scientific reports

OPEN

Efficient workflow scheduling using an improved multi-objective memetic algorithm in cloud-edge-end collaborative framework

Guangzhang Cui[1,2], Wei Zhang[2,3], Weiwei Xu[1] & Hujun Bao[1]✉

With the rapid advancement of large-scale model technologies, AI agent frameworks built on foundation models have become a central focus of artificial-intelligence research. In cloud-edge-end collaborative computing frameworks, efficient workflow scheduling is essential to reducing both server energy consumption and overall makespan. This paper addresses this challenge by proposing an Improved Multi-Objective Memetic Algorithm (IMOMA) that simultaneously optimizes energy consumption and makespan. First, a multi-objective optimization model incorporating task execution constraints and priority constraints is developed, and complexity analysis confirms its NP-hard nature. Second, the IMOMA algorithm enhances population diversity through dynamic opposition-based learning, introduces local search operators tailored for bi-objective optimization, and maintains Pareto optimal solutions via an elite archive. A dynamic selection mechanism based on operator historical performance and an adaptive local search triggering strategy effectively balance global exploration and local exploitation capabilities. Experimental results on 10 standard datasets demonstrate that IMOMA achieves improvements of 93%, 7%, and 19% in hypervolume and 58%, 1%, and 23% in inverted generational distance compared to MOPSO, NSGA-II, and SPEA-II algorithms. Additionally, ablation experiments reveal the influence mechanisms of scheduling strategies, server configurations, and other constraints on optimization objectives, providing an engineering-oriented solution for real-world cloud-edge-end collaborative scenarios.

Keywords Cloud-edge-end collaborative framework, Workflow scheduling, Multi-objective memetic algorithm, Dynamic opposition-based learning, Energy optimization operator, Makespan optimization operator

With the rapid advancement of large-scale model technologies, AI agents based on foundational models are profoundly transforming the production activities and lifestyles of human society. From consumer-grade scenarios such as intelligent customer service and personalized education to professional fields like industrial decision-making and medical diagnosis, AI agent applications exhibit significant characteristics of computational intensity, diverse scenarios, and real-time responsiveness. For example, e-commerce customer-service systems must achieve user-intent understanding and multi-turn dialogue generation within millisecond-level latency, whereas autonomous driving systems require dynamic path planning with centimeter-level positioning accuracy. These applications impose stringent requirements on the spatio-temporal distribution of computing resources[1,2].

While cloud computing provides abundant resources, its high latency renders it unsuitable for latency-sensitive applications[3]. To address this limitation, edge computing has emerged as a decentralized computational model[4]. By leveraging geographically distributed edge servers, tasks can be processed closer to end users, reducing latency and enhancing response times[5]. However, edge servers have limited computational capabilities. When many tasks are scheduled simultaneously or require significant resources, edge servers may struggle to meet application demands.

Since single computational frameworks (e.g., cloud-edge or edge-end) fail under complex scenarios, the cloud-edge-end collaborative framework offers an effective solution. In smart manufacturing scenarios, this framework can offload high-complexity production scheduling tasks to the cloud while deploying real-time

[1]State Key Laboratory of Computer Aided Design and Computer Graphics, Zhejiang University, Hangzhou 310012, China. [2]Image Derivative Inc, Hangzhou 311100, China. [3]Department of automation, Qingdao University, Qingdao 266071, China. ✉email: 12021174@zju.edu.cn

device control tasks at edge nodes. Nevertheless, existing scheduling mechanisms still face the following key challenges:

1. Complex task dependencies[6–9]: Real-world workflows often exhibit intricate dependency relationships. Improper task decomposition can lead to cross-node data transmission delays. For instance, in an AR navigation system, environmental modeling tasks depend on the output results of visual SLAM.
2. Execution constraints[10,11]: Edge servers in practical scenarios are restricted by geographical distribution, privacy protection, and hardware heterogeneity, making them unable to execute all tasks. When edge resources are insufficient to meet task demands while minimizing latency, tasks must be allocated to specific servers. For example, face recognition from home cameras requires feature extraction at regional edge servers to ensure security and low latency, while medical image analysis tasks must be performed at edge nodes compliant with HIPAA standards.

Furthermore, traditional task scheduling algorithms typically focus on optimizing a single objective, like minimizing latency or energy consumption. In practical scenarios, these goals often conflict, requiring trade-offs among multiple objectives. To solve this issue, multi-objective optimization algorithms have gained significant attention in research. Multi-objective optimization algorithms identify optimal trade-offs among multiple goals, satisfying the diverse demands of real-world applications. The Memetic Algorithm (MA)[12], combining evolutionary algorithms with local search strategies, is widely applied in combinatorial optimization, scheduling, and NP-hard problems, greatly enhancing genetic algorithm search efficiency. By incorporating local search mechanisms, MA enhances global exploration and solution quality, enabling generation of high-quality Pareto fronts for complex problems.

To address these challenges, this study proposes an Improved Multi-Objective Memetic Algorithm (IMOMA) to solve the dependency-aware workflow scheduling problem in cloud-edge-end collaborative frameworks. The algorithm enhances population diversity through a dynamic opposition-based learning strategy, designs dual-objective local search operators to optimize energy consumption and makespan, and establishes an operator selection mechanism based on historical performance. Experimental results demonstrate that compared to classical algorithms, IMOMA achieves a 93% improvement in the hypervolume indicator and a 58% optimization in the inverted generational distance, providing an efficient scheduling solution for real-time intelligent applications. The main contributions of this paper are as follows:

1. Model: Proposes a workflow scheduling model for cloud-edge-end collaborative frameworks considering execution location constraints and priority constraints. Through mathematical proof, the NP-hardness of this problem is revealed, and a mixed-integer programming model with 0–1 decision variables is established, laying a theoretical foundation for subsequent algorithm design.
2. Algorithm: Develops an Improved Multi-Objective Memetic Algorithm (IMOMA). Introduces a Dynamic Opposition-based Learning (DOL) strategy that automatically adjusts the search direction according to the evolutionary state of the population, significantly improving global convergence efficiency while maintaining population diversity. Designs two specialized local search operators to deeply optimize the objectives of energy consumption and makespan. Integrates a dynamic operator selection mechanism based on historical performance to effectively balance the algorithm's capabilities in global exploration and local exploitation. Adopts a density estimation-based external archive mechanism to maintain the Pareto solution set and further enhances solution quality and distribution uniformity through an adaptive local search triggering strategy.
3. Experiment: Conducts comprehensive experiments on 10 datasets of varying scales to evaluate the impact of scheduling strategies, server configurations, and replica constraints on optimization objectives. Compared to three classical algorithms (MOPSO[13], NSGA-II[14], SPEA-II[15]), IMOMA demonstrates superior performance in solution quality and efficiency. Additionally, component ablation analysis validates the contributions of each algorithm module to optimizing energy consumption and makespan.

The paper is structured as follows: Sect. 2 reviews related work, Sects. 3–4 define the problem and propose the scheduling model, Sect. 5 describes the algorithm, Sect. 6 presents experimental verification, and Sect. 7 concludes the paper.

Related work

In recent years, cloud-edge-end collaboration has become a prominent research area, gaining widespread attention. Task scheduling, as a major challenge in cloud-edge-end collaboration, has garnered considerable attention and led to promising research outcomes[1–5].

Task scheduling

The goal of task scheduling is to effectively allocate and optimize resources while meeting various constraints, such as energy consumption, task dependencies, and deadline requirements. In recent years, researchers have proposed various methods to address task scheduling in different scenarios. Dai et al.[16] investigated the collaborative task offloading problem in mobile edge computing and end-to-end systems, aiming to reduce latency and avoid network congestion. They proposed a collaborative task offloading framework and a learning-based algorithm to minimize system costs, including task delay and offloading costs. Zhu et al.[17] applied an improved guided population profile whale optimization algorithm (IGOWOA) to address the task offloading problem in mobile edge computing, with the goal of minimizing user energy consumption, task response delay, and the number of deployed cloudlets. You and Tang[18] explored multi-objective task offloading by combining delay, energy consumption, and task execution cost into a particle swarm optimization framework. In vehicular task offloading, Zhou et al.[19] addressed the unique challenges of task offloading in satellite edge computing networks, considering Low Earth Orbit (LEO) satellite mobility and heterogeneous resource constraints. Their Mobility-aware Cooperative Offloading Algorithm (MCO-A) effectively reduced network latency and energy consumption but was computationally expensive when applied to large-scale task sets. In the context of the deep integration of the Internet of Things, fog computing and cloud computing, Panda[20] proposed an EDP-TO task offloading algorithm. The algorithm selects fog nodes with multi-objective functions, takes into account load balance, and performs well in terms of energy consumption, delay and fairness after three steps. The research shows that compared with the FTO algorithm, the EDP-TO algorithm has significantly improved energy consumption, delay and fairness.

As applications become increasingly complex, they are usually composed of multiple interdependent subtasks to form a workflow. The dependencies among tasks imply that the output of some subtasks will serve as the input for other subtasks, which determines the execution order and processing time. Effectively managing these dependencies is a crucial challenge in task scheduling and has a significant impact on system performance.

Al-HabobA et al.[21] decomposed applications into a series of sequential tasks, offloading them to multiple mobile edge computing servers. A Genetic Algorithm(GA) was employed to optimize the offloading process, aiming to minimize task delay and reduce the likelihood of offloading failures. However, their approach primarily focused on sequential tasks and exhibited limited capability in handling complex dependency structures. Liu et al.[22] investigated dependency-based task scheduling in vehicular edge computing. They proposed a multiple applications multiple tasks scheduling (MAMTS) algorithm that prioritizes tasks within multiple Directed Acyclic Graphs (DAGs) to achieve an optimized scheduling strategy. Although effective, the method struggles to address real-time dynamic dependencies in practical scenarios. Maray et al.[23] modeled task dependencies using DAGs and focused on optimizing task scheduling under strict deadline constraints for delay-sensitive tasks. They employed a Markov Decision Process (MDP) to minimize the total completion time. While their approach demonstrated effectiveness in balancing dependencies and deadlines, it faced challenges in scalability and computational cost when applied to large-scale task sets with complex dependency structures.

Solution method

With the rapid increase in the number of tasks on terminal devices, task scheduling has become a crucial challenge in cloud-edge collaborative computing. To address this issue, various heuristic and metaheuristic approaches have been proposed. Rao[24,25] proposed TLBO algorithm with simple and no adjustment parameters and Jaya algorithm without parameters. Tak et al.[26]. solved the task scheduling problem on VM by applying TLBO and Jaya algorithm to cloud computing, and simulated and verified it in five datasets.

Topcuoglu et al.[27] proposed HEFT and CPOP two task scheduling algorithms for heterogeneous processors. By designing a parameterized graph generator, the experimental comparison results show that these two algorithms are better than previous methods in terms of scheduling quality and cost. Kumar et al.[28] proposed a workflow scheduling algorithm based on task granularity, GSS, which optimizes task prioritization by combining B-level path length and task local interaction (the execution-to-communication ratio of precursor/successor tasks). Experiments show that GSS significantly reduces completion time in scientific workflows (e.g. CyberShake, Montage), and improves virtual machine utilization, providing a more efficient solution for task scheduling in cloud computing environments.

Sun et al.[29] employed Ant Colony Optimization (ACO) to optimize delay, energy consumption, and load balancing, and further extended their work to joint optimization in cloud-edge architectures. While effective for vehicular networks, these methods relied on static optimization models, limiting their applicability to dynamic and heterogeneous environments.

Jiang et al.[30] proposed a greedy multi-objective optimization algorithm for the energy-efficient task scheduling problem of edge heterogeneous multiprocessor systems, redesigned the insertion repair and local search operators, and developed a probabilistic mutation to avoid local optima. Zhang et al.[31] proposed an offloading decision method based on evolutionary algorithms, which enhances convergence and population diversity by employing cascading clustering and incremental learning selection mechanisms. Nandi et al.[32] developed a metaheuristic task offloading strategy using Social Cognitive Optimization (SCO) to balance service delay and energy consumption. While effective, their approach simplifies the multi-objective problem into a single weighted treatment, potentially overlooking conflicts between objectives. Salehan et al.[33] proposed an online offloading algorithm to minimize energy consumption and request execution time, but it focused exclusively on edge or cloud offloading, failing to exploit device-edge-cloud collaboration for enhanced resource efficiency. Addressing this limitation, You and Tang[18] applied Particle Swarm Optimization (PSO) in collaborative computing environments, achieving improvements in energy efficiency and latency reduction. Similarly, Guo and Liu[34] formulated a Cloud-MEC collaborative problem and proposed a cooperative computing offloading scheme to minimize mobile device energy consumption. Chakraborty and Mazumdar[35] utilized GA to optimize energy consumption with delay constraints, while Shukla P[36] employed a hybrid Meta-heuristic based Optimized Resource Scheduling Algorithm(HORSA) to reduce manufacturing time and cost while maximizing resource utilization. Yao[37] constructed a cloud workflow scheduling model that minimizes both execution time and cost, and proposed the MOEA/D algorithm based on weight vector adjustment and local search. By introducing an external elite population to guide the variation of subproblems, utilizing an adaptive weight vector adjustment strategy, and employing a three-point quadratic interpolation approximation for local search, the distribution of solutions and convergence speed were optimized. Similarly, Song et al.[38] applied MOEA/D to balance conflicting objectives such as energy and time consumption.

Summary

Task scheduling in cloud-edge-end collaborative computing primarily revolves around task offloading strategies, task dependency management, and multi-objective optimization. This includes collaborative offloading in mobile edge scenarios and modeling dependencies via directed acyclic graphs (DAGs). Existing solutions such as HEFT, CPOP, and GSS algorithms enhance performance through task prioritization and granularity optimization but lack multi-objective coordination capabilities. While methods like PSO and MOEA/D balance energy consumption and latency, they fail to deeply exploit problem characteristics.

Therefore, this paper addresses the dependency-aware task scheduling problem in the cloud-edge-end collaborative framework and models it as a multi-objective optimization problem. Moreover, IMOMA is proposed to efficiently solve this problem. Compared with existing methods, IMOMA incorporates a DOL and local optimization operators based on objective, significantly enhancing the global search capability and solution uniformity in task scheduling. This effectively resolves challenges related to task dependencies and multi-objective conflict optimization. Table 1 summarizes the comparison of existing works in terms of task dependency handling, scheduling constraints, solutions, and application scenarios.

Problem description

Scenario description

In this study, the cloud-edge-end collaborative computing framework consists of cloud servers, edge servers, and terminal devices. As depicted in Fig. 1, the system includes *l* cloud servers, *n* edge servers, and *m* terminal devices.

$CS = \{cs_1, cs_2, \cdots, cs_l\}$ denotes the set of cloud servers; $ES = \{es_1, es_2, \cdots, es_n\}$denotes the set of edge servers; $D = \{dev_1, dev_2, \cdots, dev_m\}$denotes the set of terminal devices. Each terminal device handles several applications;$App = \{A_1, A_2, \cdots, A_h\}$denotes the set of applications, with each A_j comprising multiple interconnected tasks. And the tasks of each application have the same priority, represented by a number. Tasks can run on edge servers, be scheduled to cloud servers, or be restricted to certain edge servers. When the resources on a server can support multiple tasks running simultaneously, the task priority is not considered. However, when resources are insufficient, tasks with higher priority (priority 1 being the highest) will be executed first. Table 2 provides a summary of the symbols utilized in this problem.

Application description

$A_j = \{T_j^1, T_j^2, \cdots, T_j^{p_j}\}, j \in \{1, 2, \cdots h\}$ denotes the task set. When an application needs to be run, it can be split into multiple tasks and assigned to different computing facilities. These tasks have dependencies, represented by the adjacency matrix E_j, meaning the next task can only be executed once the previous task is completed. Each task is confined to execution on one server. Furthermore, diverse resource requirements (such as CPU, GPU, memory, GPU memory, network I/O, etc.) as well as qualified servers capable of performing these tasks need to be considered.

In the adjacency matrix E_j,defining $e_j(a, a) = 1$, $e_j(a, b) = 1$ signifies that predecessor task *a* has transmitted data to the server hosting task *b*. $e_j(a, b) = 0$ implies that predecessor task *a* has not sent data to the server hosting task *b*. The adjacency matrix E_j is defined as:

$$E_j = \begin{bmatrix} e_j(1,1) & e_j(1,2) & \cdots & e_j(1,p_j) \\ e_j(2,1) & e_j(2,2) & \cdots & e_j(2,p_j) \\ \cdots & \cdots & \cdots & \cdots \\ e_j(p_j,1) & e_j(p_j,2) & \cdots & e_j(p_j,p_j) \end{bmatrix}. \tag{1}$$

Work	Scenario	Solution method	Task dependencies	Scheduling constraints	Objective
[16]	Edge-end	LBCMAB	✗	✗	minimize the latency and offloading cost(S)
[17]	Edge-end	IGOWOA	✗	✗	minimize user energy consumption, task response delay, and the number of deployed cloudlets(M)
[18]	Edge-end	PSO	✗	✓	minimize time delay, energy consumption and task execution cost(M)
[19]	Edge-end	MCO-A	✗	✗	minimize the network latency and energy consumption(S)
[20]	Cloud-edge-end	EDP-TO	✗	✓	minimize the latency
[21]	Edge-end	GA	✓	✗	minimize latency and offloading failure probability(S)
[22]	Edge-end	MAMTS	✓	✗	minimize the average completion time of multiple applications(S)
[23]	Edge-end	MDP	✓	✓	minimizes the total completion time(S)
[29]	Cloud-edge-end	ACO	✗	✗	minimize system latency, energy consumption and load balancing rate(M)
[33]	Edge-end	SCO	✓	✗	minimize energy consumption and time consumption(S)
[36]	Edge-end	GA	✓	✗	minimize energy consumption(S)
[37]	Cloud-edge-end	HORSA	✓	✗	minimize makespan, cost and maximize resource utilization (M)
[39]	Edge-end	MOEA/D	✗	✓	minimize energy consumption and makespan (M)
Ours	Cloud-edge-end	IMOMA	✓	✓	minimize energy consumption and makespan(M)

Table 1. Comparison of related work. LBCMAB (Learning-Based Co-Offloading Approach Based on MAB); S: Single-objective optimization; M: Multi-objective optimization

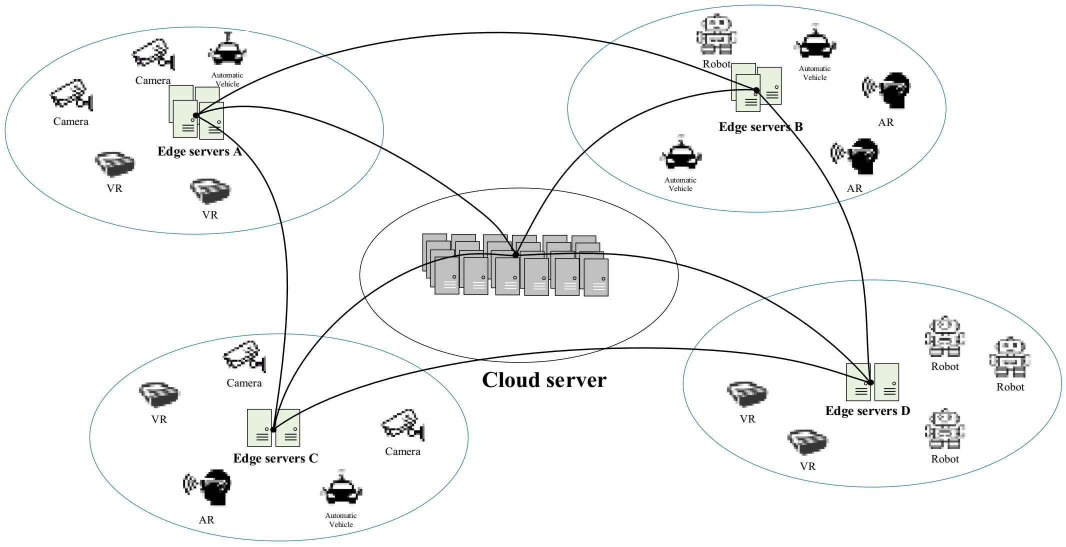

Fig. 1. Cloud-edge-end collaborative computing framework.

Notation	Definition
CS	The set of cloud servers
ES	The set of edge servers
D	The set of terminal devices
App	The set of applications
A_j	The set of *j*-th applications
T_j^i	The *i*-th task of *j*-th applications
E_j	The adjacent matrix of the dependencies of A_j
$R1_j^i$~$R6_j^i$	Resource requirements during task execution
R_x^1~R_x^6	The resource of server *x*
ε	The proportion of additional server resources occupied by task replicas
$\{pre_j^i, sub_j^i\}$	The predecessor and successor task sets of T_j^i
K_j^i	The scheduling strategy of the *i*-th task of *j*-th applications
TD_j^i	The amount of data transmitted to successor tasks
W_j^i	The execution time required for *i*-th task in *j*-th application
$u_x^1 \sim u_x^6$	Represent the utilization of resources of the server *x*
α_i^c	Task *i* is assigned to the *n*-th cloud server
β_i^e	Task *i* is allocated to the *m*-th edge server
$TT_{x,y}^{i,j}$	Represent the time required to transfer data for task T_j^i in application A_j from server *x* to server *y*
TR_x^y	Denote the data transfer rate between servers *x* and *y*
dis_x^y	Represent the transmission distance between *x* and *y*
TS_x^y	Indicate the transmission speed
$ST^{i,j}$	Start time of the T_j^i
$FT^{i,j}$	Finish time of the T_j^i

Table 2. Notation.

Servers utilize corresponding resources while executing a task T_j^i. The six-dimensional array$(R1_j^i, R2_j^i, R3_j^i, R4_j^i, R5_j^i, R6_j^i)$denotes the CPU, GPU, memory, video memory, bandwidth, and storage resource requirements during task execution. The resource of server x attributes are defined as $(R_x^1, R_x^2, R_x^3, R_x^4, R_x^5, R_x^6), x \in \{1, 2, \cdots, l+n\}$.

To ensure the high availability of the application, some tasks may have multiple replicas. These task replicas are deployed on the corresponding cloud servers or edge servers and can quickly start up when a task fails to execute or when a server experiences a fault. Each replica will occupy an additional ε of task resources on the server.

Task description

$\{pre_j^i, sub_j^i\}$ denotes the predecessor and successor task sets of the j-th task in application A_j.pre_j^i refers to the collection of all predecessors for the i-th task T_j^i in application A_j. sub_j^i defines the set of all successors for the j-th task in application A_j.If task A_j is an independent task, then both sets $pre_j^i$and $sub_j^i$are empty sets. Figure 2 shows the dependencies among 8 tasks in an application, with v_start, v_end representing virtual nodes for task initiation and completion.

Scheduling strategy

In various applications, some tasks require specific server scheduling due to factors such as geographical location, privacy, security, and performance demands(e.g., GPU-dependent tasks). To address this, a binary variable K_j^i is defined to represent the scheduling requirements of the i-th task T_j^i in application A_j: when $K_j^i = 1$, the task must be scheduled for execution on the cloud server; when $K_j^i = 1$, the task can be executed on any server; and when $K_j^i = -1$, this task can only be executed on specific edge servers.

$$K_j^i = \begin{cases} 1\ ,\ T_j^i\ \ must\ be\ executed\ on\ the\ cloud \\ 0\ ,\ T_j^i\ \ can\ be\ executed\ on\ any\ server \\ -1\ ,\ T_j^i\ \ must\ be\ executed\ on\ specific\ edge\ server \end{cases}$$

Each task includes the following information: $T_j^i = \{pre_j^i, sub_j^j, R1_j^i \sim R6_j^i, K_j^i, TD_j^i, W_j^i\}$

When task $T_j^i$is completed, data needs to be transmitted to the location of its successor tasks. The amount of data transmitted to successor tasks is denoted as TD_j^i. W_j^i represents the execution time required for i-th task in j-th application.

Mathematical model

In this section, we analyze and formulate the research problem, taking into account task dependencies, resource requirements, and geographical limitations in cloud-edge-end task scheduling. We further verify that the problem is NP-hard.

Decision variables

Scheduling decisions for tasks are expressed using binary variables α_i^c and β_i^e.$\alpha_i^c = 1$ denotes that task i is assigned to the c-th cloud server; otherwise, $\alpha_i^c = 0$.$\beta_i^e = 1$ represents that task i is allocated to the e-th edge server; otherwise, $\beta_i^e = 0$.

Application completion time

$TT_{x,y}^{i,j}$represents the time required to transfer data for task T_j^i in application A_j from server x to server y. TR_x^y denotes the data transfer rate between servers x and y, dis_x^yrepresents the transmission distance between these

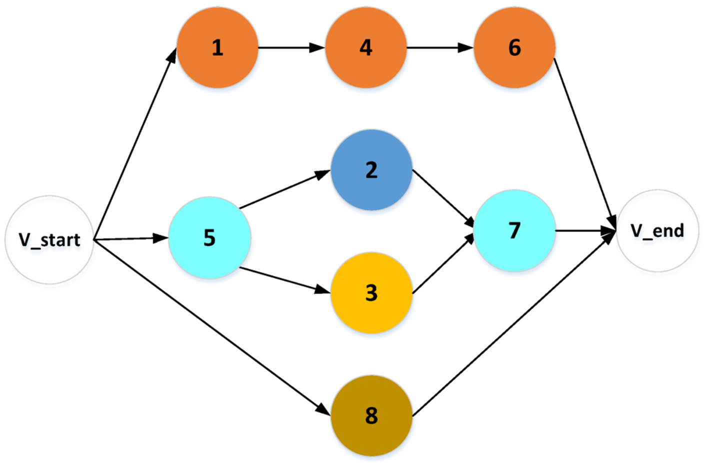

Fig. 2. Dependency-aware task.

servers, and TS_y^x indicates the transmission speed. These parameters apply where $x, y \in \{1, 2, \cdots, l+n, x \neq y\}$. The equation for data transmission time is as follows:

$$TT_{x,y}^{i,j} = \frac{TD_j^i}{TR_x^y} + \frac{dis_x^y}{TS_x^y} \tag{2}$$

Task T_j^i starts execution when its immediate predecessor tasks are completed, and their results are transmitted to the server hosting T_j^i. $ST^{i,j}$ represents the start time of task T_j^i in application A_j.$FT_x^{i,j}$ represents to the finish time of task T_j^i in application A_j on server *x*. For any task, $T_j^{i'}$ belongs to the predecessor task set pre_j^i, where$x, y \in \{1, 2, \cdots, l+n, x \neq y\}$. The start time and finish time equation are as follows:

$$ST^{i,j} = max(FT^{i',j} + TT_{x,y}^{i',j}), \forall i' \in pre_j^i \tag{3}$$

$$FT^{i,j} = ST^{i,j} + W_j^i \tag{4}$$

The makespan of an application CT_j is defined as the maximum end time among all its tasks.

$$CT_j = max\{FT^{i,j}\}, \forall j \in \{1, 2, \cdots h\}\ , i \in \{1, 2, \cdots, p_j\}. \tag{5}$$

Energy consumption model

The power of the server *x* can be divided into idle power $P_x^0$and operating power, where $x \in \{1, 2, \cdots, l+n\}$. Operating power mainly depends on CPU and GPU utilization, assuming that it follows a linear relationship. The power consumption P_x of server *x* is defined in Eq. (6):

$$P_x = P_x^0 + u_x^{cpu} k_{cpu} + u_x^{gpu} k_{gpu} \tag{6}$$

where u_x^1, u_x^2 represent the CPU and GPU utilization, and $k_{cpu}, k_{gpu}$are unit power coefficients.

The energy consumption $EC_x^t$of server *x* during time interval t_i is expressed in Eq. (7), where $I_x = \{t_1, t_2, \cdots, t_q\}$defines the intervals with constant load for server *x*:

$$EC_x^{t_i} = P^x t_i \tag{7}$$

Here, t_i denotes the duration where CPU and GPU load on the server remain constant. The total energy consumption across all servers and time intervals during scheduling is computed as follows:

$$EC_{total} = \sum_{j=1}^{l+n} \sum_{i=1}^{|I_j|} EC_j^{t_i}. \tag{8}$$

Constraint analysis

A valid application scheduling solution must meet the following conditions:

1. Scheduling constraint.

Each task must be executed on a single server, and task *i* must adhere to the scheduling strategy during execution.

$$\begin{cases} \sum_{c=1}^{l} \alpha_i^c = 1, \sum_{e=1}^{n} \beta_i^e = 0, K_j^i = 1 \\ \sum_{c=1}^{l} \alpha_i^c + \sum_{e=1}^{n} \beta_i^e = 1, K_j^i = 0 \qquad \forall i \in \{1, 2, \cdots, p_1, \cdots, \sum_{j=1}^{h} p_j\}, j \in \{1, 2, \cdots, h\}. \\ \sum_{c=1}^{l} \alpha_i^c = 0, \sum_{e=1}^{n} \beta_i^e = 1, K_j^i = -1 \end{cases} \tag{9}$$

2. Dependency constraint.

The goal of task dependency scheduling is to ensure that the start time $ST^{i,j}$ of each task$T_j^i$is no earlier than the completion time$FT_x^{i,j}$of all its predecessor tasks T_j^iplus the transmission time $TT_{x,y}^{i',j}$:

$$ST^{i,j} \geqslant max(FT^{i',j} + \prod e_j(i',i) \cdot \sum_{x,y}^{l+n} (\varphi_{i'}^{x} \cdot \varphi_{i}^{y} \cdot TT_{x,y}^{i',j})), \forall j \in \{1,2,\cdots h\}, i \in \{1,2,\cdots,p_j\}, i' \in pre_j^i. \quad (10)$$

3. Processing resource constraints.

When tasks are executed on the cloud or edge, each task must satisfy the processing resource constraints of its server. The server resource usage limit is defined as:

$$\max(u_x^1, u_x^2, \mathrm{u}_x^3, u_x^4, u_x^5, u_x^6) \leqslant 1, \forall x \in \{1,2,\cdots,l+n\}. \quad (11)$$

When tasks of application *i* are assigned to server ccc, and if the start time of task T_j^i is ST_j^i, the server's resources at that moment should not be less than the requirements of T_j^i.

$$\begin{cases} \sum_{x}^{l+n} \varphi_i^x \mathrm{R1}_j^i \leqslant \left(1-u_x^1\right) R_x^1 \\ \sum_{x}^{l+n} \varphi_i^x \mathrm{R2}_j^i \leqslant \left(1-u_x^2\right) R_x^2 \\ \sum_{x}^{l+n} \varphi_i^x \mathrm{R3}_j^i \leqslant \left(1-u_x^3\right) R_x^3 \\ \sum_{x}^{l+n} \varphi_i^x \mathrm{R4}_j^i \leqslant \left(1-u_x^4\right) R_x^4 \\ \sum_{x}^{l+n} \varphi_i^x \mathrm{R5}_j^i \leqslant \left(1-u_x^5\right) R_x^5 \\ \sum_{x}^{l+n} \varphi_i^x \mathrm{R6}_j^i \leqslant \left(1-u_x^6\right) R_x^6 \end{cases}, \quad \forall x \in \{1,2,\cdots,l+n\}, j \in \{1,2,\cdots h\}, i \in \{1,2,\cdots,p_j\}. \quad (12)$$

Our objective is to identify a feasible scheduling scheme that minimizes server energy consumption and the makespan of all applications. Therefore, the problem addressed in this paper can be formulated as follows:

$$\min\ EC_{total} \quad (13)$$

$$\min\ \max\{CT_j\}. \quad (14)$$

s.t. (9), (10), (11), (12)

$$\varphi_i^x = \begin{cases} \alpha_i^x\ , x \leqslant l \\ \beta_i^{x-l}, x > l \end{cases} \quad (15)$$

$$\alpha_i^c, \beta_i^e, \varphi_i^x \in \{0,1\}. \quad (16)$$

Among them, constraint (15) is an auxiliary variable that indicates whether task *i* is scheduled on server *x*, and (16) defines the domain of the decision variable.

Complexity analysis

In this section, we prove that the problem discussed in this paper is NP-hard.

Lemma 1 *Workflow scheduling in cloud-edge-end collaborative framework is NP-hard.*

We reduce the simplified version of problem A to a Job Shop Scheduling Problem (JSP)[38]with known NP difficulty. The formal definition of problem A and its reduction process to JSP are as follows:

Formal definition of problem A
Parameter:
Task set $T = \{t_1, t_2, t_n\}$
Server set $S = C \cup E = \{s_1, s_2, ..., s_m\}$, where *C* and *E* denote cloud and edge servers, respectively.
Task dependency matrix $D \subseteq T \times T$where$(t_j, t_i) \in D$indicates t_j must complete before t_i

Execution time matrix$E = [e_{i,s}]_{n \times |S|}$, where $e_{i,s}$ is the execution time of task$t_i$on server s.
Data transmission delay matrix $\Delta = [d_{s_1,s_2}]_{|S| \times |S|}$
Objective: minimize the maximum completion time$C_{\max} = \max_{t_i \in T}\{\text{completion time of } t_i\}$
Dependency constraint: If $(t_j, t_i) \in D$, then $start(t_i) \geqslant finish(t_j) + d_{s_j,s_i}$, where s_j and$s_i$are the servers assigned to t_j and t_i, respectively.

Reduction to JSP
JSP instance construction.
Job set: $J = \{J_1, J_2, ..., J_n\}$, where each task $t_i$corresponds to a job J_i.
Operations: Each job J_i has an operation sequence$O_{i,1} \rightarrow O_{i,2} \rightarrow \cdots \rightarrow O_{i,k}$, where *k* is the length of the dependency chain for t_i.
Machine assignment: Operation$O_{i,m}$is assigned to machine *m*, with processing time $p_{i,m} = e_{i,m}$.
Setup time: $\tau_{m_1,m_2} = d_{m_1,m_2}$
Precedence constraints: If $(t_j, t_i) \in D$, the last operation of $J_i$must precede the first operation of J_i.

Solution equivalence
Proposition Problem A has a feasible solution if and only if the corresponding JSP instance has a feasible solution, and their makespans are identical.

Proof **Necessity**: Given a schedule σ for Problem A, construct a JSP schedule σ':

1. Map t_i assigned to server *s* in σ to J_i assigned to machine $m = s$ in σ'.
2. Enforce dependencies D via precedence constraints.
3. Map transmission delays to machine setup times. The makespan of σ'equals that of σ

Sufficiency: Given a JSP schedule σ', construct a schedule $\sigma$for Problem A

1. Map J_i assigned to machine *m* in$\sigma'$to t_i assigned to server$s = m$in σ.
2. Enforce precedence constraints via dependency constraints D. The makespan of σ equals that of σ'.

Thus, the two problems are equivalent in solution space and objective function.

NP-hard conclusion
Since JSP is NP-hard and Problem A is a special case of JSP, Problem A is also NP-hard. The original task scheduling problem in cloud-edge frameworks, which includes additional constraints, inherits this NP-hardness.
This reduction formally establishes the complexity of task scheduling in cloud-edge frameworks, justifying the necessity of metaheuristic algorithms like IMOMA.

Algorithm design
To effectively solve NP-hard problems and provide near-optimal solutions within a reasonable time, we propose IMOMA. The algorithm integrates global search, local search, and an archive mechanism to balance exploration and exploitation. Global search leverages DOL to expand the search space and avoid local optima; local search uses energy and makespan optimization operators to refine solution quality; the archive stores non-dominated solutions and maintains diversity through crowding distance, ensuring rapid convergence to the Pareto Front (PF). IMOMA effectively addresses multi-objective conflicts and generates well-distributed, high-quality solutions.

Encoding and decoding
The representation of solutions affects the search efficiency of metaheuristic algorithms. In this study, a two-layer encoding scheme is designed for IMOMA, as shown in Fig. 3. Specifically, the encoding scheme consists of the following layers: the first layer represents the task sequence; and the second layer indicates the server index assigned to each task. Subsequent operations are applied exclusively to the second-layer encoding.
Figure 3 depicts a scenario where the encoding scheme is applied to a setup with nine tasks and four servers—three edge servers (1–3) and one cloud server (4).

Archive
In multi-objective optimization, an archive stores non-dominated solutions during the optimization process, ensuring comprehensive coverage of the PF. The archive also employs diversity maintenance strategies to improve the uniformity of solution distribution. The method involves the following steps:
Initialization: The archive starts empty, with non-dominated solutions from the initial population being stored.
Iteration process: In each iteration, new non-dominated solutions are added to the archive, which is then updated to preserve diversity.
Capacity control: The archive employs a crowding distance-based pruning mechanism to maintain diversity. Solutions with smaller crowding distances are removed first, as they contribute less to the exploration of the solution space. This approach effectively balances convergence precision and diversity preservation, ensuring a robust search process. Additionally, the archive size is set to a proportion arc of the population size *pop*, providing a trade-off between computational efficiency and solution quality.

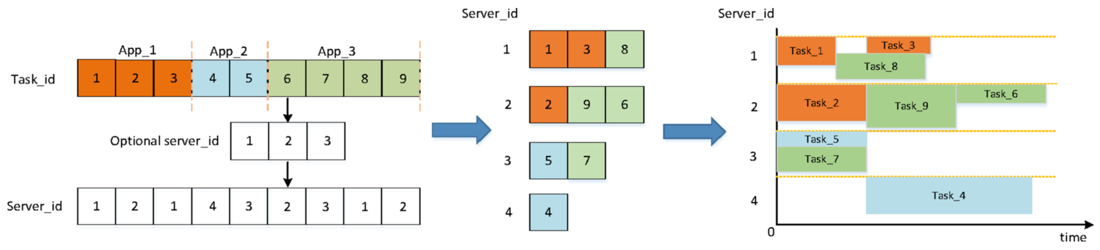

Fig. 3. Example of encoding and decoding.

Global optimization strategies

In IMOMA, global search leverages DOL and crossover to enhance exploration capabilities, ensuring a broader search space and greater solution diversity.

Dynamic opposition-based learning strategy

Traditional Opposition-Based Learning generates opposite solutions at fixed distances, which lacks randomness, leading to insufficient population diversity and an increased risk of being trapped in local optima. In contrast, DOL introduces a dynamic adjustment mechanism during algorithm iterations, adaptively modifying opposite solutions based on the current search phase. This approach not only expands search space but also enhances adaptability, reducing the risk of local optima and increasing the probability of convergence to the global optimum. The dynamic adjustment steps of the DOL search process are as follows:

Generating opposite solutions: For the current population *P*, generate opposite solutions for each individual based on Eq. 17, forming a new population $\tilde{P}$. $\bar{P}$represents the opposite population of *P*.

$$\tilde{P} = P + \omega \cdot \bar{P} \tag{17}$$

$$\omega(g) = \alpha \cdot \sin(\pi \cdot \frac{g}{G}) + \beta \cdot \sin(2\pi \cdot \frac{g}{G}) + \lambda. \tag{18}$$

Combining populations: Merge *P* and $\tilde{P}$ to form a combined population P'.

Non-dominated sorting: Perform fast non-dominated sorting on P'.

Elite retention: Use an elitist strategy to retain top-ranked individuals from P', maintaining a fixed population size for the next generation.

In Eq. 18, α and β are oscillation factors controlling the amplitude of ω during different search stages (early, middle, and late), while λ ensures ω remains non-negative. Here, *g* denotes the current iteration number, and *G* is the total number of iterations.

Crossover

The crossover operation aims to enhance solution diversity and to avoid premature convergence by combining genetic information from selected parents. In this study, a biased uniform crossover strategy is employed to maintain diversity while improving solution quality. The process involves the following steps.

First, a parent individual $x_A = (x_{A_1}, x_{A_2},, \cdots x_{A_j} \cdots, x_{A_n})$ is randomly selected from the archive. Meanwhile, another parent individual $x_B = (x_{B_1}, x_{B_2},, \cdots x_{B_j} \cdots, x_{B_n})$ is randomly selected from the non-pareto front solutions in the current population. Then, a random encoding string $s = (s_1, s_{2,}, \cdots s_j \cdots, s_n)$ of the same length is generated, with each element ranging from 0 to 1. For the corresponding positions in the encoding of x_A and x_B, the following operation is performed: if $s_j > 0.4$, the gene x_{A_j} from $x_A$is retained in the offspring $x_{A'}$; otherwise, the gene x_{B_j} from x_B in the offspring $x_{B'}$ is retained. Repeat this process pop/2 times. pop offspring individuals are generated to form the current population. Figure 4 is a schematic diagram of the offspring generated after the crossover of the given parent individuals [2, 1, 3, 3, 1, 4] and [4, 2, 4, 3, 1, 2].

Local optimization strategy

The core of the memetic algorithm lies in the design of local search strategies, which aim to refine high-quality individuals in the population and enhance solution quality in multi-objective optimization. The key challenges in this process involve addressing three critical questions: which individuals to select for local search, how to exploit the selected individuals, and when to trigger the local search.

Individual selection

High-quality individuals are selected for local optimization using a rank-based probability distribution. This method prioritizes potential solutions while maintaining population diversity. The selection probability p_i for the *i*-th individual is defined as Eq. 19.

$$p_i = \frac{1/r_i}{\sum_{j=1}^{pop}(1/r_j)}. \tag{19}$$

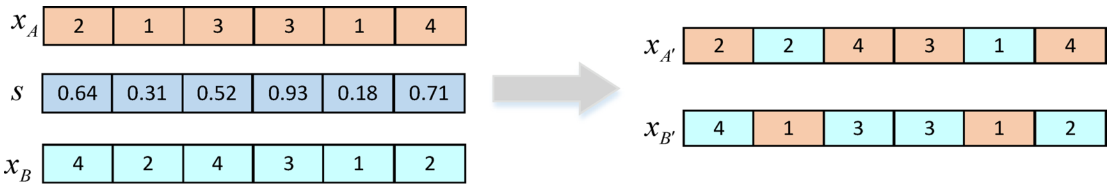

Fig. 4. Example of crossover operator.

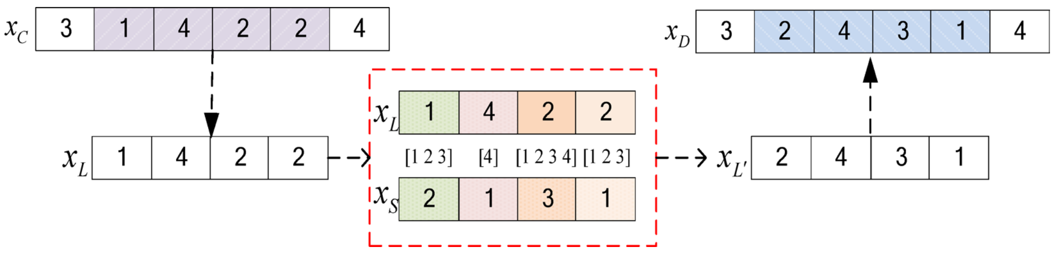

Fig. 5. Illustration of mutation operator application.

pop represents the total number of individuals in the archive; r_i denotes the rank of the *i*-th individual, ensuring that higher-ranked solutions are selected more frequently.

Local search
Mutation: The mutation operator aims to improve local search efficiency, break through local optima, and enhance population diversity. To this end, a partial segment redistribution method is employed, following these steps.

Step 1: Randomly choose a segment x_L of length L.

Step 2: Create a random integer sequence $x_S$to represent the position of the replacement server in the set of available servers.

Step 3: Utilize the elements x_S as an index to replace the corresponding genes in the selected segment x_L, resulting in the adjusted segment $x_{L'}$.

Step 4: Finally return x'_L, yielding a mutated new solution x_D.

For instance, an individual [3,1,4,2,2,4] is mutated into the individual illustrated in Fig. 5.

Energy consumption optimization operator (EO): To reduce server energy consumption, tasks can be consolidated onto a subset of servers, thereby minimizing the total number of active servers and reducing overall energy consumption. The specific approach is as follows: first, randomly select a subset of servers from the active server pool to serve as target servers for task consolidation; second, identify all non-target servers and remove their tasks for rescheduling; third, redistribute the tasks from the non-target servers to the selected target servers, ensuring that all resource constraints are met. As illustrated in Fig. 6, the system initially operates four servers running different tasks. Servers 1, 3, and 4 are designated as target servers. Consequently, all three tasks on Server 2 are removed and rescheduled to Servers 1, 3, and 4, respectively. This process effectively reduces the number of active servers from 4 to 3, thereby minimizing energy consumption.

Makespan optimization operator (MTO): To optimize system performance, particularly by reducing the transmission time between dependent tasks, this paper proposes MTO. The operator aims to schedule dependent tasks on the same server whenever possible, minimizing communication overhead and improving scheduling efficiency. The specific implementation steps are as follows: first, randomly select a subset of applications with dependent tasks as the scheduling targets, and reschedule their tasks; second, assign dependent tasks to the same server whenever feasible, and adjust their execution order to minimize transmission time between tasks. As shown in Fig. 7, Servers 1 and 2 are randomly selected as the operation targets. Task 2 from Server 2 is inserted before Task 3 on Server 1, and the execution order of other tasks is adjusted accordingly to reduce the transmission time between tasks.

Local search trigger strategy
To balance exploration and exploitation while reducing runtime, the local optimization strategy is probabilistically triggered. The trigger probability ρ increases dynamically with the number of iterations, as defined in Eq. (20). When ρ exceeds a random value, the local optimization strategy is executed. This design limits local optimization in the early stages to preserve exploration potential and increases its application in the later stages to accelerate convergence.

$$\rho(g) = \left(\tan\left(\frac{\pi \cdot g}{G}\right)\right)^S. \tag{20}$$

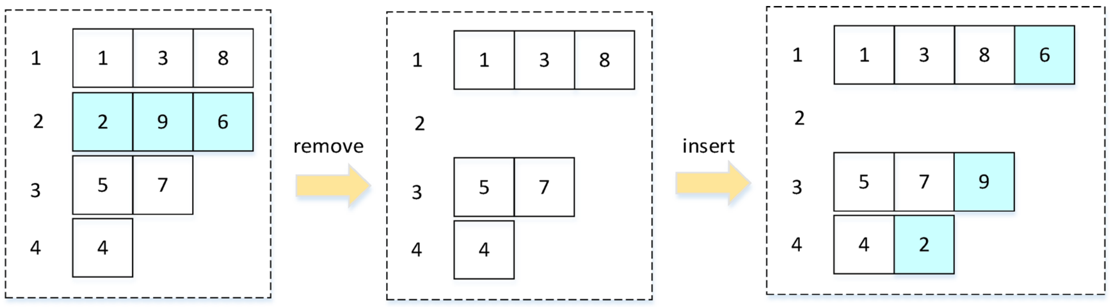

Fig. 6. Illustration of energy consumption optimization operator.

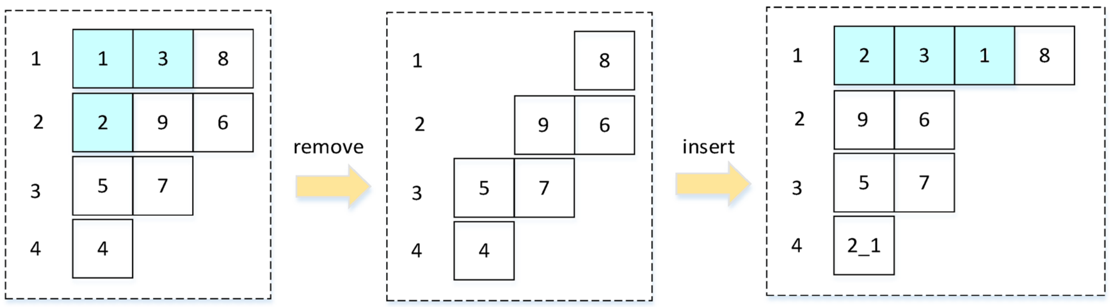

Fig. 7. Illustration of makespan optimization operator.

Here, *S* is the suppression factor, ranging between (0,0.5], regulating the growth rate of ρ. Larger values of *S* result in slower ρ growth.

Adaptive weight adjustment

To enhance search efficiency in global and local searches, an adaptive weight adjustment strategy dynamically adjusts the selection probabilities of operators. Initially, all operators are assigned equal weights, with selection probabilities proportional to their weights. During each iteration, the score of an operator is updated based on the frequency of successful perturbations: the score increases by σ_1 when $|PF|$ increases, by σ_2 ($\sigma_2 < \sigma_1$) when $|PF|$ stays the same, and by σ_3 ($\sigma_3 < \sigma_2$) when $|PF|$ decreases. At the end of each iteration, the weight of an operator w_{j+1} is updated based on its score and usage frequency using Eq. (21):

$$w_{j+1} = w_j \cdot (1 - \eta) + \eta \cdot \frac{\pi_j}{\theta_j}. \tag{21}$$

Here, η is the response factor controlling the influence of the score on weight adjustment, π_j represents the operator's score, and θ_j denotes its usage frequency.

Summary of the algorithm

Building upon the preceding algorithm design, the complete workflow of IMOMA is illustrated in Fig. 8. In addition, the time complexity of IMOMA is $O(G \cdot N^2)$, *N* is the population size.

Numerical experiments

In this section, we designed and conducted a series of experiments to evaluate the performance of IMOMA and analyze the impact of various factors on the problem. First, datasets reflecting real-world scenarios were developed, and energy consumption formulas were constructed based on actual CPU and GPU loads. Subsequently, key parameter configurations for IMOMA were determined through orthogonal experiments. Then, comprehensive evaluations were performed using metrics such as Hypervolume (HV) and Inverted Generational Distance (IGD) to assess the algorithm's convergence performance, solution quality, and distribution uniformity. Statistical reliability of the experimental results was further verified through significance analysis. Finally, the influence of different factors on the optimization objectives was thoroughly investigated.

Data description

In this study, we developed a specialized dataset to simulate dependency-aware task scheduling in cloud-edge-end collaborative environments. The dataset was constructed using real-world data from a company in Hangzhou, combined with extensive practical experience, and is available upon request from the authors.

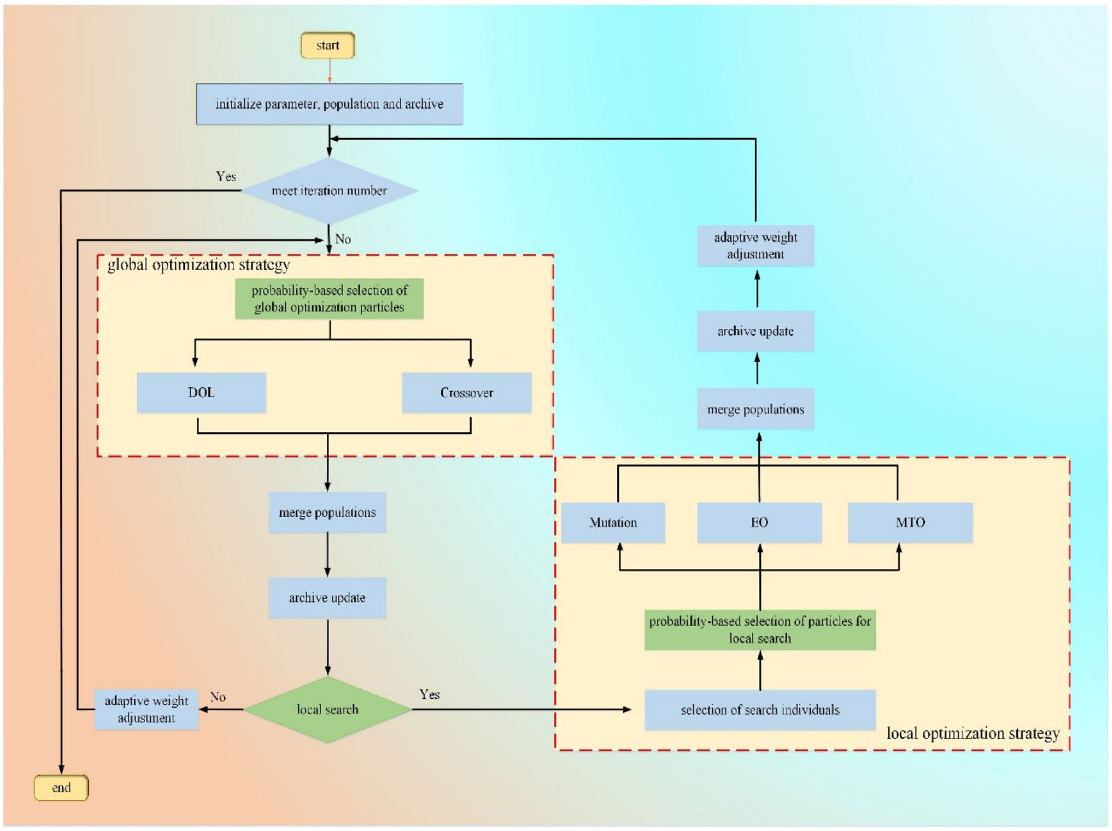
start
initialize parameter, population and archive
meet iteration number
Yes
No
global optimization strategy
probability-based selection of global optimization particles
DOL
Crossover
merge populations
archive update
local search
No
Yes
adaptive weight adjustment
end
adaptive weight adjustment
archive update
merge populations
Mutation
EO
MTO
probability-based selection of particles for local search
selection of search individuals
local optimization strategy

Fig. 8. Algorithm flowchart.

Instance	Number of applications	Number of edge servers	Number of edge servers	Priority range	Range of task numbers	Range of copy numbers
data_50	50	10	2	[1,3]	[1,3]	[1,3]
data_75	75	10	2	[1,3]	[1,3]	[1,3]
data_100	100	10	2	[1,3]	[1,3]	[1,3]
data_200	200	10	2	[1,3]	[1,3]	[1,3]

Table 3. Data information.

Additionally, energy consumption data of servers under varying CPU and GPU loads were collected, based on which the energy consumption formula was derived.

Data information
Existing public datasets fail to fully meet the specific requirements of this study, particularly in terms of application resource demands and task scheduling characteristics. To address this, we constructed four datasets of varying scales based on real-world resource demand data from applications in Hangzhou, combined with extensive practical experience. Among them, datasets with application sizes of 50, 75, and 100 each contain three instances, while the dataset with an application size of 200 contains a single instance. Each dataset consists of a fixed number of edge servers (10) and cloud servers (2). The number of tasks, task priorities, and task replicas for each application are all distributed within the range [1,3]. Detailed information is provided in Table 3, while Table 4 presents additional parameter settings.

Energy consumption formula
In this study, the energy consumption of servers is mainly influenced by the actual utilization rates of CPUs and GPUs. To develop a relevant model, we adopt a method based on Ridge regression and Lasso regression[40] assuming a linear relationship between server energy consumption and the loads of CPUs and GPUs. This model aims to accurately predict the energy consumption of servers under various workloads by collecting actual energy - consumption data at different CPU and GPU utilization levels. The resulting energy consumption formula is presented in Eq. (22), where u_x^{cpu}, u_x^{gpu}denotes CPU load and denotes GPU load.

$$P_x = 248.78 + 3.17u_x^{cpu} + 2.10u_x^{gpu}. \tag{22}$$

Parameter	Value	Parameter	Value
dis_x^y	[10,500]km	λ	2
TR_x^y	1Gbps	η	0.5
TS_x^y	$2 \times 10^8 m/s$	σ_1	33
G	100	σ_2	13
α	0.5	σ_3	9
β	0.5	ε	0.02

Table 4. Other parameters.

	Parameter combination			
	pop	arc	S	RV
1	20	0.3	0.2	0.5932
2	20	0.5	0.5	0.6338
3	20	0.7	0.3	0.6058
4	40	0.3	0.5	0.6979
5	40	0.5	0.3	0.7078
6	40	0.7	0.2	0.6979
7	60	0.3	0.3	0.6802
8	60	0.5	0.2	0.7409
9	60	0.7	0.5	0.7365

Table 5. Orthogonal experiment.

Experimental setup

The experimental environment in this study consists of MATLAB 2022a, Windows 10 operating system, and an AMD Ryzen 7 5800 × 8-core processor. To assess the performance of the multi-objective algorithm, two widely used metrics are employed:

HV: This metric calculates the volume of the hypercube formed by the non-dominated solution set PF and a reference point $ref_{\max}$, which is determined by the maximum objective values of all scheduling schemes. A higher HV value indicates better solution diversity.

IGD: This metric evaluates the convergence and distribution of solutions by calculating the average Euclidean distance of each solution in PF to the true Pareto front. A smaller IGD value indicates better performance.

Both metrics are normalized to account for variations across different instances, with normalization formulas given in Eq. (23). The reference point distance $d_{\max}$ and normalization range are determined based on the endpoints of PF.

$$\mathrm{HV}_{norm} = \frac{\mathrm{HV}\ (\mathrm{PF}, ref_{\max})}{S(ref_{\max}, ref_{\min})}, \quad IGD_{norm} = \frac{IGD(PF)}{d_{\max}} \tag{23}$$

Parameter experiments

Preliminary experiments identified three key parameters of IMOMA: population size (*pop*), archive size (arc), and suppression factor (*S*), where arc is a fraction of the archive size. The optimal combination of these parameters was determined using orthogonal experimental design (L9 (3^3)). Other parameters were configured based on preliminary experimental results and domain expertise. To evaluate the parameter settings, the average HV index was used as the response value (RV), and the algorithm was executed 10 times on multiple datasets to mitigate randomness. For the nine parameter combinations in Table 5, we ran IMOMA independently ten times on each benchmark dataset and recorded the average hypervolume (HV) as the response variable (RV), where a larger value indicates better performance. Table 6 presents the mean HV at each parameter level, along with the range and influence ranking.

As shown in Table 6, the population size (*pop*) has the most significant impact on algorithm performance, contributing to an average 15% improvement in HV compared to other parameter levels. A larger population size enhances global search capability by expanding the search space, thus improving solution diversity. Archive size (arc) ranks second in importance, balancing storage efficiency and solution quality. A moderate archive size avoids excessive computational overhead while ensuring effective solution retention. Finally, the suppression factor (*S*) primarily influences the exploration-exploitation trade-off, where overly small values lead to premature convergence, while excessively large values reduce convergence speed.

Based on the experimental results, the population size was chosen at Level 3, the archive size at Level 2, and the suppression factor at Level 3; therefore, the optimal parameter combination is: $pop = 60$, $arc = 0.5$, and $S = 0.5$.This configuration strikes a balance between search efficiency and solution quality.

level	Parameter		
	pop	arc	S
1	0.6109	0.6571	0.6773
2	0.7012	**0.6942**	0.6646
3	**0.7192**	0.6801	**0.6894**
range	0.1803	0.0371	0.0248
influence ranking	1	2	3

Table 6. Average response values of each parameter level.

Strategy	IMOMA	MOMA_V1	MOMA_V2	MOMA_V3
DOL	√	×	√	√
EO	√	√	×	√
MTO	√	√	√	×

Table 7. Identification of each algorithm and improvement strategies for fusion.

	IMOMA			MOMA_V1			MOMA_V2			MOMA_V3		
	HV	IGD	time(s)	HV	IGD	time(s)	HV	IGD	time(s)	HV	IGD	time(s)
data_50_01	**0.82**	**0.03**	27.39	0.77	**0.03**	27.71	0.77	**0.03**	32.11	0.75	**0.03**	28.06
data_75_03	**0.74**	**0.08**	53.44	0.71	0.09	55.23	0.73	0.1	62.03	0.73	0.11	61.28
data_100_02	**0.81**	**0.06**	68.71	0.77	0.08	67.55	0.79	**0.05**	73.41	0.8	**0.06**	76.60
data_200_01	**0.82**	**0.1**	140.10	0.8	0.15	132.99	0.81	0.12	185.85	0.81	0.11	161.26

Table 8. Comparison of solution results under different operators.

Verification of operator effectiveness

The IMOMA proposed in this study integrates three key strategies: DOL, EO, and MTO. To assess the contribution of each strategy to algorithm performance, a controlled variable method was employed, wherein three algorithm variants were constructed for comparison: MOMA_V1 refers to algorithm 1 excluding DOL, MOMA_V2 refers to algorithm 2 without EO, and MOMA_V3 refers to algorithm 3 excluding MTO, as shown in Table 7. Table 8 presents the average HV, IGD, and runtime results for these three algorithms across four different problem scales, where the best average results are highlighted in bold.

As shown in Table 8, the full version of IMOMA, integrating all three strategies, consistently achieves the best HV and IGD across all scenarios, demonstrating superior performance in both solution diversity and convergence precision. Specifically: **MOMA_V1** performs well for small-scale problems, achieving comparable HV and runtime to IMOMA. However, as the problem size increases, its IGD worsens significantly, suggesting that the absence of DOL reduces solution diversity and convergence precision. **MOMA_V2** achieves reasonable runtime but exhibits poorer HV and IGD compared to IMOMA, indicating that EO plays a critical role in improving solution quality. **MOMA_V3** shows slightly longer runtime and inferior HV compared to IMOMA, highlighting the importance of MTO in balancing efficiency and quality. These results confirm that the combination of DOL, EO, and MTO enhances IMOMA's global search capability and solution efficiency, particularly for large-scale problems. The findings validate the effectiveness of integrating all three strategies to achieve robust and scalable task scheduling solutions.

Comparison with other algorithms

To comprehensively evaluate the relative performance of IMOMA, it is compared with three widely used multi-objective optimization algorithms: MOPSO[13], NSGA-II[14], and SPEA-II[15]. The time complexity of these algorithms is $O(G \cdot N^2)$, ensuring a similar computational cost and fairness in comparisons. The experiments are conducted under two scenarios: equal runtime and equal iterations, aiming to evaluate solution quality and convergence efficiency under fair conditions (with identical parameter configurations and other parameters set according to the respective references). Tables 9 and 10 present the average HV and IGD values for all algorithms, with the best results highlighted in bold. Additionally, Fig. 9 visualizes the Pareto frontiers obtained by the four algorithms for task counts of 50, 75, 100, and 200, providing a clear comparison of solution diversity and convergence performance.

According to Table 9, IMOMA achieves superior HV results across most datasets under the same runtime conditions, demonstrating its ability to generate diverse, high-quality solutions. For larger task counts (e.g., data_200_01), IMOMA shows a clear HV advantage, highlighting its scalability and robustness in handling complex problems. However, IMOMA exhibits slightly higher IGD values than NSGA-II on data_50_01 and

	IMOMA		MOPSO[13]		NSGA-II[14]		SPEA-II[15]	
Data	HV	IGD	HV	IGD	HV	IGD	HV	IGD
data_50_01	**0.87**	0.05	0.50	0.07	0.67	**0.04**	0.71	0.05
data_50_02	**0.80**	0.09	0.45	0.07	0.59	**0.05**	0.67	0.05
data_50_03	**0.68**	**0.08**	0.48	0.18	0.60	0.09	0.61	0.14
data_75_01	**0.86**	**0.09**	0.38	0.17	0.56	0.09	0.61	0.12
data_75_02	**0.87**	**0.10**	0.54	0.35	0.65	0.18	0.66	0.22
data_75_03	**0.88**	**0.10**	0.47	0.55	0.57	0.33	0.64	0.44
data_100_01	**0.93**	**0.24**	0.63	0.76	0.74	0.36	0.78	0.37
data_100_02	**0.88**	**0.14**	0.53	0.68	0.68	0.40	0.72	0.40
data_100_03	**0.86**	**0.06**	0.46	0.21	0.59	0.10	0.64	0.16
data_200_01	**0.86**	0.10	0.45	0.11	0.60	0.10	0.70	**0.06**

Table 9. Experimental results of four algorithms at the same time.

	IMOMA			MOPSO[13]			NSGA-II[14]			SPEA-II[15]		
Data	HV	IGD	time(s)	HV	IGD	Time(s)	HV	IGD	Time(s)	HV	IGD	Time(s)
data_50_01	**0.82**	**0.03**	27.39	0.49	0.06	24.62	0.81	**0.03**	26.63	0.73	0.04	24.33
data_50_02	**0.86**	**0.07**	25.42	0.52	0.12	22.53	0.82	**0.07**	24.82	0.68	0.12	22.75
data_50_03	**0.83**	0.24	28.94	0.38	0.73	26.78	0.63	**0.10**	27.56	0.53	0.18	25.12
data_75_01	**0.85**	**0.04**	51.01	0.46	0.14	47.81	0.83	**0.04**	66.64	0.75	0.09	60.32
data_75_02	**0.74**	**0.13**	50.25	0.41	0.34	46.67	0.70	0.14	64.65	0.64	0.16	58.25
data_75_03	0.74	0.08	53.44	0.40	0.10	50.19	**0.77**	**0.06**	69.48	0.71	0.07	62.73
data_100_01	**0.77**	**0.12**	65.38	0.38	0.12	63.91	0.64	0.18	80.46	0.57	0.26	73.41
data_100_02	**0.81**	**0.06**	68.71	0.45	0.29	65.14	0.80	0.11	86.19	0.75	0.11	75.93
data_100_03	**0.88**	**0.09**	70.35	0.42	0.19	66.85	0.87	0.12	87.62	0.79	0.09	77.01
data_200_01	**0.82**	0.10	140.10	0.29	0.19	127.48	0.72	**0.09**	180.53	0.67	0.12	165.66

Table 10. Experimental results of four algorithms with the same number of iterations.

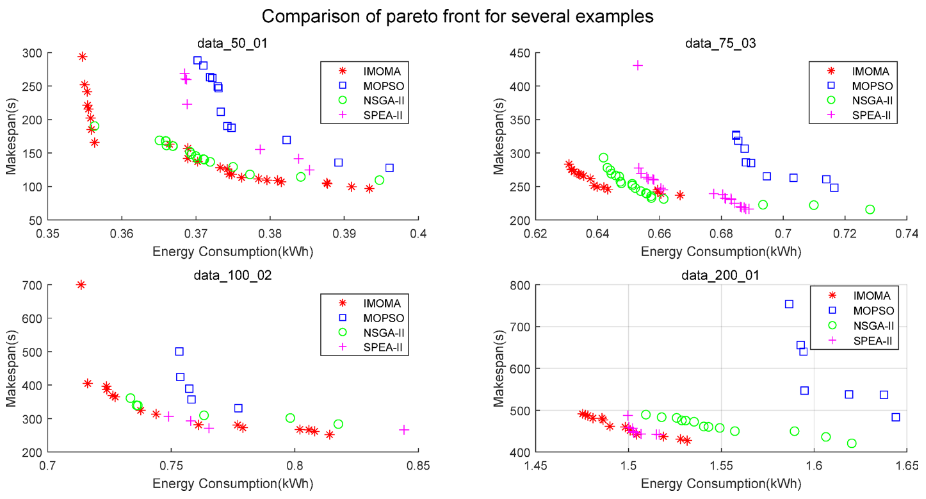

Fig. 9. Comparison of PF for several algorithms on examples.

data_50_03, and larger IGD than SPEA-II on data_200_01, indicating marginal advantages of these algorithms in specific scenarios.

Table 10 shows that, under equal iteration settings, IMOMA consistently outperforms competitors in most cases, delivering better solution distribution and higher convergence precision. On average, IMOMA improves HV by 93%, 7%, and 19% over MOPSO, NSGA-II, and SPEA-II, respectively, and achieves IGD improvements of 58%, 1%, and 23%. The significant improvement over MOPSO highlights the limitations of particle swarm optimization in addressing dependency-aware, discrete optimization problems.

Figure 9 further illustrates that IMOMA consistently produces well-distributed Pareto fronts, providing better coverage of the objective space. These performance advantages stem from three key design features in IMOMA that are absent or underdeveloped in the classical methods:

- MOPSO lacks both a dedicated local search mechanism and an elite archive to retain high-quality solutions. While its lightweight particle-swarm framework incurs low computational overhead, it cannot maintain sufficient population diversity in discrete, dependency-aware scheduling problems, leading to low HV values and uneven front coverage.
- NSGA-II uses a crowding-distance operator to preserve diversity, but its generic crossover and mutation operators are not tailored to the structure of workflow-scheduling solutions. As a result, NSGA-II converges more slowly and yields lower solution quality than IMOMA on our model.
- SPEA-II employs a strength-Pareto archive to filter non-dominated solutions, which helps maintain diversity but at the cost of high computational effort for large-scale instances. Moreover, SPEA-II does not include an adaptive local-search trigger, so under identical runtime it converges less precisely (higher IGD) than IMOMA.

In contrast, IMOMA's dynamic opposition-based learning broadens the global search space; its two objective-specific local search operators fully exploit the underlying problem structure; and its adaptive triggering strategy progressively balances exploration and exploitation. Moreover, the archive mechanism—which preserves and prunes solutions according to crowding distance—ensures that these components collectively enable IMOMA to produce well-distributed, high-quality Pareto fronts for practical multi-objective scheduling.

Analysis of key factors in workflow scheduling

This section analyzes the key factors influencing cloud-edge collaborative task scheduling, focusing on the impact of scheduling strategies, server numbers, and the number of replicas on energy consumption and makespan. Understanding these factors is critical to optimizing scheduling efficiency and achieving a balanced trade-off between energy and performance in real-world applications. Furthermore, the optimal solution on the PF, denoted as $pareto\_best$, is defined as the solution that minimizes the total deviation from the entire PF. This definition ensures that the selected solution represents a balanced compromise between conflicting objectives, providing actionable insights for scheduling decisions.

Validation of scheduling strategies

This experiment explores the effects of three scheduling strategies on the optimization objectives: (1) **Cloud-Edge Collaboration** (tasks are allocated to both cloud and edge servers); (2) **Edge-Only Scheduling** (tasks are assigned exclusively to edge servers); and (3) **Cloud-Only Scheduling** (tasks are allocated solely to cloud servers). The datasets were adjusted accordingly to suit the requirements of these three strategies. Each strategy was run 10 times across problem instances of four different scales, recording the average energy consumption and makespan of the $pareto\_best$. The results are compiled in Table 11 and visualized in Fig. 10.

Table 11; Fig. 10 illustrate that the edge server scheduling strategy generally outperforms the other two strategies in terms of energy consumption but has a significantly higher latency. The cloud server scheduling, on the other hand, exhibits lower latency but higher energy consumption. In contrast, the cloud-edge collaborative scheduling achieves a better balance between energy consumption and latency. Therefore, cloud-edge collaborative scheduling demonstrates strong flexibility in practical applications and can be rationally selected according to actual requirements.

Validation of server quantities

This section investigates the impact of server quantity on optimization objectives, focusing on energy consumption and latency in cloud-edge collaborative task scheduling. Server configuration is a critical factor that influences both system performance and resource allocation efficiency. To analyze this, four server configurations were tested: (1) 13 edge servers and 2 cloud servers; (2) 15 edge servers and 3 cloud servers; (3) 18 edge servers and 3 cloud servers; and (4) 20 edge servers and 4 cloud servers. For each configuration, the algorithm was executed 10 times, and the average energy consumption and makespan were recorded. The results summarized in Table 12 and visualized in Fig. 11.

Table 12; Fig. 11 show that increasing the number of servers effectively reduces both energy consumption and makespan, with both showing a downward trend. Increasing the number of servers helps balance the load, reduce waiting times, and enhance scheduling efficiency. Although energy consumption decreases, when the number of servers reaches a certain level (e.g., the 20_4 configuration), the improvement in energy consumption tends

	Cloud-edge collaboration		**Edge-only**		**Cloud-only**	
Data	**EC**	**MS**	**EC**	**MS**	**EC**	**MS**
data_50_01	0.36	156.36	0.27	337.91	0.44	44.43
data_75_03	0.64	256.21	0.47	514.65	0.87	92.78
data_100_02	0.73	320.18	0.56	616.74	1.06	108.39
data_200_01	1.49	486.90	1.14	996.27	2.06	158.84

Table 11. Different scheduling strategies.

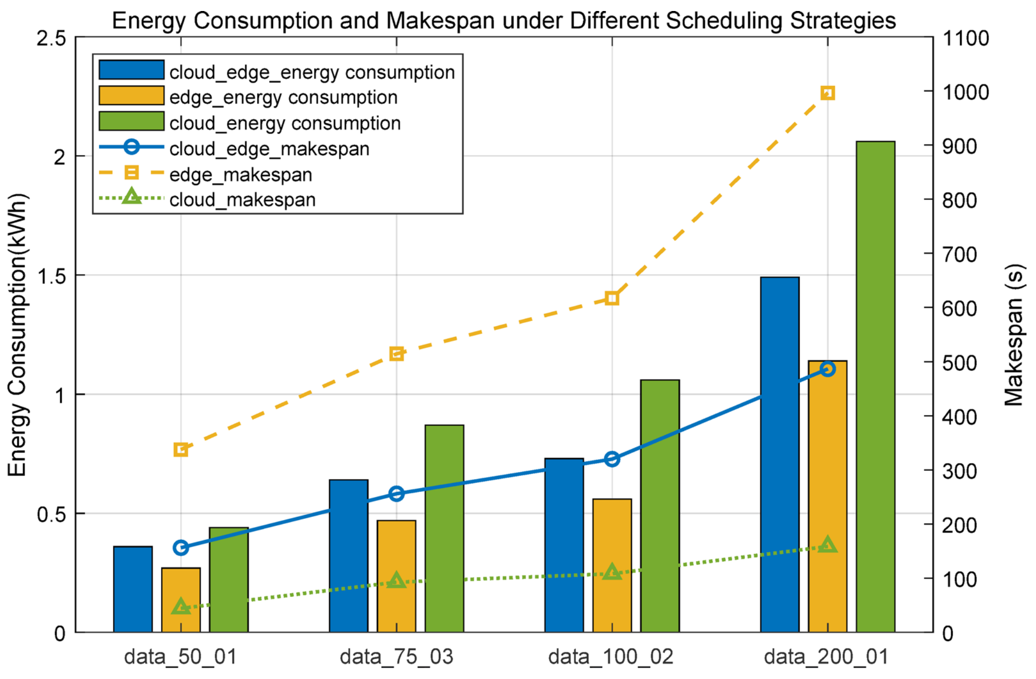

Fig. 10. Energy and time consumption under different scheduling strategies.

	IMOMA_10_2		IMOMA_13_2		IMOMA_15_3		IMOMA_18_3		IMOMA_20_4	
Data	**EC**	**MS**	**EC**	**MS**	**EC**	**MS**	**EC**	**MS**	**EC**	**MS**
data_50_01	0.36	156.36	0.34	140.40	0.32	133.31	0.31	114.58	0.31	130.28
data_75_03	0.64	256.21	0.57	234.34	0.56	228.35	0.54	225.81	0.54	259.41
data_100_02	0.73	320.18	0.70	285.62	0.70	265.16	0.70	258.92	0.70	239.64
data_200_01	1.49	486.90	1.43	441.03	1.42	435.06	1.42	424.14	1.41	430.09

Table 12. Impact of the number of servers on the solution.

to stabilize, and the delay slightly increases. This could be due to the excessive number of servers expanding the search space, thereby increasing scheduling complexity. Overall, moderately increasing the number of servers can effectively reduce both energy consumption and makespan, improving scheduling performance. However, excessive servers should be avoided to prevent efficiency bottlenecks.

Validation of multiple replicas

The number of task replicas has a significant impact on resource allocation efficiency across servers. To study the effects of different replica numbers on optimization objectives, the experiment included the following four ranges: no replicas, [1,2], [1,4], and [1,5]. The datasets were adjusted to meet the experimental requirements and align with the algorithm's constraints. The algorithm was executed 10 times for each configuration, and the average energy consumption and makespan were recorded. The results are presented in Table 13 and visually displayed in Fig. 12.

Table 13 indicates that increasing the number of replicas significantly affects makespan, primarily due to increased resource redundancy, reduced available server capacity, and prolonged task waiting times. In contrast, the impact of replicas on energy consumption is less noticeable, with only minor fluctuations observed across different configurations. While increasing the number of replicas enhances task fault tolerance by ensuring higher reliability, it also results in significant makespan and slightly higher energy consumption. These findings highlight the importance of carefully balancing the number of replicas in practical applications to optimize scheduling performance while meeting system requirements for both reliability and efficiency.

Conclusion

This paper proposes the IMOMA algorithm to address the dependency-aware task scheduling problem under the cloud-edge-end collaboration framework. The algorithm integrates the DOL mechanism to enhance global search capability, utilizes the external archiving mechanism to accelerate convergence, and introduces local optimization operators, effectively improving the performance of task scheduling in terms of energy consumption and makespan. Experiments show that the proposed multi-objective memetic algorithm demonstrates remarkable advantages in tackling task scheduling challenges in the cloud-edge collaborative

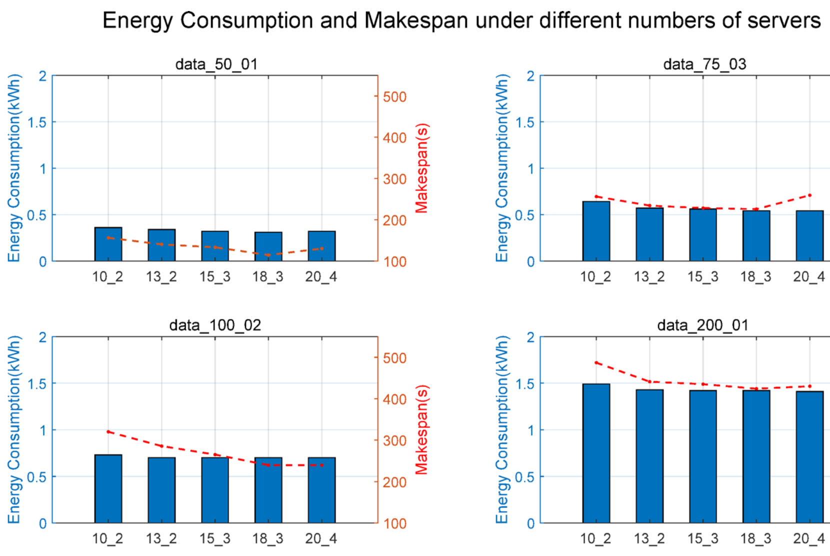

Fig. 11. Energy consumption and completion time under different server quantities.

	[1,1]		[1,2]		[1,3]		[1,4]		[1,5]	
	EC	MS	EC	MS	EC	MS	EC	MS	EC	MS
data_50_01	0.36	156.06	0.36	156.33	0.36	156.36	0.37	163.09	0.38	175.11
data_75_03	0.63	245.65	0.64	248.34	0.64	256.21	0.65	268.81	0.66	270.83
data_100_02	0.73	306.24	0.72	313.66	0.73	320.18	0.74	331.03	0.76	349.90
data_200_01	1.43	466.40	1.45	482.28	1.49	486.90	1.57	497.05	1.63	505.14

Table 13. Impact of the number of replicas on the solution.

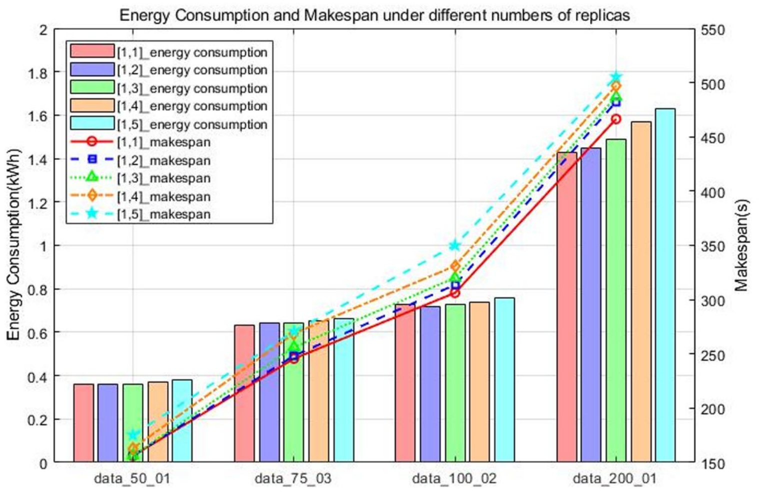

Fig. 12. Energy and makespan under different numbers of replicas.

environment. Compared with MOPSO, NSGA-II, and SPEA-II, its HV index improves by 93%, 7%, and 19% respectively, and the IGD index improves by 58%, 1%, and 23% respectively.

Compared with scheduling modes relying solely on edge computing or cloud computing, cloud-edge collaborative scheduling has more advantages in balancing energy consumption and delay. Future research will focus on the following directions: First, centering on the flexibility optimization of multi-copy task scheduling, constructing a dynamic resource allocation model, and designing an adaptive copy management strategy combined with the reinforcement learning framework to achieve fine-grained control of resource consumption while ensuring task reliability. Second, expanding the applicability verification of the algorithm in complex scenarios, covering typical vertical fields such as industrial Internet of Things, intelligent transportation systems, and Agent applications, and verifying the algorithm's universality in heterogeneous environments through cross-domain experiments. Third, combining the characteristics of server load fluctuations, constructing a multi-source heterogeneous resource prediction model integrating spatio-temporal features, and achieving dynamic perception and pre-judgment of computing resources.

Data availability

The datasets used and/or analyzed during the current study available from the corresponding author on reason able request.

Received: 3 January 2025; Accepted: 23 June 2025
Published online: 13 August 2025

References

1. Gasmi, K. et al. A survey on computation offloading and service placement in fog computing-based IoT. *J. Supercomput*. **78**, 1983–2014 (2022).
2. Jin, X. M. et al. A survey of research on computation offloading in mobile cloud computing. *WIREL. NETW.* **28**, 1563–1585 (2022).
3. Mao, Y. et al. A survey on mobile edge computing: the communication perspective. *IEEE Commun. Surv. Tutor.* **19**, 2322–2358 (2017).
4. Islam, A. et al. A survey on task offloading in multi-access edge computing. *J. Syst Archit*. **118**, 102225 (2021).
5. Shi, W. et al. Edge computing: vision and challenges. *IEEE Internet Things J.* **3**, 637–646 (2016).
6. Xu, J., Chen, L. & Zhou, P. Joint service caching and task offloading for mobile edge computing in dense networks. In *IEEE INFOCOM 2018–IEEE Conf. Comput. Commun*, 207–215 (2018).
7. Tang, T., Li, C. & Liu, F. Collaborative cloud-edge-end task offloading with task dependency based on deep reinforcement learning. *Comput. Commun.* **209**, 78–90 (2023).
8. Chen, L. et al. Collaborative service placement for edge computing in dense small cell networks. *IEEE Trans. Mob. Comput.* **20**, 377–390 (2019).
9. Kai, C. et al. Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability. *IEEE Trans. Cogn. Commun. Netw.* **7**, 624–634 (2020).
10. Zhang, L. et al. Min-max worst-case design for computation offloading in multi-user MEC system. In *IEEE INFOCOM 2020–IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 1075–1080 (2020).
11. Li, K. Scheduling independent tasks on multiple cloud-assisted edge servers with energy constraint. *J. Parallel Distrib. Comput.* **184**, 104781 (2024).
12. Moscato, P. & Cotta, C. A gentle introduction to memetic algorithms. In *Handbook of Metaheuristics*, 105–144 (Springer US,2003).
13. Coello, C. A. C. & Lechuga, M. S. MOPSO: A proposal for multiple objective particle swarm optimization. In *Proceedings of the 2002 Congress on Evolutionary Computation* (CEC'02) vol. 2, 1051–1056 (IEEE, 2002).
14. Deb, K. et al. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol Comput*. **6**, 182–197 (2002).
15. Zitzler, E. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. *Inst. Für Techn. Inf. und Kommunkationary (TIK)-report* vol. 103, 1–21 (2001).
16. Dai, X. et al. Task co-offloading for D2D-assisted mobile edge computing in industrial internet of things. *IEEE Trans. Industr Inf.* **19**, 480–490 (2022).
17. Zhu, X. & Zhou, M. C. Multi-objective optimized cloudlet deployment and task offloading for mobile-edge computing. *IEEE Internet Things J.* **8**, 15582–15595 (2021).
18. You, Q. & Tang, B. Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things. *J. Cloud Comput*. **10**, 41 (2021).
19. Zhou, J. et al. Mobility-aware computation offloading in satellite edge computing networks. *IEEE Trans. Mob. Comput.* **23**, 9135–9149 (2024).
20. Panda, S. K. et al. An energy, delay and priority-aware task offloading algorithm for fog computing incorporating load balancing. *J. Supercomput*. **81** (1), 1–24 (2025).
21. Al-Habob, A. A., Dobre, O. A. & Armada, A. G. Sequential task scheduling for mobile edge computing using genetic algorithm. In *2019 IEEE Globecom Workshops (GC Wkshps),*, 1–6 (IEEE, 2019).
22. Liu, Y. et al. Dependency-aware task scheduling in vehicular edge computing. *IEEE Internet Things J.* **7**, 4961–4971 (2020).
23. Maray, M. et al. Dependent task offloading with deadline-aware scheduling in mobile edge networks. *Internet Things*. **23**, 100868 (2023).
24. Rao, R. Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *Int. J. Ind. Eng. Comput.* **7** (1), 19–34 (2016).
25. Rao, R. V., Savsani, V. J. & Vakharia, D. P. Teaching–learning-based optimization: an optimization method for continuous non-linear large scale problems. *Inf. Sci.* **183** (1), 1–15 (2012).
26. Tak M, Joshi A, Panda S K. Cloud task scheduling algorithms using teaching-learning-based optimization and Jaya algorithm[C]//Proceedings of the 2022 Fourteenth International Conference on Contemporary Computing. 106–113 (2022).
27. Topcuoglu, H., Hariri, S. & Wu, M. Y. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.* **13** (3), 260–274 (2002).
28. Kumar, M. S. et al. Granularity-based workflow scheduling algorithm for cloud computing. *J. Supercomput*. **73**, 5440–5464 (2017).
29. Sun, Y. et al. Vehicular task offloading and job scheduling method based on cloud-edge computing. *IEEE Trans. Intell. Transp. Syst.* **24**, 14651–14662 (2023).
30. Jiang, Q. et al. Metsm: multiobjective energy-efficient task scheduling model for an edge heterogeneous multiprocessor system. *Future Gener Comput. Syst.* **152**, 207–223 (2024).
31. Zhang, J., Gong, B., Waqas, M., Tu, S., & Han, Z. A hybrid many-objective optimization algorithm for task offloading and resourceallocation in multi-server mobile edge computing networks. *IEEE Trans. Serv. Comput.*, **16**, 3101–3114 (2023).
32. Nandi, P. K. et al. Task offloading to edge cloud balancing utility and cost for energy harvesting internet of things. *J. Netw Comput. Appl.* **221**, 103766 (2024).
33. Salehan, A., Deldari, H. & Abrishami, S. An online context-aware mechanism for computation offloading in ubiquitous and mobile cloud environments. *J. Supercomput*. **75**, 3769–3809 (2019).
34. Guo, H. & Liu, J. Collaborative computation offloading for multiaccess edge computing over fiber–wireless networks. *IEEE Trans. Veh. Technol.* **67**, 4514–4526 (2018).
35. Chakraborty, S. & Mazumdar, K. Sustainable task offloading decision using genetic algorithm in sensor mobile edge computing. *J. King Saud Univ. Comput. Inf. Sci.* **34**, 1552–1568 (2022).

36. Shukla, P. & Pandey, S. MOTORS: multi-objective task offloading and resource scheduling algorithm for heterogeneous fog-cloud computing scenario. *J. Supercomput*. **80**, 22315–22361 (2024).
37. Yao, X. A multi-objective cloud workflow scheduling optimization based on evolutionary multi-objective algorithm with decomposition. *Res. Square* (2021).
38. Song, F. H. et al. A multi-objective computation offloading algorithm for mobile-edge computing. *IEEE Internet Things J.* **7**, 8780–8799 (2020).
39. Dauzère-Pérès, S. et al. The flexible job shop scheduling problem: A review. *Eur. J. Oper. Res.* **314**, 409–432 (2024).
40. Bedoui, A. & Lazar, N. A. Bayesian empirical likelihood for ridge and Lasso regressions. *Comput. Stat. Data Anal.* **145**, 106917 (2020).

Acknowledgements
This work was supported by the Zhejiang Provincial Science and Technology Program (No. 2023C01042).

Author contributions
G. C. and W. Z. established the system model and designed the scheduling algorithm. G. C. designed the experiments. W. Z. and W. X. conducted the experiments. G. C., W. Z., and H. B. analyzed the results. All authors reviewed the manuscript.

Declarations

Competing interests
The authors declare no competing interests.

Additional information
Correspondence and requests for materials should be addressed to H.B.

Reprints and permissions information is available at www.nature.com/reprints.