$

The cloud execution energy consumption of task T_i is expressed as $e_{i,0}^{cloud}$:

$$e_{i,0}^{cloud} = k_{cloud} f_i^{ser2} C_i \tag{12}$$

Then the total delay consumption of the whole offloading model system can be expressed as the sum of the delay of the task performing local computation and the delay of the task performing offloading, i.e., it is expressed as T:

$$\begin{aligned} T = \max\{&(1 - x_{i,j}) t_i^{local} + x_{i,j}\left(t_{i,j}^{tran} + t_{i,j}^{edge}\right) \\ &+ x_{i,0}\left(t_{i,0}^{tran} + t_{i,0}^{cloud}\right)\} \end{aligned} \tag{13}$$

Similarly, the total system energy consumption is expressed as E:

$$\begin{aligned} E = \sum_{i=1}^{m} &(1 - x_{i,j}) e_{i,j}^{local} + x_{i,j}\left(e_{i,j}^{edge} + e_{i,j}^{tran}\right) \\ &+ x_{i,0}\left(e_{i,0}^{cloud} + e_{i,0}^{tran}\right) \end{aligned} \tag{14}$$

The mathematical model described in this paper is a constrained joint optimization problem to optimize the offloading decision, offloading scheduling under the condition of limited server resources. Therefore, the optimization objective equation of the model can be expressed as:

α, β is calculated by entropy weighting method based on the historical processing data of the edge nodes, and $\alpha + \beta = 1$

$$\min W = \alpha T + \beta E \tag{15}$$

$$s.t.$$

$$f_{i,j}^{ser} \le F_i \tag{16}$$

$$v_{i,j} \in \{0, 1\} \tag{17}$$

$$T_i \le D_i \tag{18}$$

$$F_{Queue_i} \le m_i^{ser} \tag{19}$$

The objective function Eq. (15) represents a linear combination of system latency and energy consumption. Constraint (16) states that the computational processing power of the end device should not exceed the frequency of the edge server. Constraint (17) specifies that a task can only be offloaded to a single edge server. Constraint (18) indicates that the total latency of an endpoint task should not exceed its maximum tolerable latency, and if it exceeds, it should be offloaded to a cloud center for processing. Constraint (19) states that the amount of data of a pending task on a node should not exceed the remaining memory of that node. Problem (15) can be solved by finding the optimal offloading decision. However, the offloading decision vector V represents a feasible set of binary variables, and the objective function represents a non-convex problem. In this paper, we propose a joint solution to the offloading decision problem on terminal devices by combining service reliability analysis method and an improved bee colony optimization method.

Algorithm implementation

In this section, we first propose the system model of a Service Reliability Analysis and Elite-Artificial Bee Colony Offloading. Then, we describe SRA and E-ABCO in detail.

SRA-E-ABCO model

The bee colony algorithm is utilized as an optimization method to determine the best offloading solution for tasks by constructing an optimization model (shown in Fig. 2) that meet user needs and system reliability requirements. The model comprises two significant components: (a) Service Reliability Analysis (SRA): This component takes the service request of the end device as input and generates a set of offloading decision vectors, known as the elite population, as output. It predicts whether the end task should be offloaded or not through the LR model. Subsequently, it calculates the correlation degree between the edge nodes and the matching degree of the end device with the edge node. This calculation employs the service processing records stored in the edge nodes to derive a set of offloading vectors for the elite population. (b) Elite-Artificial Bee Colony Offloading (E-ABCO): This component takes the elite population obtained from the service reliability analysis method as part of the input and yields the optimal offloading decision vectors as output. The population is initialized using inverse learning and tent mapping techniques. Furthermore, the location update formula is enhanced by the elite population, and the adaptation formula incorporates delay and energy consumption considerations to ensure load balancing among nodes. Ultimately, this process yields a set of optimal offloading decision vectors.

Service Reliability Analysis (SRA)

As shown in Fig. 2a, this method first employs the LR model in machine learning to model and predict whether a task on the corresponding end device should be offloaded or not. Subsequently, it obtains a set of reliable offloading decision vectors by conducting reliability analysis on the surrounding edge servers available for offloading. The LR model and the service reliability analysis are detailed below.

The logistic regression model is essentially a modified maximum likelihood estimation model with a binomial categorical dependent variable. When offloading tasks from a terminal device for classification purposes, it is common to use the decision of whether to offload or not as the dependent variable (0 indicates not to offload, while 1 indicates to offload). The attributes of the terminal device, such as CPU utilization, memory utilization, remaining power, and network bandwidth, serve as the independent variables, represented as $x = (x_1, x_2, x_3, x_4)$.

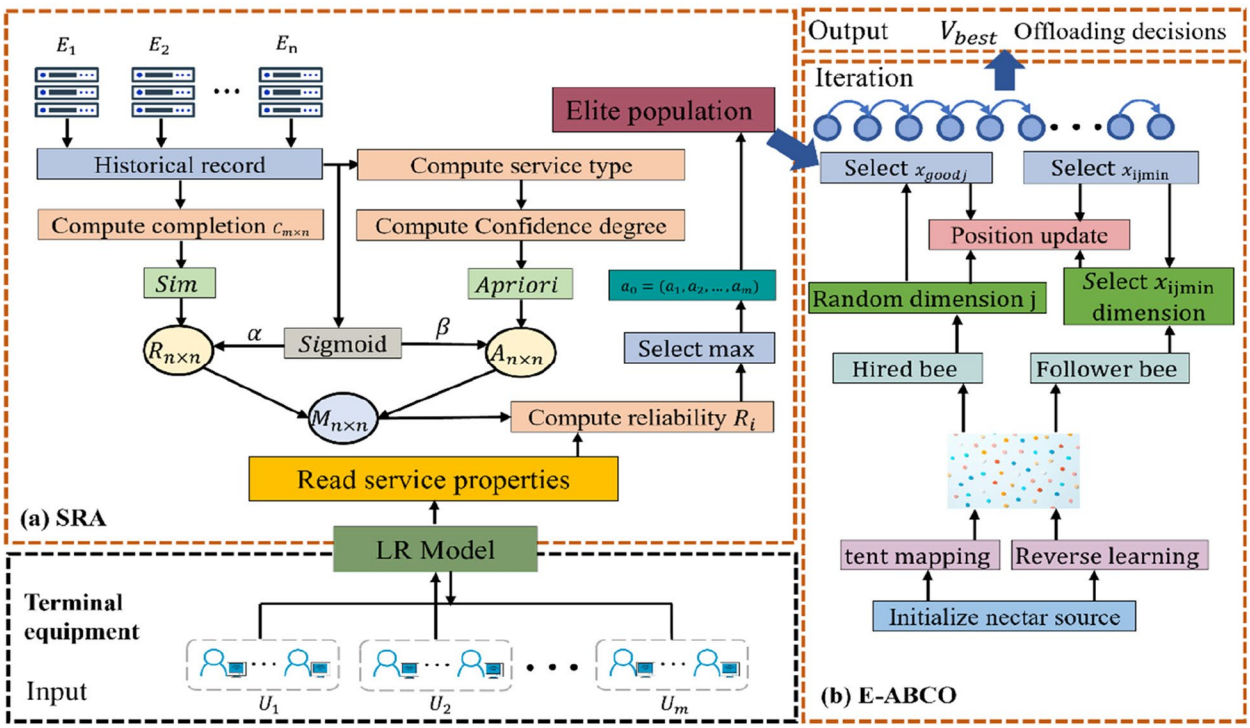

Fig. 2 SRA-E-ABCO Model

The training dataset in this paper is based on the Kaggle real dataset of cloud-fog computing from the official website. The dataset is binary classified (offloaded and not offloaded) by unsupervised learning to provide the training dataset for the LR model. Finally, we get the training dataset as $x = (x_1, x_2, x_3, x_4, x_5)$, where x_1 is the CPU utilization, x_2 is the memory utilization, x_3 is the remaining power, x_4 is the network bandwidth, and x_5 is a label indicating whether the task needs to be offloaded to the cloud for processing. The value is 0 or 1, where 0 means that the task is computed locally on the end device, and 1 means that the task needs to be offloaded.

When making task offloading decisions, in addition to considering service reliability, we also need to consider the resource utilization of the terminal device. For this reason, we can predict whether the task should be offloaded to the cloud for processing by analyzing attributes such as CPU utilization, memory utilization, remaining power, and network bandwidth of the end device. The following section describes how to make predictions using the LR model. LR, or Logistic Regression, is a probability-based classification algorithm that can be employed for predicting binary classification problems. In this paper, we aim to predict whether a task should be offloaded to the cloud for processing or not using the LR model. The prediction formula for the LR model is provided below:

$$h_\theta(x) = g\left(\theta^T x\right) \tag{20}$$

where θ is a model parameter, x is a sample feature vector, and $g(z)$ is a sigmoid function with the expression:

$$\begin{gathered} g(z) = \frac{1}{1+e^{-z}} \\ z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_\theta x_4 \end{gathered} \tag{21}$$

When performing model training, we need to define the loss function. In the LR model, the commonly used loss function is the logarithmic loss function with the expression:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[\begin{array}{c} y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) \\ +\left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right) \end{array}\right] \tag{22}$$

where m is the number of samples, y is the sample label, and $h_\theta(x)$ is the model prediction value.

For model training, we need to use the gradient descent algorithm to minimize the loss function. The update formula for the gradient descent algorithm is:

$$\theta_j := \theta_j - \alpha m \sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x^{(i)} \tag{23}$$

where α is the learning rate, which controls the step size of each parameter update.

In this paper, we can consider attributes such as CPU utilization, memory utilization, remaining power, and network bandwidth of the terminal device as the input features of the LR model. The output label of the LR model would indicate whether the terminal task should be offloaded to the cloud for processing. By training and predicting with the LR model, we can determine whether the terminal task should be offloaded, thereby making the decision on terminal task offloading more rational.

Based on the results predicted by the LR model, the terminal tasks earmarked for offloading are analyzed for service reliability as follows: The definition of service reliability analysis involves scrutinizing services processed by the edge nodes with regard to delay. It assesses the matching degree of existing service requests and similar service requests corresponding to the edge nodes, ultimately determining service reliability by analyzing the matching degree of the edge nodes. This section primarily focuses on utilizing historical processing data cached by the edge nodes through a series of matrix variations and analyses. The goal is to obtain a set of offloading vectors for subsequent use as elite populations.

In this paper, we make use of the existing user task dataset to initialize ten sets of service request records that have been processed at each edge node in the following approximate format: (*1, type, maxtime, truetime*), where *truetime* is the actual processing time at the edge node, and we add a data attribute to perform better service reliability analysis completion:

$$completion = \frac{maxtime - truetime}{maxtime}$$

It is used to indicate the degree of completion (user satisfaction) of the service, so we end up with the following actual data:

(*1, type, maxtime, truetime, completion*)

It is assumed that each end-user device has had a service request data record at each edge node.

Firstly, calculate the connectivity correlation between edge nodes. The user satisfaction matrix of service processing results $C_{m\times n}$ is constructed by completion, as shown in Table 1. where m is the number of terminal devices and n is the number of edge nodes.

The service similarity between the historical data of the edge nodes is calculated using Eq. (24).

$$Sim = \cos\theta = \frac{\boldsymbol{k}_i \times \boldsymbol{k}_j}{|\boldsymbol{k}_i| \times |\boldsymbol{k}_j|} \tag{24}$$

where $\boldsymbol{k}_i$ and $\boldsymbol{k}_j$ are the i th edge node and the j th edge node, respectively, i.e., the column vectors consisting of the data in column i and column j of Table 1.

Table 1 User-edge server satisfaction matrix

Terminal equipment	Edge node			
	e_1	e_2	...	e_n
u_1	C_{11}	C_{12}	...	C_{1n}
u_2	C_{21}	C_{22}	...	C_{2n}
⋮	⋮	⋮	⋮	⋮
u_m	C_{m1}	C_{m2}	...	C_{mn}

Calculate the correlation between edge nodes. Firstly, using the service types of the edge nodes in the dataset, the matrix of service types corresponding to the edge nodes is obtained. And the confidence level of the frequent binomial set is obtained by calculating *conf*. Then the confidence of the frequent binomial set is obtained using the *Apriori* algorithm and its related principle (Eq. 25) to find the correlation between edge nodes and form the correlation matrix $association_{n\times n}$.

$$association(i,j) = \frac{2conf(i,j) * \mathrm{con\,f}(j,i)}{\mathrm{conf}(i,j) + \mathrm{con\,f}(j,i)} \tag{25}$$

Finally, the edge node matching degree is calculated, i.e., the similarity and correlation between edge nodes are weighted and fused as shown in Eq. 26, where the weight coefficients are determined using the *sigmoid* function is determined as shown in Eq. 27.

$$\begin{aligned} M(i,j) &= \alpha(i,j) \times \mathrm{Sim}(i,j) + \\ &(1-\alpha(i,j)) * \mathrm{association}(i,j) \end{aligned} \tag{26}$$

$$\alpha(i,j) = 2 \times \left(1 - \frac{1}{1+e^{-|Iu|}}\right) \tag{27}$$

Where α is the weighting factor that Iu denotes the number of times a service request issued by a user is processed at the edge node i and j number of times it is processed. Calculate the edge node matching degree matrix $M_{n\times n}$, as shown in Table 2.

Then when the end device issues a new service request, the task type is analyzed and filtered with all the resource types of the edge node, the edge node with the corresponding task type is selected, the service reliability of the corresponding task is calculated $R_i = \frac{\sum_1^n M_{ik}}{n}$, and set the offload vector $a = (a_1, a_2, \ldots, a_k)$, the $a_i = maxR_i = \frac{\sum_1^n M_{ik}}{n}$, through the above service reliability analysis, a set of offloading vectors is obtained $a_0 = (a_1, a_2, \ldots, a_k)$, which will be used as the elite population of the bee colony algorithm.

Table 2 Edge node matching matrix

Edge node	e_1	e_2	...	e_n
e_1	M_{11}	M_{12}	...	M_{1n}
e_2	M_{21}	M_{22}	...	M_{2n}
⋮	⋮	⋮	⋮	⋮
e_n	M_{n1}	M_{n2}	...	M_{nn}

At the same time, delay limits are set so that the maximum tolerable delay cannot be exceeded, the edge server resource limit cannot be exceeded, and the minimum reliability is offloaded to the cloud center for processing.

Elite Artificial Bee Colony Offloading (E-ABCO)

The optimization process for the terminal task offloading strategy based on the bee colony algorithm is as follows: we start by initializing each honey source and assigning a hired bee to each source. The hired bee searches for new honey sources using a specific search method. Then, based on the adaptability of the honey sources, we calculate the probability that the honey source found by the hired bee will be followed by a subsequent bee. The subsequent bee then searches for new honey sources using its own search method. As shown in Fig. 2b, this iterative process continues until the maximum number of iterations is reached.

The honey source update formula of the basic artificial bee colony algorithm involves randomly searching for a new honey source around the target honey source, leading to slow convergence due to the overly blind search. In this paper, we borrow the variation operator from the differential evolutionary algorithm and use the neighboring nectar sources as a guide when searching for new nectar sources. The improved nectar update formula searches more purposefully and has stronger local search ability. At the same time, the honey source fitness formula in this paper is improved. The fitness formula of the basic artificial bee colony algorithm sometimes does not truly reflect the goodness of honey sources, which seriously affects the optimization accuracy of the algorithm.

The improved fitness equation is as follows:

$$f_i = \frac{\alpha T_i + \beta E_i}{1 + Q_i - W_i} \tag{28}$$

In the fitness formula, the reliability of the service mainly depends on the number of current service resources remaining in the assigned edge node Q, the number of services in the waiting queue W, and the latency of processing the task T and the energy consumption required to perform that service E. Where α and β represent the weights of latency and energy consumption, respectively (calculated using the entropy weighting method described above), with the closer to 1 indicating the greater importance placed on them. The more the number of remaining services, the less the number of services waiting for the resource, the

less the energy consumption of executing the service, and the less the delay of executing the service i.e., the higher the reliability of the service, the more the reliability formula improves the fitness formula, thus optimizing the efficiency of the search for an excellent honey source.

$$fitness_i = \begin{cases} \frac{1}{1+f(X_i)}, f(X_i) \geq 0 \\ 1 + \left|f(X_i)\right|, otherwise \end{cases} \tag{29}$$

Assuming that there are$n$ edge nodes, then at each edge node is equal to one nectar source, i.e., there are n number of offloading decision vectors generated. $X_i = (x_1, x_2, \ldots, x_m)$ where $x_i = j$ denotes the terminal deviceui offloaded to the first$j$ edge node for processing. In order to enable the artificial bee colony algorithm to be applied to the task offloading problem in the CEE scenario, the following mapping relationship needs to be established to represent the offloading decision by the location of the nectar source, and transform the process of searching for the offloading decision into the process of searching for the nectar source by the artificial bee colony algorithm by using the *Tent* mapping to generate the initial population, which does not change the nature of randomness that is present at the time of initialization, but maintains the diversity of the initial population.

$$x_n = \begin{cases} \frac{x_{n-1}}{a}, 0 \leq x_{n-1} \leq a \\ \frac{(1-x_{n-1})}{1-a}, a < x_{n-1} \leq 1 \\ x_0 \text{ is a } (0,1) \text{ random number} \end{cases} \tag{30}$$

According to the chaotic sequence obtained from the above equation, mapping it to the range of values of the variable[1, n] to obtain the mapped population X_i The mapped population is obtained. Then the individuals in the search space are relocated to more promising regions through reverse learning to help the algorithm jump out of the local optimum:

Reverse learning is defined as follows: suppose $X_i = (x_1, x_2, \ldots, x_m)$ is any point in the D-dimensional search space, and $X_i \in [1, n]$, then X_i The inverse point of is defined as $oX_i = (ox_1, ox_2, \ldots, ox_m)$ where oxi is computed according to the following equation:

$$ox_i = 1 + n - x_i \tag{31}$$

Based on the fitness function values simultaneously evaluate the current *Tent* mapping population and the reverse population and selects the more favorable n individuals as the initial population. In this way, the diversity of the initial population can be improved, thus accelerating the convergence of the algorithm.

Improvements to the position updating formulae for employed bees are obtained:

$$V_{ij} = X_{ij} + \varphi\left(x_{goodj} - X_{ij}\right) + \delta(X_{ik} - X_{vk}) \tag{32}$$

where x_{good} is an elite individual, a randomly selected individual from the elite population that ranks in the top five in terms of fitness for everyone in the a_0 offloading vector above.

$$V_{ij} = X_{ijmin} + \varphi\left(X_{ijmin} - X_{vj}\right) \tag{33}$$

where X_{ijmin} is the least adapted individual in that nectar source.

According to the nature of the three types of bees, a two-bees search strategy is proposed, i.e., after the employed bees have found a nectar source, the position update is carried out through Eq. (32), due to the random nature of the bee colony algorithm, a single random position update is prone to make a premature fall into the local optimum. Therefore, this paper alters the position updating formula of the follower bees, and the follower bees carry out a local optimization of this nectar source through Eq. (33), thus preventing a premature falling into local optimum.

That is, the position update formula for the two-bees search is proposed using Eqs. (32) and (33), after the hiring bees are updated, the following bees position update selects the individual with the smallest fitness in the current offloading decision, and the position update is carried out according to Eq. (33), and the cycle is carried out sequentially until the number of iterations is reached.

When searching for a new nectar source, a greedy selection strategy is followed by first initializing the $V_{best} = X_1$:

$$V_{best} = \begin{cases} V_{ij}, V_{ij} \geq V_{best} \\ V_{best}, V_{ij} < V_{best} \end{cases}$$

After a certain number of iterations of optimization, a set of offloading decision vectors V_{best} is finally obtained, i.e., the service requests initiated by the terminal devices are offloaded according to V_{best} to satisfy the user's needs.

Coding of SRA-E-ABCO in cloud-edge-end environments

In order to enable the artificial bee colony algorithm to be applied to the edge computing offloading problem, the following mapping relationship needs to be established to represent the offloading decision by the location of the nectar source, and the process of searching

for the offloading decision is transformed into the process of searching for the nectar source by the artificial bee colony algorithm, so it is necessary to discretize the artificial bee colony algorithm.

The offloading decision vectors are coded for CEE environments based on M terminal mobile devices, N edge servers and 1 cloud server. That is, 0 is used to denote local computation, *1, 2...N* denotes offloading to the corresponding numbered edge server computation, and $N+1$ number denotes offloading to the cloud server computation. Discrete processing of the nectar initialization formulas (30) and (31) is obtained:

$$A_i = X_i * (N+1) \tag{34}$$

where X_i is the initialised population of [0, 1] obtained from the Tent mapping and inverse learning of Eqs. (30) and (31) above, which is mapped to between [0, N+1] by the above formulae to apply to the task offloading of the cloud-side-end scenario. The final nectar A_i is obtained, and each nectar A_i represents an offloading decision vector with 1 row and M columns, and the values of the elements in the vector indicate the locations where the device needs to offload the tasks.

Similarly, the position update Eqs. (32) and (33) are discretized so that their position updated values are also mapped between $[0, N+1]$.

Obtained by discretizing the position update Eq. (32):

$$V_{ij} = \begin{cases} \left(X_{ij} + \left[Y_{ij}\right]\right)\%(N+2), \left[Y_{ij}\right] \geq 0 \\ \left(X_{ij} + \left[Y_{ij}\right] + N + 2\right)\%(N+2), otherwise \end{cases}$$
$$\left[Y_{ij}\right] = X_{ij} + \varphi\left(x_{goodj} - X_{ij}\right) + \delta(X_{ik} - X_{vk}) \tag{35}$$

Where $\left[Y_{ij}\right]$ is the term in Eq. (35) other than X_{ij} the other terms in Eq. (35), i.e., equal to the increment of displacement.

Obtained by discretizing the position update Eq. (33):

$$V_{ij} = \begin{cases} \left(X_{ij} + \left[Y_{ij}\right]\right)\%(N+2), \left[Y_{ij}\right] \geq 0 \\ \left(X_{ij} + \left[Y_{ij}\right] + N + 2\right)\%(N+2), otherwise \end{cases}$$
$$\left[Y_{ij}\right] = X_{ij} + \varphi\left(X_{ijmin} - X_{vj}\right) \tag{36}$$

Since the range of A_i obtained by encoding the nectar source is [0, N+1], and φ and δ are both random numbers of [-1, 1], the range of $[Y_{ij}]$ is [-N-1, N+1]. If $[Y_{ij}]=0$, it means that the offloading is still in the original position, if $[Y_{ij}]>0$, it means that it is necessary to move the computational position of the device backward by $[Y_{ij}]$ bits, and then take the remainder of $N+2$, and a loop structure is constructed to ensure that the update is still an integer of $[0, N+1]$. If $[Y_{ij}]<0$, then move $[Y_{ij}]$ forward accordingly, as shown in Eqs. (35) and (36).

Algorithm 1. SRA-E-ABCO

***Input*:**
Elite population vector a_0.
***Output*:**
Optimal offloading decision vector V_{best} after iterative optimization.
Step 1:
Initialization of nectar source parameters, i.e., initialization assignment of n nectar sources
using Eqs. (30) and (31) N initialized nectar sources were generated by selecting the top n
individuals in the fitness rankings of the populations and reverse populations to form a nectar.
source, and so on.
Step 2:
Discrete mapping of nectar sources through Eq. (34) yields n nectar sources applicable to the
cloud-side-end scenario.
Step 3:
Start iteratively improving the bee colony algorithm by constantly updating the offloading decision.
vector V_{best}.
while $t \leq G_{max}$**(Maximum number of iterations of the iteration not reached)**
while $i \leq FoodNumber$
$random\ k$
The position update is performed according to Eq. (35), and the kth dimension of the ith.
nectar source is selected for the update.
$If\left(f(V) > f(V_{best})\right)$
$V_{best} = V$
$End\ if$
end while
while $i \leq FoodNumber$
The following bee uses a roulette algorithm to decide whether to follow or not to follow.
The least adapted individual s for this nectar source was calculated, and the s-dimension was updated based on Eq. (36)
$If\left(f(V) > f(V_{best})\right)$
$V_{best} = V$
$end\ if$
end while
end while
return V_{best} The optimal offloading vector is obtained, and the end-device tasks are scheduled for offloading.

SRA-E-ABCO analysis

To validate the efficacy of the algorithm in high-dimensional scenarios, experiments were performed in 30 and 50 dimensions using three test functions obtained from IEEE CEC 2017. Among these, F1 represents a single-peak function, while F8 and F9 represent multi-peak functions. The proximity of the algorithm's optimization outcomes to the optimal values of each test function serves as a measure of the algorithm's optimization capability.

SRA-E-ABCO was compared with the classical ABC algorithm and the improved swarm algorithm GABC for experiments, setting 30-dimensional time $G_{max} = 1000$ times, 50-dimensional time $G_{max} = 2000$ times, initialised

in a uniform way to ensure fairness, each algorithm solves the 3 test functions independently and records the final finding results. Repeat the above process for 30 times and calculate the mean and standard deviation of the final optimization results of each algorithm on the 3 test functions.

The results obtained are shown in Tables 3 and 4, where the data in Table 3 are the mean and standard deviation of the three optimization algorithms ABC, GABC, and SRA-E-ABCO after the optimization calculations on 30 dimensions for the three test functions F1, F8, and F9 respectively. The data in Table 4 shows the mean and standard deviation of the three optimization algorithms ABC, GABC, SRA-E-ABCO after optimization computation on 50 dimensions for the three test functions F1, F8 and F9 respectively. The bolded part is the optimal value, and there are cases in the table where the values are the same but not bolded, which is because there is rounding when the data are counted by scientific notation, and the real value will be slightly larger than the bolded value. From the table, it is obvious that the improved swarm algorithm SRA-E-ABCO in this paper has achieved better results on the data of the three test functions.

In the above two tables, the first column is the name of the function, columns 2–4 are the experimental data of the three optimization algorithms, column 5 is the minimum value of the three functions, and the last row is the number of times that the three optimization algorithms achieved the best experimental results. The experimental results show that at 30 dimensions SRA-E-ABC is able to approximate the respective minima of the three test functions faster, but the advantage over the other two optimization algorithms is not very obvious. While at 50 dimensions, the advantage of SRA-E-ABC is gradually manifested with the increase of the number of dimensions and iterations, and after 2000 iterations, SRA-E-ABCO is obviously closer to the convergence values of the three test functions.

The following convergence curve graph to show the convergence effect, because with the increase in the number of iterations, the function of the value of the jump is large, so through the lg function to shrink, so that it is easier to reflect the experimental effect from the line graph. As shown in the Fig. 3 below, the convergence curves of F1 single-peak test function and F9 multi-peak test function in 30 and 50 dimensions respectively are shown. In this paper, to better demonstrate the difference in convergence accuracy through the line graph, the maximum number of iterations is set to 0–8000 times. The experiments show that SRA-E-ABCO has better optimal searching ability and is more advantageous for high-dimensional problems.

Experimental results and analysis

In this paper, we use IntelliJ IDEA Community Edition 2022.3.1 for computational migration system model simulation construction of MEC to perform experimental simulation of the proposed SRA-E-ABCO method.

Datasets

The dataset selected for this paper is *Kaggle* Cloud-Fog Computing dataset in the official website, this dataset is generated by selecting 13 nodes and the number of tasks starts from 40 and increases in intervals of 40 tasks and increases to 120, which also includes the details of the tasks and the details of the nodes.

Table 3 Comparison of 30-dimensional algorithm results (means ± std.)

Function	ABC	GABC	SRA-E-ABCO	F_{min}
F1	1.95E-09 ± 2.24246E-09	8.77E-08 ± 1.07647E-07	**1.20076E-11 ± 4.62515E-11**	0
F8	-1.19E + 04 ± 1.80E + 02	-1.20E + 04 ± 1.65E + 02	**-1.21E + 04 ± 1.38E + 02**	-12,569.5
F9	1.221643497 ± 1.0480	4.10E-01 ± 5.16E-01	**3.91E-05 ± 0.00010177**	0
Best	0	0	**3**	

Table 4 Comparison of 50-dimensional algorithm results (means ± std.)

Function	ABC	GABC	SRA-E-ABCO	F_{min}
F1	1.23E-11 ± 1.9111E-11	6.32E-09 ± 1.27068E-08	**3.08E-14 ± 7.18437E-14**	0
F8	-1.99E + 04 ± 1.10E + 02	-2.01E + 04 ± 1.71E + 02	**-2.03E + 04 ± 1.66E + 02**	-20,949.5
F9	1.428443 ± 1.112589	0.95233338 ± 0.493758465	**4.84E-07 ± 4.817E-07**	0
Best	0	0	**3**	

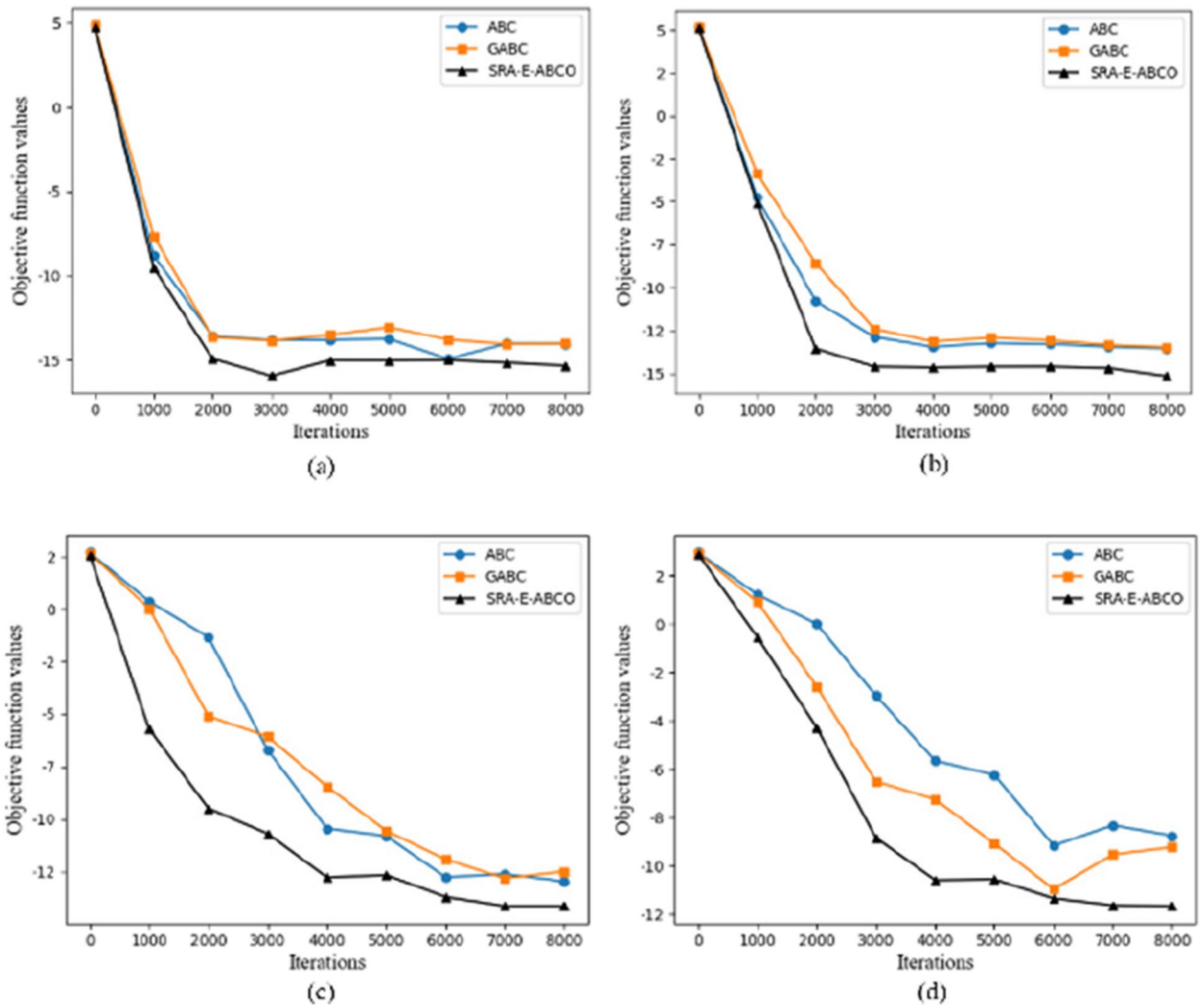

Fig. 3 Convergence plot of test function curves

Baseline schemes

To evaluate the performance of the proposed computational offloading strategy, we introduce the following six baselines:

- Only in Local (OL): the mobile device performs all computing tasks locally at the highest CPU frequency.
- All offloaded to edge node execution (All in MEC, AM): all computational tasks are fully offloaded to the edge servers for execution without executing any tasks on the end devices and cloud servers.
- Greedy Computation Offloading (GCO) [28]: the mobile device selects the least costly execution for each computational task based on execution delay and transmission delay.
- Deep neural network-based offloading methods (DNN models, DNNs) [29]: use learning between multiple deep neural networks (DNNs) to generate task offloading policies, and rank and reasonably allocate resources based on the principle of stack.
- Directed Artificial Bee Colony Algorithm (DABC) [30]: temporarily searching for a solution during the hiring and following bee phases and providing a solution to the ABC algorithm provides directional information and explores 20% of the surrounding neighborhood nectar sources, synergizing the computational resources at the edge end.

Experiments are conducted to investigate the effects of different number of tasks on the system latency and overall overhead, different number of edge nodes,

incremental increase in data volume of end tasks and incremental increase in CPU frequency of edge servers on the overall system latency and overhead.

Experimental parameter settings

In this paper, we consider CEE environments with Macro Base stations (MBS), several Small Base stations (SBS) and a data cloud center within a certain range, and each SBS deploys a MEC server with moderate computing power. Without loss of generality, the number of SBS is set to 10–30, that is, there are 10–30 MEC servers. This paper considers a certain number of users through the macro station unified offloading to the corresponding MEC server, and ignore the delay of offloading tasks from terminal devices to the macro base station, and the delay of the macro base station to each edge server is different. The main parameters are shown in the Table 5:

Results and analysis

The experiment first compares the impact of offloading latency and overall system overhead for different numbers of tasks in several task offloading methods.

As shown in Fig. 4, the offloading delay of all six methods increases by different magnitudes as the number of tasks increases from 40 to 120, and it can be seen from the figure that the other four methods are significantly better than the OL and AM methods. Among the six methods, the offloading delay of the OL method is particularly large, and the increase of the offloading delay of the OL method is the largest with the increase of the number of tasks, which is because all its tasks are processed locally on the terminal device and are limited by the CPU computation frequency of the terminal device,

Table 5 Basic simulation parameters

	Parameter	Value
m	Number of terminal devices	[40, 120]
n	Number of edge nodes	[10, 30]
g_i^j	Channel gain	$127 + logd$
C	The number of CPU cycles requested by the service	[0, 1]
N_0	Noise power in uplink bandwidth	2×10^{-13}
W	System bandwidth	15 MHz
P_i	Terminal transmission power	1w
T_i^{max}	Maximum time delay	1–3 s
f^{ser}	Terminal equipment processing capability	4–8 GHz
f_{edge}	Processing power of the edge server	16–24 GHz
f_{cloud}	Processing power of the cloud center	30 GHz
c_i	Terminal device CPU energy coefficient	10^{-25}
k_i	Edge server CPU energy coefficient	10^{-26}
a	Learning rate	0.0001

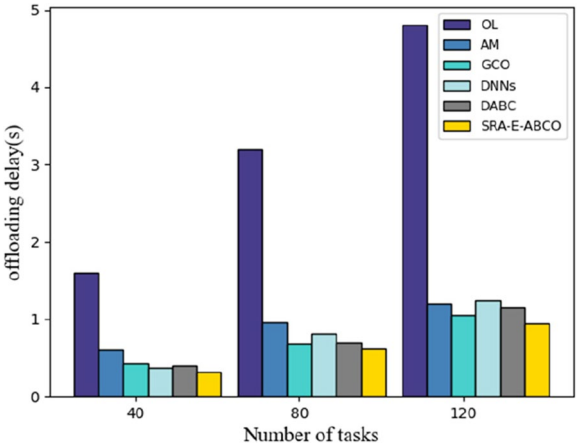

Fig. 4 Plot of offloading delay for different number of tasks

so the overall offloading performance of the OL method is the poorest. The AM method is to offload all the tasks from the terminal device to edge nodes for execution, and he makes full use of the rich computing resources of edge nodes, so the latency of processing tasks is much lower than that of OL. GCO offloads the tasks to the edge nodes that have the smallest computation time for completing the computation, and because of the different priorities of the tasks, it will result in the waiting time of the tasks with low priority being too long, but because the number of tasks in the latency scenario of this paper is not a large number of tasks, the waiting latency for the tasks with low priority is not very long.

This is also the advantage of GCO in this scenario. DNNs method has a slightly higher system overhead when the number of tasks is 120 since the task latency is supposed to be smaller in the single base station multi-user scenario, and since it self-predicts the task offloading labels with a slight error. When the number of tasks is small, its offloading efficiency is higher, and as the number of tasks gradually increases, the error gradually increases. The DABC method is a nectar source directed exploration algorithm with better optimization capability, which fully coordinates the computational resources of end devices and edge nodes in this scenario, but its optimization capability is slightly worse compared to SRA-E-ABCO because it only explores 20% of the surrounding nectar sources. The SRA-E-ABCO method in this paper shows high performance with different number of tasks, and its advantage is more obvious as the number of tasks increases.

Figure 5 shows the effect of different number of tasks on the overall overhead of the system, the overall overhead is mainly a blend of delay and energy consumption. With the increase of the number of tasks, the overall

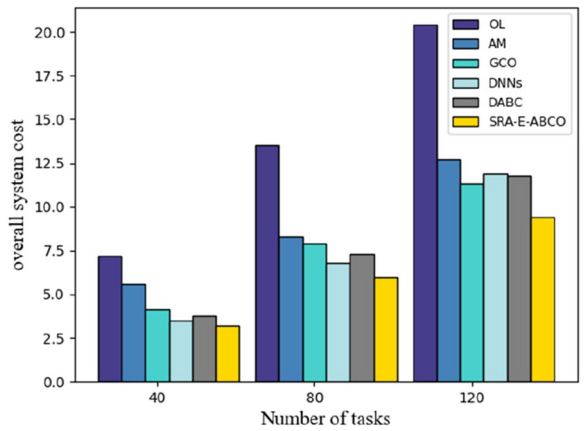

Fig. 5 System overhead graph for different number of tasks

system overhead of the six methods is increasing, but the increase is different in which the OL method has the largest increase in overhead; the other five methods can be clearly seen in this paper, the SRA-E-ABCO method has the smallest increase, the GCO can have a better performance in the delay above, but it ignores the impact of the energy consumption on the overall overhead, in this paper, the SRA-E-ABCO method integrates the offloading delay and the energy consumption to achieve a certain advantage in the overall overhead of the system. The SRA-E-ABCO method in this paper considers the offloading delay and energy consumption and achieves a certain advantage in the overall overhead of the system.

At a task number of 40, the data volume size of the offloaded tasks is gradually increased. Figure 6 shows the change in the overall system overhead from a 20% increase in the task data volume size to an 80% increase. Since under the OL method, the end device will only process one task for a period of time, its overall system overhead increases with a positive correlation trend as the task data volume increases. The performance of the two methods, GCO and AM, is more similar, and the greedy strategy always offloads the tasks to the edge servers with higher CPU computation frequency, and its drawback, i.e., the waiting latency, is gradually revealed as the task data volume increases, so the GCO has the largest increase in system overhead when the data volume increases to 80%.The prediction accuracy of DNNs decreases when the task data volume increases because there are fewer tasks with similar data volume in the training dataset in its training model, so its system overhead increases more when the task data volume increases. The DABC method explores the surrounding during the search of nectar 20% of the neighborhood and then optimizes the offloading strategy, and the following bee will replace the offloading node with the least adapted one, which is also the task with the highest offloading latency, so its offloading overall overhead is better than the other baseline methods. Finally, under this paper's SRA-E-ABCO method, the overall overhead of the system grows steadily, which indicates that this paper's method effectively synergizes the computational resources of the cloud, edge, and end, and by improving the bee colony algorithm and considering both latency and energy consumption, an efficient offloading decision is formulated, so this paper's algorithm outperforms the other five baselines.

At a task number of 40, the number of edge nodes is gradually increased. Figure 7 shows the change in the overall overhead of the system for increasing the number of edge nodes from 10 to 30. Since OL is always processed locally at the end device, the increase in the number of edge nodes has no effect on its overall overhead. The AM method offloads all the tasks to the edge nodes for processing, and since the waiting delay

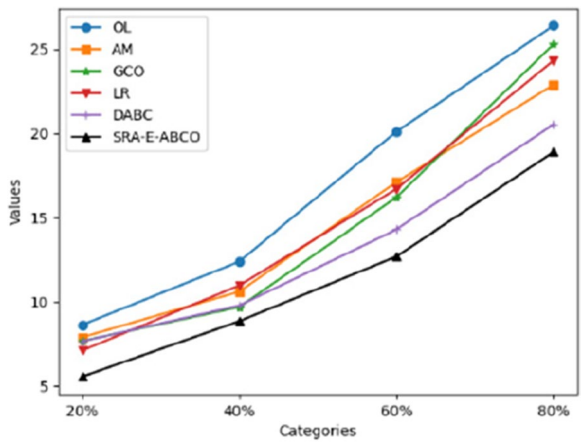

Fig. 6 System overhead diagram for incremental task data volume

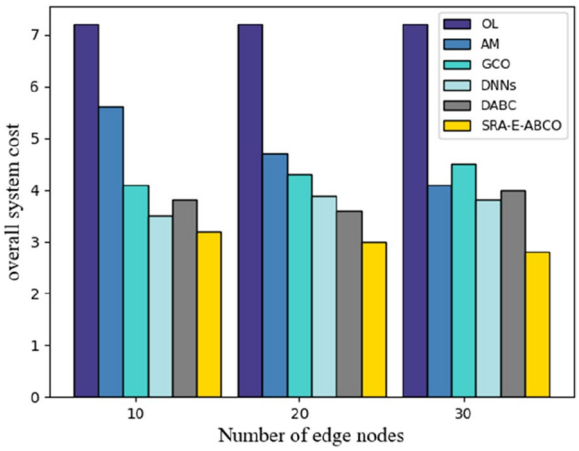

Fig. 7 System overhead graph for different number of edge nodes

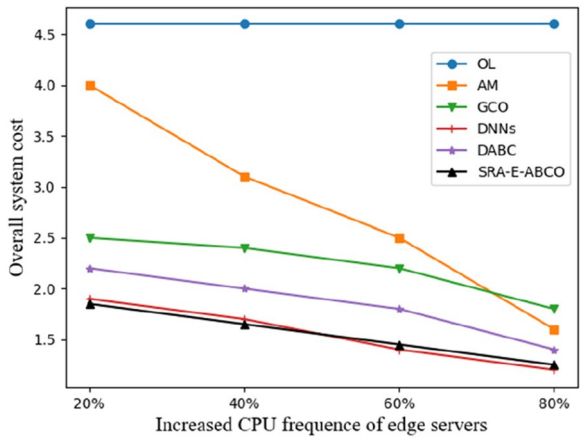

Fig. 8 Offload Latency Plot for Edge Server CPU Frequency Increments

decreases with the increase in the number of edge nodes, there is a certain reduction in the overall system overhead of the AM method. The GCO method always seeks for edge nodes that have a high CPU computation frequency node, so there is a corresponding increase in the transmission energy consumption of the edge nodes that may be increased, so there is a corresponding increase in the overall overhead of GCO. the DABC method only explores 20% of the neighborhood each time it explores the nectar, in its algorithm, the edge nodes are the nectar, so when the number of edge nodes increases that is, when the nectar becomes more, its optimization efficiency will be unstable, so there is an increase and decrease in its system overhead.

As shown in Fig. 8, in the case of constant number of tasks and gradual increase in the CPU frequency of edge servers, excluding the OL offloading method, the rest of the methods show an overall decreasing trend. the AM method, due to offloading all the terminal tasks to the edge servers, has a gradual edge-smaller offloading latency with the increase in the edge servers' processing power, but with higher randomness. the GCO and DABC have no significant advantage in the case of increasing edge servers' CPU frequency increment, the advantages of the two methods are not obvious, GCO favors edge servers with higher CPU frequency, resulting in higher transmission delay, so the offloading delay decreases by a smaller margin, while DABC cannot reflect its advantages when the number of tasks is small due to its exploration of 20% of the nectar sources. DNNs and this paper's SRA-E-ABCO, in the case of changing only the CPU frequency of edge servers, the advantage is more obvious, since DNNs trains the model based on task attributes, changing the edge server CPU frequency does not degrade its training model performance, whereas in this paper SRA-E-ABCO has a slight reduction in optimization ability when the edge server CPU frequency is incremented due to considering both latency and energy consumption, but overall it is able to reduce the offloading latency.

Finally, the SRA-E-ABCO method in this paper performs a service reliability analysis of the offloaded tasks before task offloading, and combines the obtained elite population with the improved bee colony algorithm for updating, all of which have a better optimization of the offloading strategy with comprehensive consideration of latency and energy consumption, but since some of the tasks are predicted to be processed at the end device before offloading, the increase in the number of edge nodes SRA-E-ABCO also only has a small advantage, but from other angles of analysis, it is necessary to make predictions about whether to offload or not before offloading.

Conclusion

We propose a service reliability analysis method (SRA) and a terminal task offloading method (E-ABCO) for CEE environments. The SRA method combines terminal devices task attributes, and edge node histories to predict offloading and comprehensively analyses the reliability of offloading for terminal tasks. E-ABCO method improves the traditional bee colony algorithm on initialization population, position update formula and fitness formula. It also fuses the reliable offloading vectors analyzed by the SRA model as an elite population, and finally completes the offloading of terminal tasks.

Our current work primarily focuses on analyzing the characteristics of stationary terminal devices to solve task offloading in CEE environments. However, the assumption of static data is limited in the real application scenarios, and acquiring dynamic user attributes, such as mobile trajectories and user preferences, is more challenging compared with getting static device information. Our future work will include the exploration of predicting users' mobile trajectories and leveraging collaboration between edge nodes for task offloading.

Acknowledgements
The work was supported by the National Natural Science Foundation of China under Grant No.62272243.

Authors' contributions
Shun Jiao, Haiyan Wang, and Jian Luo wrote the main manuscript text and related works. All authors reviewed the manuscript.

Funding
The work was supported by the National Natural Science Foundation of China under Grant No.62272243.

Availability of data and materials
No datasets were generated or analysed during the current study.

Declarations

Ethics approval and consent to participate
Not applicable.

Competing interests
The authors declare no competing interests.

Received: 10 January 2024 Accepted: 1 March 2024
Published online: 14 March 2024

References

1. Cisco (2020) Cisco annual internet report. White Paper (2020)
2. Kai C, Zhou H, Yi Y, Huang W (2021) Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability. IEEE Trans Cogn Commun Netw 7:624–634
3. Gao J, Chang R, Yang Z et al (2023) A task offloading algorithm for cloud-edge collaborative system based on Lyapunov optimization. Clust Comput 26(1):337–348
4. Li H, Zheng P, Wang T et al (2022) A multi-objective task offloading based on BBO algorithm under deadline constrain in mobile edge computing. Clust Comput 26:4051–4067
5. Zhang R, Zhou C (2022) Acomputation task offloading scheme based on mobile-cloud and edge computing for WBANS mobilecloud-andedgecomputingforWBANs. IEEE Int. Conf. Commun. (ICC), Seoul South Korea, p 4504–4509
6. Liu S, Yu Y, Lian X et al (2023) Dependent task scheduling and offloading for minimizing deadline violation ratio in mobile edge computing networks. IEEE J Sel Areas Commun 41(2):538–554
7. Ai Z, Zhang W, Li M et al (2023) A smart collaborative framework for dynamic multi-task offloading in IIoT-MEC networks. Peer Peer Netw Appl 16(2):749–764
8. Li X, Xu Z, Fang F, Fan Q, Wang X , Leung VC (2023) Task Offloading for Deep Learning Empowered Automatic Speech Analysis in Mobile Edge-Cloud Computing Networks. IEEE Trans Cloud Comput 11:1985–1998
9. Liu F, Huang J, Wang X (2023) Joint Task Offloading and Resource Allocation for Device Edge-Cloud Collaboration With Subtask Dependencies. IEEE Trans Cloud Compu 11:3027–3039
10. Lv X, Du H, Ye Q (2022) TBTOA: a DAG-based task offloading scheme for mobile edge computing. In Proc. IEEE Int. Conf., 2022, p 4607–4612
11. Ko H, Kim J, Ryoo D, Cha I, Pack S (2023) A Belief-Based Task Offloading Algorithm in Vehicular Edge Computing. IEEE Trans Intell Transp Syst 24:5467–5476
12. Gao M, Shen R, Shi L, Qi W, Li J, Li Y (2023) Task Partitioning and Offloading in DNN-Task Enabled Mobile Edge Computing Networks. IEEE Trans Mob Comput 22:2435–2445
13. Zhang Y, Liang Y, Yin M et al (2021) A review of computation offloading schemes in mobile edge computing. J Comput 44(12):2406–2430
14. Li H, Zheng P, Wang T et al (2023) A multi-objective task offloading based on BBO algorithm under deadline constrain in mobile edge computing. Clust Comput 26(6):4051–4067
15. Gao J, Chang R, Yang Z, Huang Q, Zhao Y, Wu Y (2022) A task offloading algorithm for cloud-edge collaborative system based on Lyapunov optimization. Cluster Comput 26:337–348
16. Tang X, Liu Y, Zeng Z, Veeravalli B (2023) Service Cost Effective and Reliability Aware Job Scheduling Algorithm on Cloud Computing Systems. IEEE Trans Cloud Comput 11:1461–1473
17. Li J, Liang W, Huang M et al (2020) Reliability-aware network service provisioning in mobile edge-cloud networks. IEEE Trans Parallel Distrib Syst 31(7):1545–1558
18. Li J, Liang W, Huang M, Jia X (2019) Providing reliability-aware virtualized network function services for mobile edge computing. In Proc. IEEE 39th Int. Conf. Distrib. Comput. Sys., p 732–741
19. Zhang W, Zeadally S, Zhou H et al (2022) Joint service quality control and resource allocation for service reliability maximization in edge computing. IEEE Trans Commun 71(2):935–948
20. Yu Q, Zhang W, Notaro P et al (2023) HiMFP: hierarchical intelligent memory failure prediction for cloud service reliability. In: 2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), IEEE, p 216–228
21. Luo N, Xiong Y (2021) Platform software reliability for cloud service continuity-challenges and opportunities. In: 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS), IEEE, p 388–393
22. Hu J, Cai L, Pan J (2021) Mesh network reliability analysis for ultra-reliable low-latency services. In: 2021 IEEE 18th International Conference on Mobile Ad Hoc and Smart Systems (MASS), Virtual, p 198–206
23. Jia J, Yang L, Cao J (2021) Reliability-aware dynamic service chain scheduling in 5G networks based on reinforcement learning. In Proc. IEEE Conf. Comput. Commun. (INFOCOM), p. 1-10
24. Fu X, Sun Y, Wang H et al (2023) Task scheduling of cloud computing based on hybrid particle swarm algorithm and genetic algorithm. Clust Comput 26(5):2479–2488
25. Rizvi N, Ramesh D, Wang L et al (2022) A workflow scheduling approach with modified fuzzy adaptive genetic algorithm in IaaS clouds. IEEE Trans Serv Comput 16(2):872–885
26. Senthil Kumar AM, Padmanaban K, Velmurugan AK et al (2023) A novel resource management framework in a cloud computing environment using hybrid cat swarm BAT (HCSBAT) algorithm. Distrib Parallel Databases 41(1–2):53–63
27. Cai L (2022) Decision-making of transportation vehicle routing based on particle swarm optimization algorithm in logistics distribution management. Clust Comput 1–12
28. Chen L, Wu J, Zhang J et al (2020) Dependency-aware computation offloading for mobile edge computing with edge-cloud cooperation. IEEE Trans Cloud Comput 10(4):2451–2468
29. Meng L, Wang Y, Wang H, Tong X, Sun Z, Cai Z (2023) Task offloading optimization mechanism based on deep neural network in edge-cloud environment. J Cloud Comput 12:1–12
30. Thirugnanasambandam K et al (2022) Directed Artificial Bee Colony algorithm with revamped search strategy to solve global numerical optimization problems. Autom Softw Eng 29:1–31

Publisher's Note