Journal of Engineering, Project, and Production Management
2026, 16(4), 2025-279

Application of Cloud-Fog Computing on Task Scheduling Using ACO Algorithms and Slack Priority

Lun A[1], Huchun Qi[2], and Xiaohai Guo[3]

[1] Associate Professor, Department of Information Engineering, Inner Mongolia Vocational College of Chemical Engineering, Hohhot, 010070, China, E-mail: alun1110@163.com (corresponding author).

[2] Associate Professor, Department of Information Engineering, Inner Mongolia Vocational College of Chemical Engineering, Hohhot, 010070, China

[3] Associate Professor, Network Security and Information Service Center, Inner Mongolia Vocational College of Chemical Engineering, Hohhot, 010070, China

Project Management
Received November 11, 2025; revised May 6, 2026; accepted May 13, 2026
Available online May 29, 2026

Abstract: To achieve efficient task scheduling under cloud-fog computing, this study constructs a scheduling model that considers slack priority and Directed Acyclic Graphs (DAGs). Specifically, tasks are modeled as DAGs, and the slack is calculated by determining core time parameters, which are then adjusted to obtain an ordered task sequence. To solve this ordered task sequence, a hybrid algorithm combining an improved particle swarm optimization algorithm and an ant colony optimization algorithm is designed. The results show that the standard deviations of the hybrid algorithm under the $f1$, $f10$, and $f19$ test functions are significantly smaller than those of the comparison algorithms. In a case study of intelligent urban monitoring scenarios, the designed scheduling model achieves a maximum fog resource utilization rate of 93.58% an average cloud resource utilization rate of 86.54%. It is evident that the designed hybrid algorithm exhibits outstanding stability and optimization capabilities, and the scheduling model can provide an effective solution for cloud-fog computing task scheduling.

Keywords: Ant colony optimization, cloud-fog computing, DAG, particle swarm optimization, relaxation priority, slack priority, task scheduling.

DOI 10.32738/JEPPM-2025-279

1. Introduction

At present, with the swift advancement of digital technology, the number of Internet of Things (IoT) devices is growing exponentially (Bansal and Aggarwal, 2024a). According to Gartner's 2024 edge computing Technical Report, the acceptable delay of delay sensitive applications of the Internet of Things is ≤ 50ms, while traditional cloud computing, due to the "terminal cloud" long-distance transmission, the delay is generally 80-200ms, unable to meet the demand (Mentzer et al., 2024; Zhang et al., 2025). Edge computing (the core technology of fog computing) can shorten the transmission distance to 1-10km, and reduce the delay by 40% -60%, but fog computing alone cannot carry high complexity tasks, so cloud and fog fusion architecture has become an inevitable choice and also highlights the importance of efficient scheduling (Li et al., 2024). However, there are two core issues with current cloud computing Task Scheduling (TS): firstly, insufficient handling of task dependencies. Most existing methods do not fully characterize the complex dependencies between tasks, which can lead to long waiting times for critical path tasks. The second issue is that algorithms are prone to getting stuck in local optima, meaning that fixed parameter settings in multi-modal or composite scenarios can be limited to regional solutions. Meanwhile, TS varies in different scenarios. For example, in the Industry 4.0 scenario, tasks are focused on "device collaborative control", requiring latency ≤ 10ms, reliability ≥ 99.9%, and task dependence on dynamic changes. In the context of smart healthcare, tasks such as "real-time vital sign monitoring" and "medical image analysis" need to balance latency and resource utilization. Therefore, conducting TS under Cloud-Fog Computing (CFC) is of great importance. Currently, commonly used methods to address this issue include rule-based scheduling methods, greedy algorithms, Ant Colony Optimization (ACO) algorithms, genetic algorithms, and others (Rao and Qin, 2024a).

Ahmadabadi et al. (2024) proposed a new Multi-Objective (MO) function capable of jointly minimizing three types of objectives for the TS problem in IoT fog-cloud systems. They defined a new operator called "star-quake" and incorporated it into an MO gravitational search algorithm. The results showed that this method reduced Energy Consumption (EC) by

22%. Ali et al. (2024), aiming to achieve reliable, latency-efficient, and energy-saving communication in the complex environment of Cloud-Fog Computing (CFC) paradigms, proposed a CFC network Task Scheduling (TS) algorithm based on MO Harris Hawks Optimization. The results indicated that this algorithm improved optimization of transmission latency and Energy Consumption (EC) by 25% compared to similar scheduling algorithms. Mousavi et al. (2023) aimed to achieve high-quality services and system performance while accounting for device EC in fog computing architectures, and they designed a constrained bi-objective optimization problem that simultaneously minimized server EC and overall response time. The results showed that this algorithm effectively met the scheduling requirements of latency-sensitive IoT applications. Daghayeghi and Nickray (2024) address the issues of limited computing power in IoT devices, high latency, and power consumption in cloud processing, and proposed a TS model under the fog-cloud paradigm. They also formulated TS as an MO optimization problem. Their results demonstrate that this solution outperformed other benchmark algorithms in terms of service response time and EC.

However, existing methods have four significant limitations: rule-based methods (such as fixed priority) lack elasticity and cannot adapt to dynamic tasks and resource states. Meta-heuristic algorithms have poor convergence and are prone to falling into local optima. The coupling degree of the hybrid strategy is low, and there is no parameter scheduling feedback loop. Most methods lack dynamic adaptability and fail to account for changes in task dependencies and network states. To achieve TS under CFC, this study considers task slack priorities, employs Directed Acyclic Graphs (DAGs), and constructs a corresponding scheduling model. Additionally, a solution algorithm based on the Improved Particle Swarm Optimization (IPSO) algorithm and ACO (ACO-IPSO) is designed. The study aims to address issues such as inaccurate quantification of task priorities and the susceptibility of algorithm parameters to local optima in existing methods, while providing reliable solutions for TS across different scenarios. The distinction between the research objectives and existing methods lies in three key innovations: (1) Task Dependency Modeling: The study constructs a DAG-based task dependency model that integrates relaxation priority mechanisms to enable dynamic priority quantification. This addresses the limitation of traditional methods, which often suffer from vague characterizations of dependencies. (2) Hybrid Algorithm Design: The study proposes an ACO-IPSO hybrid algorithm that dynamically optimizes ACO's core parameters through an IPSO framework. This mitigates the issues of parameter sensitivity and local optima inherent in conventional algorithms. (3) Heuristic Factor Enhancement: By introducing a logarithmic transformation to regulate extreme values in ACO's heuristic factors, the study improves the algorithm's exploration capability in multi-modal and composite scenarios, thereby preventing premature convergence caused by overly dominant heuristic guidance.

2. Research Design

To achieve TS under CFC, the study employs DAGs, considers task slack priorities and constructs a corresponding scheduling model. To solve the task sequence, the study utilizes the ACO algorithm.

2.1. Construction of a CFC TS Model Considering Slack Priority and DAG

To schedule tasks under CFC, ensuring timely responses to demands while reducing EC, the study begins with tasks with priority constraints. These tasks are modeled as DAGs, and task attributes are defined to clarify the dependency logic and execution constraints among tasks. Subsequently, the study calculates the slack for each task to determine its priority, thereby highlighting tasks with the tightest deadlines or on the most critical paths. Finally, based on the slack-based priorities, the study transforms the initial DAG into an ordered task sequence to compress the solution space. The hierarchical model diagram of cloud and fog architecture resources is shown in Fig. 1.

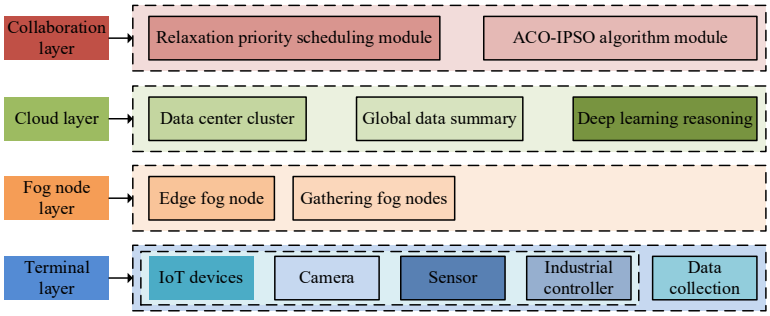


Fig. 1. Cloud and fog architecture resource layered model diagram

As shown in Fig. 1, the cloud architecture resource layering model diagram includes the terminal layer, fog node layer, cloud layer, and collaborative layer. The terminal layer contains IoT devices and is responsible for generating tasks. Fog nodes are divided into edge nodes and aggregation fog nodes to perform tasks with low to medium complexity. The cloud layer is composed of a cluster of data centers that handle high-complexity tasks. The collaboration layer realizes cross-layer resource collaboration. The task allocation process under CFC is illustrated in Fig. 2.

In Fig. 2, task allocation under cloud computing presents a three-level task allocation logic of "terminal (generating DAG tasks) -fog node (low latency execution/forwarding) -cloud (high complexity tasks)", and the task allocation process involves terminal devices, task information, allocation strategies, fog nodes, scheduling algorithms, and cloud servers. This process clearly depicts the complete flow path of tasks generated from the terminal, matched with resources through scheduling strategies and algorithms, and processed or forwarded to the cloud for computation at nearby fog nodes. It

intuitively reflects the data transmission, task unloading, and collaborative execution mechanism between the terminal layer, fog node layer, and cloud layer. The EC A_c of the cloud server is expressed as shown in Eq. (1) (Rao and Qin, 2024b).

$$A_c = B_d \times G + \left(B_{\max} - B_d\right) \times \int_0^G F_c(g)^E \, dt \tag{1}$$

In Eq. (1), B_d represents the idle power of the cloud server, G denotes the total task execution duration, G is the full-load power, $F_c(g)$ indicates the utilization rate of the Central Processing Unit (CPU) of the cloud server at time g, and E represents the power consumption coefficient. The EC $A_c = B_d \times G + \left(B_{\max} - B_d\right) \times \int_0^G F_c(g)^E \, dt$ of the fog node is expressed as shown in Eq. (2) (Shukla and Pandey, 2024).

$$A_f = H \times \left[B_k \times G + \left(B'_{\max} - B_k\right) \times \int_0^G F_f(g) dt \right] \tag{2}$$

In Eq. (2), H represents the number of fog nodes involved in scheduling, B_k denotes the idle power of a single fog node, B_k signifies the full-load power of a single fog node, and $F_f(g)$ indicates the CPU utilization of fog nodes at moment g. An example of a DAG is shown in Fig. 3 (Goel and Tiwari, 2024; Ahmed et al., 2023).

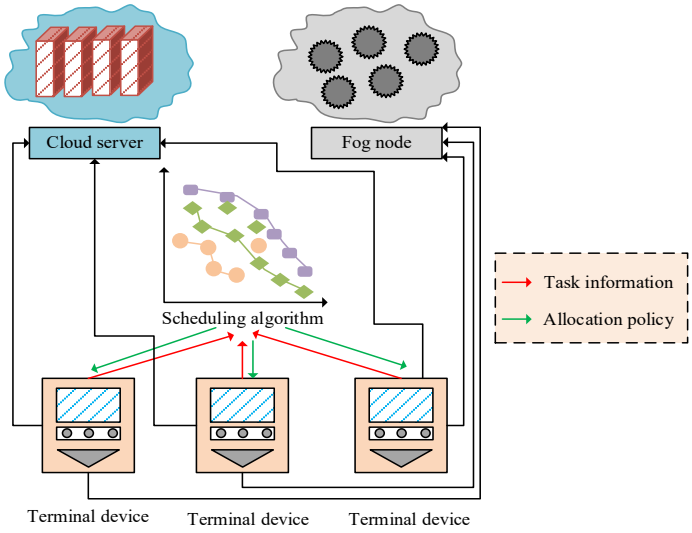


Fig. 2. Task allocation process in CFC

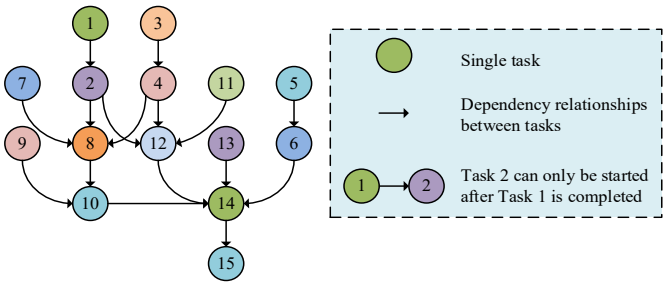


Fig. 3. Example of DAG

In Fig. 3, the core structure of DAG includes vertices, directed edges, hierarchical relationships, and parallel dependency chains. Based on the operational logic of DAGs, it is necessary to calculate four core time parameters for tasks, namely the earliest start time $U(i)$, the earliest finish time $V(i)$, the latest finish time $X(i)$, and the latest start time $Y(i)$, which provide a foundation for subsequent calculation of slack. Among them, the expression for $U(i)$ is presented in Eq. (3).

$$U(i) = \begin{cases} 0 & , Pred(i) = \varnothing \\ \max\{V(j) | \ j \in Pred(i)\} & , Pred(i) \neq \varnothing \end{cases} \tag{3}$$

In Eq. (3), both i and j represent tasks, with i being the main task and j being the "predecessor task" associated

with task i. $Pred(i)$ denotes the set of all direct predecessor tasks of task i, that is, tasks that have directed edges pointing to task i. $V(j)$ represents the earliest finish time of task j. The expression for $V(i)$ is shown in Eq. (4).

$$V(i) = U(i) + G(i) \tag{4}$$

In Eq. (4), $G(i)$ represents the execution time of task i. The expression for $X(i)$ is shown in Eq. (5).

$$X(i) = \begin{cases} K_{all} & , Succ(i) = \varnothing \\ \min\{Y(z) |\ z \in Succ(i)\} & , Succ(i) \neq \varnothing \end{cases} \tag{5}$$

In Eq. (5), z represents the "successor task" associated with task i, and K_{all} denotes the total deadline for the entire task flow. $Succ(i)$ is the set of all direct successor tasks of task i, that is, the tasks to which task i has directed edges. $Y(z)$ represents the latest start time of task z. The expression for $Y(i)$ is presented in Eq. (6).

$$Y(i) = X(i) - G(i) \tag{6}$$

Based on the DAG dependency modeling and the resource characteristics of fog computing, the TS model developed in this research sets “minimizing the total EC of fog computing” and “minimizing the maximum response time of tasks” as its central objectives. This model incorporates three major constraints: dependency constraints, resource constraints, and delay constraints. The expression of MO optimization function is shown in Eq. (7).

$$\begin{cases} \min A_{all} = \min(A_c + A_f) \\ \max T = \max\left\{\left[U(i) + G(i) + \Delta t_u(i)\right] - L_{start}(i) \mid i = 1, 2, ..., n\right\} \end{cases} \tag{7}$$

In Eq. (7), $L_{start}(i)$ represents the arrival time of task i, and n represents the total number of tasks. $\Delta t_u(i)$ represents the transmission delay corresponding to the target resource (cloud/fog) allocated by task i, which is preliminarily determined by the allocation strategy. The expression of $\Delta t_u(i)$ is shown in Eq. (8).

$$\begin{cases} \Delta t_u(i) = \min\left(\Delta t_{u1}(i), \Delta t_{u2}(i), \Delta t_{u3}(i)\right) \\ \Delta t_{u1}(i) = \dfrac{Z(i)}{S_{T-F}} \\ \Delta t_{u2}(i) = \dfrac{Z(i)}{S_{T-F}} + \dfrac{Z(i)}{S_{F-C}} + t_{F-proc}(i) \\ \Delta t_{u3}(i) = \dfrac{Z(i)}{S_{T-F1}} + \dfrac{Z(i)}{S_{F1-F2}} + t_{F1-sync}(i) \end{cases} \tag{8}$$

In Eq. (8), $\Delta t_{u1}(i)$, $\Delta t_{u2}(i)$, and $\Delta t_{u3}(i)$ respectively represent direct execution from terminal to fog node, forwarding from terminal to fog node to cloud node, and collaboration from terminal to fog node to other fog nodes. S_{T-F} is the bandwidth from the terminal to the target fog node, and $Z(i)$ is the data volume of task i. S_{F-C} is the bandwidth from the fog node to the cloud node, and $t_{F-proc}(i)$ represents the forwarding preprocessing time of task i by the fog node. S_{F1-F2} is the bandwidth from the source fog node ($F1$) to the target fog node ($F2$), and $t_{F1-sync}(i)$ represents the task synchronization time between fog nodes. S_{T-F1} is the communication bandwidth from the terminal to the first fog node. The constraint conditions are shown in Eq. (9).

$$\begin{cases} L(i) \geq \max\left\{O(j) |\ j \in Pred(i)\right\} \\ N_m(g) \leq N_{\max} \\ \max\left\{O(i)\right\} \leq K_{all} \\ U(i) + G(i) + \Delta t_u(i) \leq D(i) \end{cases} \tag{9}$$

In Eq. (9), $D(i)$ represents the deadline for task i. $L(i)$ represents the start time of i, and $O(j)$ is the completion time of j. $N_{\max}$ represents the maximum allowable utilization rate of any resource, and $N_m(g)$ is the utilization rate of resource m at time g. $\max\left\{O(i)\right\}$ represents the maximum of all task completion times. The basic slack only considers the time margin of task execution itself, but there is a significant difference in transmission delay in the three-level architecture of “terminal fog cloud” in cloud computing. If this delay is ignored, it will lead to an overestimation of the task’s actual executable window (Kumar et al., 2024; Wu et al., 2025). According to the “End Edge Cloud Collaborative Scheduling Theory”, the effective slack of tasks needs to meet the requirement of “task executable

window ≥ transmission delay+execution delay" (Mirzapour-Moshizi et al., 2024). Therefore, it is necessary to introduce transmission delay to correct the basic slack and ensure that the priority ranking aligns with the actual resource allocation in cloud and fog environments. The expression of slack priority $Q(i)$ is presented in Eq. (10).

$$Q(i) = Y(i) - U(i) = X(i) - V(i) \tag{10}$$

The time-consuming $\Delta t_c(i)$ of task transmission from terminal to cloud is shown in Eq. (11).

$$\Delta t_c(i) = \frac{R(i)}{S_c} \tag{11}$$

In Eq. (11), $R(i)$ represents the data volume of task i, and S_c is the bandwidth from the terminal to the cloud. The time-consuming $\Delta t_f(i)$ of task transmission from terminal to fog is shown in Eq. (12).

$$\Delta t_f(i) = \frac{R(i)}{S_f} \tag{12}$$

In Eq. (12), S_f represents the bandwidth from terminal to fog. The corrected slack priority is shown in Eq. (13).

$$Q(i)' = Q(i) - \Delta t_u(i) \tag{13}$$

By solving the slack priority, the final ordered task sequence is obtained. To better determine priorities, the study integrated the DAG and the Critical Path Method (CPM). Consequently, the final revised slack priority is shown in Eq. (14).

$$\begin{cases} Q_{key}(i)' = 1.2 \times \left(Q(i) - \Delta t_u(i)\right) \\ Q_{non\text{-}key}(i)' = 1.0 \times \left(Q(i) - \Delta t_u(i)\right) \end{cases} \tag{14}$$

In Eq. (14), $Q_{key}(i)'$ represents critical tasks and $Q_{non\text{-}key}(i)'$ represents non-critical tasks. When prioritizing tasks, the first step is to determine whether the task belongs to the critical path: critical tasks are prioritized in the scheduling queue, and tasks belonging to the critical path are sorted in ascending order according to the corrected $Q_{key}(i)'$ value (the lower the value, the higher the priority). Non-critical tasks are sorted in ascending order of $Q_{non\text{-}key}(i)'$ value and are scheduled only after critical task resources are allocated.

2.2. Design of Task Sequence Solving Method Considering an Improved ACO Algorithm

To solve task sequences for fog computing, an improved ACO algorithm has been designed. The ACO algorithm boasts advantages such as strong global search capabilities, high robustness, and ease of integration with other algorithms (Shreyas et al., 2025). The main process of the ACO algorithm is illustrated in Fig. 4.

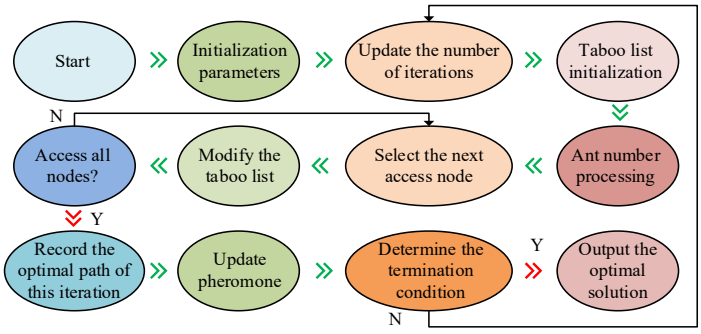


Fig. 4. The main process of the ACO algorithm

From Fig. 4, the main process of the ACO algorithm involves initializing the tabu list and modifying the tabu list. The task encoding method corresponding to Fig. 4 adopts integer encoding based on the DAG sequence. Each task is represented by a unique number (T1-T11), mapping the order of task execution to ant path selection. Meanwhile, embedding DAG task dependencies into taboo tables ensures that ants only select successor tasks that satisfy dependency constraints, achieving effective expression of scheduling schemes and feasible solution search. However, the ACO algorithm also has certain shortcomings, such as the tendency for the heuristic factor to take on extreme values and the susceptibility to getting trapped in local optima. Therefore, two improvements have been made to the algorithm. The first improvement involves introducing a logarithmic function to adjust the state transition probability formula, thereby suppressing extreme values of the heuristic factor. The revised state transition probability is shown in Eq. (15).

$$P_{ab}^{h}(t)=\frac{e^{\varphi\log\left(\delta_{ab}(x)\right)+y_1}e^{\theta\log\left(\mu_{ab}(x)\right)+y_2}}{\sum_{q\in w}e^{\varphi\log\left(\delta_{aq}(x)\right)+y_1}e^{\theta\log\left(\mu_{aq}(t)\right)+y_2}} \tag{15}$$

In Eq. (15), h represents ants, a and b represent different nodes, and q represents optional nodes. x is the current number of iterations, $\delta_{ab}(x)$ and $\delta_{aq}(x)$ are the pheromone concentrations between points a and b and between a and q at the x th iteration, respectively, and $\mu_{ab}(x)$ and $\mu_{aq}(x)$ are heuristic functions. w is the logarithmic transformation, and φ is the pheromone importance factor. θ is the heuristic information importance factor, y_1 and y_2 are linear combination bias terms. w is the allowed access set. The second improvement is the introduction of IPSO algorithm to improve it, in order to optimize the core parameters φ and θ of ACO through the group search ability and fast convergence characteristics of PSO. PSO algorithm has the advantages of fast convergence speed, few parameters, easy adjustment and strong group cooperation (Bansal and Aggarwal, 2024b; Gu et al., 2025). The main process of PSO is shown in Fig. 5.

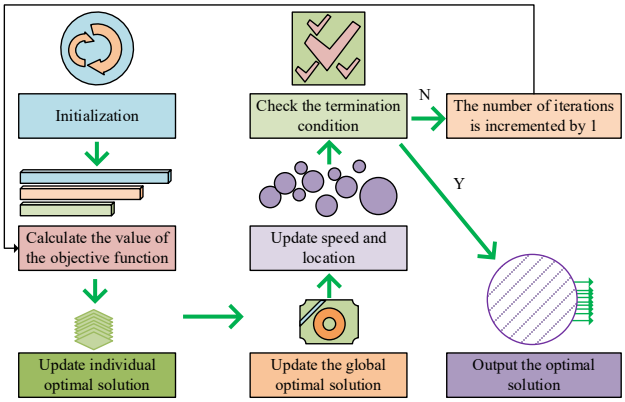


Fig. 5. The main process of PSO

From Fig. 5, the main process of the PSO algorithm encompasses initialization and calculating the objective function values. However, the PSO algorithm also has certain limitations, such as the overly simplistic setting of the inertia weight W. Therefore, the research introduces a nonlinear inertia weight W' to improve the algorithm. The expression for W' is shown in Eq. (16).

$$W'=W\cdot e^{-\frac{0.9x}{x_{\max}}} \tag{16}$$

In Eq. (16), $x_{\max}$ represents the max iteration count. Based on this nonlinear mapping, the inertia weight is no longer a static constant. The main process of the final ACO-IPSO algorithm is illustrated in Fig. 6.

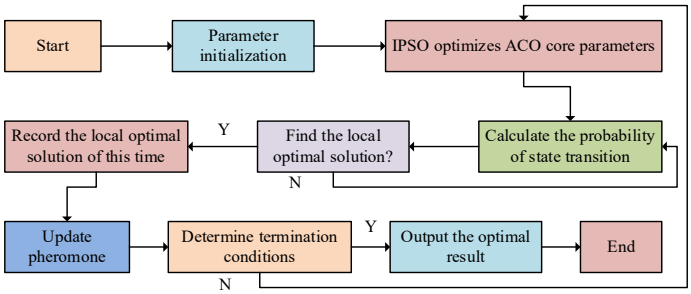


Fig. 6. The main process of the final ACO-PSO algorithm**.**

From Fig. 6, the main process of the ACO-IPSO algorithm includes parameter initialization and optimizing the core parameters φ and θ of ACO using IPSO. Therefore, through this algorithm, the research can output an optimal TS scheme. The code of ACO-IPSO algorithm is shown in Table 1.

3. Results and Analysis

To validate the performance of the scheduling model and algorithm designed in the research, the study first configured the experimental environment, including settings for the operating system and model parameters.

3.1. Experimental Environment Setting

3.2. Performance Verification of the ACO-IPSO Algorithm

To validate the performance of the ACO-IPSO algorithm, the study employed test functions from the CEC2013 test function suite (Hebbi and Mamatha, 2023). The F1 test function is a unimodal benchmark function in the CEC2013 test set, used to test the algorithm's global optimization ability and convergence speed. The example dataset used to validate the ACO-IPSO algorithm is shown in Table 3.

Table 1. Code of ACO-IPSO algorithm

```
def IPSO_opt_ACO(ar, br, mi):
    ps = init_ps(ar, br)
    gb = min(ps, key=lambda p: fit(p[0], p[1]))
    for i in range(mi):
        W = 0.9*exp(-i/mi)
        for p in ps:
            upd_v(p, W, gb), upd_p(p, ar, br)
            p[2] = fit(p[0], p[1])
            gb = min(gb, p, key=lambda x: x[2])
    return gb[0], gb[1]
```

Table 2. Specific experimental environment information

Type		Description	Type	Description
Operating system		Windows 10 Professional 64-bit	Algorithm implementation	NVIDIA GeForce RTX 3060 GPU
CPU	Model	Intel Core i7-12700K	Programming language	Python 3.8
	Reference frequency	3.6GHz	Development tools	PyCharm 2023.1
	Highest Rui frequency	5.0GHz	Relevant libraries	SciPy, NetworkX
Fog computing simulation tool		FogBus 2.0	Cloud computing simulation tool	CloudSim 4.0
φ		2.5(optimization results)	θ	3.0(optimization results)
Maximum iteration count		300	Learning rate	0.001
Number of tasks		11 (T1-T11, including video processing and status monitoring)	Number of fog nodes	8
Cloud node count		4	Terminal-fog bandwidth	500Mbps
Terminal-cloud bandwidth		100Mbps	\	\

For comparative models, the study chose the ACO algorithm, the ACO-PSO algorithm, combination model A, and combination model B. Among them, combination model A is a scheduling model that integrates the artificial bee colony algorithm and fuzzy logic, while combination model B is a scheduling model that integrates the grey wolf optimization algorithm and the least squares support vector machine. With the help of swarm intelligence and a positive feedback mechanism, the ACO algorithm can optimize the task resource allocation path in cloud and fog scheduling and enhance distributed optimization capability. The ACO-PSO algorithm combines the ACO pheromone mechanism with the PSO particle iteration strategy, balancing global exploration and local development, and is suitable for MO optimization of large-scale tasks. Combination model A used bee colony search to match task resources, fuzzy logic to process fuzzy indicators, and improved scheduling robustness. Specifically, the combination model A adopts the artificial bee colony algorithm for task and resource matching search. Through the division of labor and cooperation among leading bees, following bees, and reconnaissance bees, it traverses the solution space to search for the optimal task allocation scheme, balancing global exploration and local development capabilities. Meanwhile, fuzzy logic is introduced to handle uncertain information in the scheduling process. Node load rate, delay tolerance, and remaining resource availability are selected as core fuzzy

indicators, and these continuous indicators are transformed into a quantifiable scheduling decision basis through fuzzification, rule inference, and deblurring. By utilizing the global optimization capability of bee colony search and the adaptive processing capability of fuzzy logic for dynamic and fuzzy information, the model can effectively cope with unstable factors such as fluctuations in cloud and fog node load, changes in network bandwidth, and sudden increases in task volume. This significantly improves the stability and reliability of scheduling results in complex and changing scenarios, thereby enhancing scheduling robustness. The combination model B used the wolf pack hunting mechanism for global optimization scheduling, and the least squares support vector machine accurately modeled resource and task features to achieve efficient task resource matching in complex cloud and fog environments. All comparison algorithms were executed on the same hardware platform, using the same version of the Python library, and the random number seed was uniformly set to 42, excluding the influence of hardware, software, and parameter differences on the results. A comparison of the standard deviations of different algorithms across various test functions is illustrated in Fig. 7.

Table 3. The example dataset used to validate the ACO-IPSO algorithm

Function name	Expression	Variable resolution
$f1$	$f_1(x)=\sum_{i=1}^{D}x_i^2$	D is the dimension, x_i is the i-th decision variable, W_i is the weight coefficient of the i-th basic function, F_i is the i-th basic test function that constitutes the composite function, M_i is the rotation matrix corresponding to the i-th basic function, o_i is the optimal value offset vector of the i-th basic function, and x is the current solution vector
$f10$	$f_{10}(x)=\sum_{i=1}^{D}\left(x_i^2-10\cos(2\pi x_i)+10\right)$	
$f19$	$f_{19}(x)=\sum_{i=1}^{10}w_i\cdot F_i(M_i(x-o_i))$	

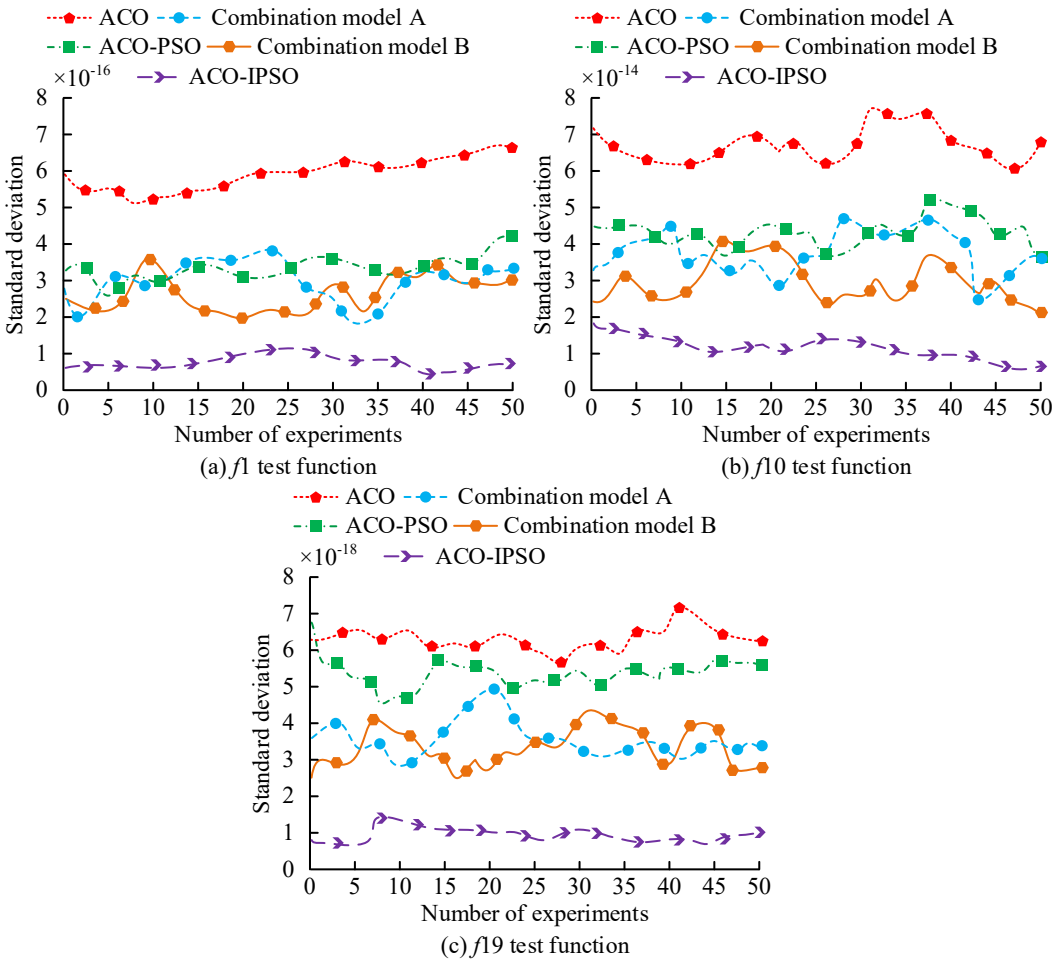


Fig. 7. Comparison of standard deviations for different algorithms under various test functions

From Fig. 7(a), under the $f1$ test function, the ACO-IPSO algorithm exhibited a maximum standard deviation of 1.13×10^{-16} and a minimum of 0.45×10^{-16}. As shown in Fig. 7(b), under the $f10$ test function, the ACO-IPSO algorithm,

designed in the study, continued to demonstrate a smaller standard deviation, with values ranging from [0.55×10^{-14}, 1.87×10^{-14}]. In contrast, the minimum standard deviations for the other four comparative algorithms all exceeded 2.0×10^{-14}. From Fig. 7(c), it is evident that under the $f19$ test function, the maximum standard deviations for the study's designed algorithm and the comparative algorithms were 1.48×10^{-18}, 7.14×10^{-18}, 6.87×10^{-18}, 4.99×10^{-18}, and 4.35×10^{-18}, respectively. In summary, the ACO-IPSO algorithm demonstrated superior stability across all test scenarios, exhibiting the smallest fluctuations in output results. In addition, all differences were found to be $p<0.05$, indicating that the advantage of ACO-IPSO in standard deviation and stability was statistically significant and not caused by random fluctuations. A comparison of the optimal values achieved by different algorithms under various test functions is illustrated in Fig. 8.

From Fig. 8(a), the optimal value achieved by the ACO-IPSO algorithm under the $f1$ test function was also significantly smaller than those of the comparative algorithms. From Fig. 8(b), it is evident that under the $f10$ test function, the ACO-IPSO algorithm, designed in the study, continued to perform better. According to Fig. 8(c), under the $f19$ test function, the maximum optimal values for the study's designed algorithm and the comparative algorithms were 1.86×10^{-16}, 7.54×10^{-16}, 7.13×10^{-16}, 6.87×10^{-16}, and 6.34×10^{-16}, respectively. In summary, across the three types of test functions, the optimal values of the ACO-IPSO algorithm were significantly superior to those of the comparative algorithms. In addition, all differences were statistically significant with $p<0.05$, indicating that ACO-IPSO had advantages in optimal values and stability, rather than being caused by random fluctuations.

To further validate the performance of the ACO-IPSO algorithm, a comparison of cutting-edge algorithms was introduced, including Deep Q-Network (DQN), Asynchronous Advantage Actor-Critic (A3C), Graph Convolutional Network (GCN), Graph Attention Network (GAT), Simplex method, and interior point method. The study investigated the impact of GPU acceleration on algorithm performance and analyzed memory usage monitoring data. The results are shown in Table 4.

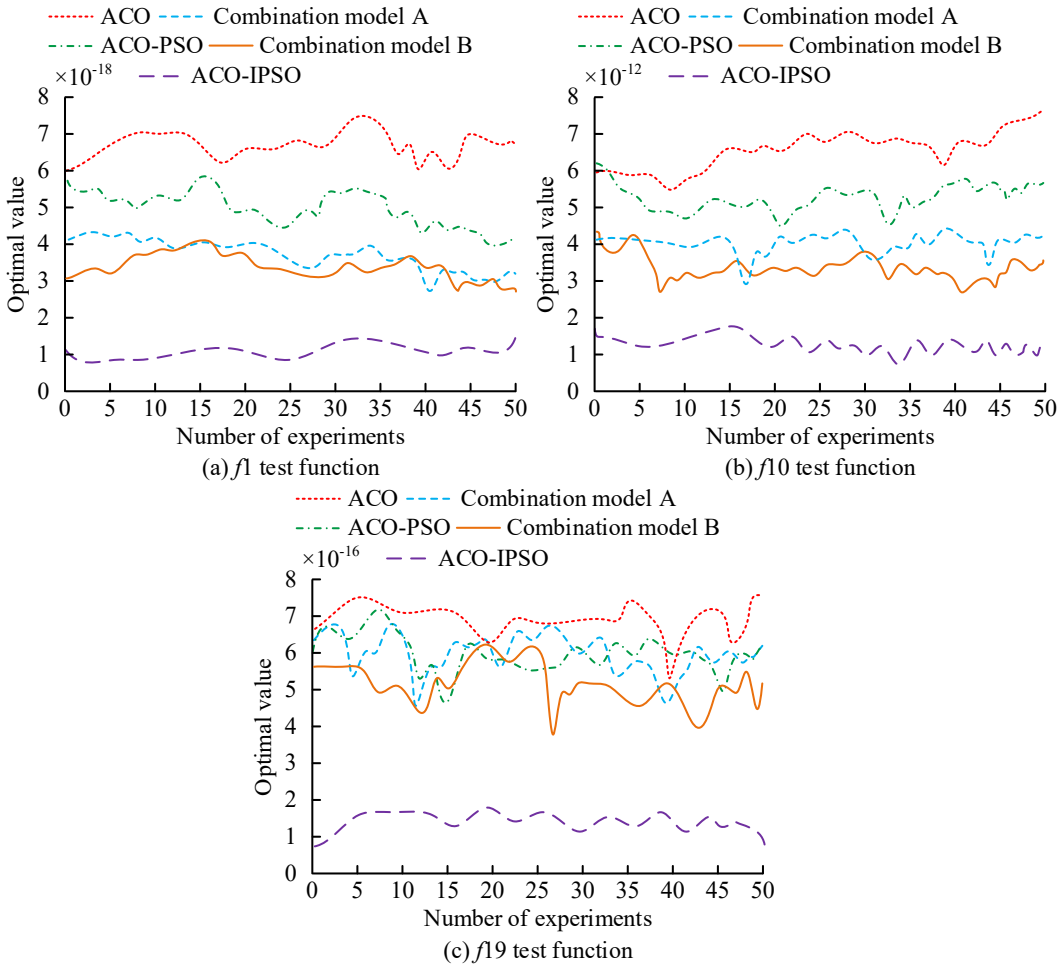


Fig. 8. Comparison of optimal values obtained by different algorithms under various test functions

In Table 4, after acceleration on the GPU, ACO-IPSO's scheduling time decreased by 73.96%, demonstrating outstanding performance advantages. Intelligent algorithms such as DQN also experienced a sharp reduction in GPU-acceleration time, whereas traditional algorithms showed little acceleration. The overall performance of ACO-IPSO was superior to the compared algorithms, and GPU significantly improved the performance of intelligent algorithms, but limited effectiveness on traditional algorithms.

Theorem 1: In cloud computing environments, the ACO-IPSO algorithm converged to the global optimal solution with

a probability of 1.

To prove Theorem 1, a state Markov chain for the ACO-IPSO algorithm was constructed, where the iterative process of the algorithm satisfies Markovian properties: the $g+1$th state depended only on the g th state and was independent of earlier states. Meanwhile, the study also demonstrated the connectivity of the state space, that is, in the hierarchical model of cloud computing resources, any task resource allocation path could be transformed through pheromone updates without isolated states. In addition, the study also verified the ergodicity of the probability transition matrix. Logarithmic transformation made the transition probabilities all positive, and the presence of a positive integer H made the H - power elements of the probability transition matrix are all positive.

Table 4. Impact of GPU acceleration and memory data analysis

Algorithm	Average scheduling time/ms		Peak memory usage/MB	
	No GPU acceleration	GPU acceleration available	No GPU acceleration	GPU acceleration available
ACO-IPSO	480	125	890	920
DQN	620	180	1250	1310
A3C	580	160	1180	1230
GCN	750	210	1560	1620
GAT	810	235	1680	1750
Simplex method	350	320	420	430
Interior point method	320	295	380	390

3.3 Example Verification of CFC TS Model

To conduct an empirical validation of the CFC TS model designed in the study, the following scenario was constructed: Suppose there was a surveillance system in a smart city, which encompassed many camera terminal devices distributed across various corners of the city. These terminal devices continuously generated video stream data processing tasks. A task set T = {T1 through T11} was also established. Within the CFC environment, fog nodes were composed of edge servers deployed near various neighborhoods, while cloud nodes were high-performance server clusters located in data centers. Furthermore, the study compared the designed TS model with the traditional fixed-priority scheduling model and the "first-come, first-served" scheduling model. Attributes of the smart city monitoring task set are shown in Table 5.

Table 5. Smart city monitoring task set attributes

Task	Data volume/MB	Execution time/ms	Dependent task	Deadline/ms	Task type
T1-video capture	150	30	None	1000	Delay-sensitive type
T2-frame filtering	120	25	T1	1050	Compute-intensive
T3-sensor acquisition	10	5	None	1000	Non-real time type
T4-sensor acquisition	8	4	None	1000	Non-real time type
T5-facial feature extraction	100	40	T2	1100	Delay-sensitive type
T6-vehicle feature extraction	80	35	T2	1100	Delay-sensitive type
T7-initial judgment of abnormality	50	20	T5,T6	1150	Compute-intensive
T8-fog load monitoring	5	3	T3,T4	1000	Non-real time type
T9-global fusion	200	50	T7	1200	High complexity type
T10-exception confirmation	10	15	T9	1250	Delay-sensitive type
T11-storage	20	8	T8,T10	1300	Non-real time type

The DAG constructed by the designed TS model and the computed task priority results are illustrated in Fig. 9.

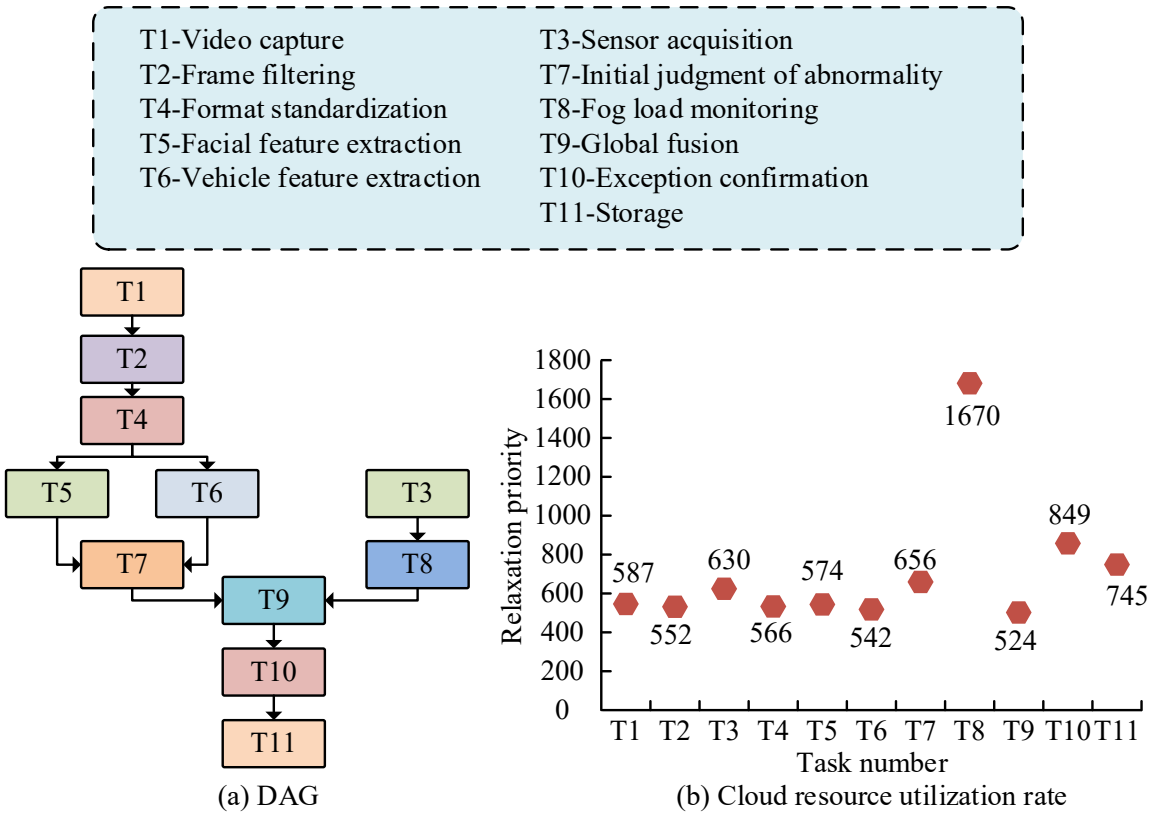
As shown in Fig. 9(a), the DAG contains three parallel dependency chains and one global aggregation chain. In Fig.

9(b), the slack priority values for the 11 tasks were 587, 552, 630, 566, 574, 542, 656, 1670, 524, 849, and 745, respectively. There were variations in the slack values among the tasks. A comparison of cloud and fog resource utilization rates among different models is shown in Fig. 10.

As shown in Fig. 10(a), in terms of fog resource utilization, the scheduling solution derived from the study's scheduling model performed better, with a maximum utilization rate of 93.58%. As shown in Fig. 10(b), concerning cloud resource utilization, the scheduling solution obtained from the study's designed scheduling model continued to outperform, achieving an average utilization rate of 86.54%. From Fig. 10(c), in the time-varying curve of fog resource utilization rate, the peak utilization rate of the scheduling scheme in this paper was high, and the fluctuation was small, maintaining 70% -93.58% for 100-500ms. Traditional fixed priority fluctuated, while 'first-come, first-served' had a higher utilization rate but fell back quickly in the later stages. From Fig. 10(d), in the curve of cloud resource utilization over time, the scheduling scheme proposed in this paper still performed better, with a maximum value of 86.17% and no overload. A comparison of the success and failure rates of different models is presented in Table 6.

In Table 6, the success rates of the scheduling scheme proposed in this paper for computationally intensive and delay-sensitive tasks were 89.65% and 84.29%, respectively, which were significantly higher than the comparison model, and the failure rate was much lower than that of the comparison model. This scheduling scheme had stronger adaptability to computationally intensive and time-sensitive tasks. A comparison of task completion times and system response times for different models is shown in Fig. 11.

As shown in Fig. 11(a), the video stream task chain completion time corresponding to the study's designed scheduling model was 820ms. As shown in Fig. 11(b), the response time was positively correlated with the critical path length. The traditional fixed priority scheduling model grew the fastest, followed by "first-come, first-served". The scheduling scheme proposed in this paper had the slowest growth rate, indicating that its response-time-optimization effect was better in long critical-path task scenarios. The comparison of service quality and energy efficiency among different models is shown in Table 7.



(a) DAG (b) Cloud resource utilization rate

Fig. 9. DAG and task priority calculation results constructed by the designed TS model

In Table 7, the deadline violation rate of the research and design scheduling scheme was only 0.8%, and the average EC per task was 25.3 millijoule (mJ) ((millijoule) is used as a unit of energy), which was significantly better than those of the traditional fixed-priority and first-come, first-served scheduling models. The research and design of scheduling schemes had outstanding performance in service quality and energy efficiency, and were suitable for the low consumption and high efficiency requirements of cloud computing. The scalability analysis of the cloud and fog computing TS model is shown in Table 8.

From Table 8, the cloud and fog computing TS model controlled the core time within a reasonable range when expanding task scale, resource types, scene adaptation, and constraint conditions, and the utilization rate of cloud and fog

resources fluctuated slightly. The model had strong scalability and could ensure scheduling efficiency and resource utilization efficiency under multi-dimensional expansion, adapting to multi-scenario applications such as smart cities and connected vehicles.

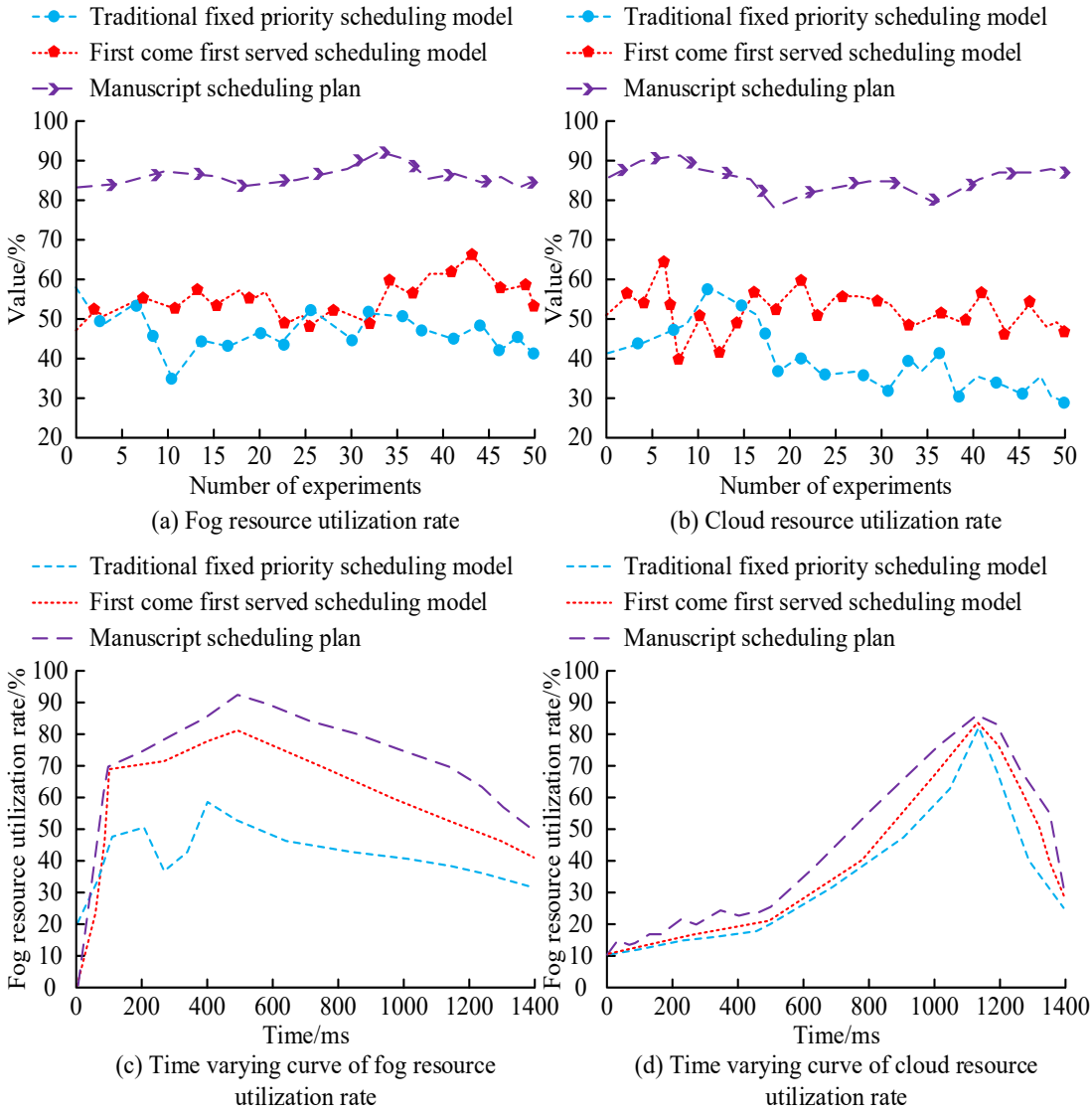


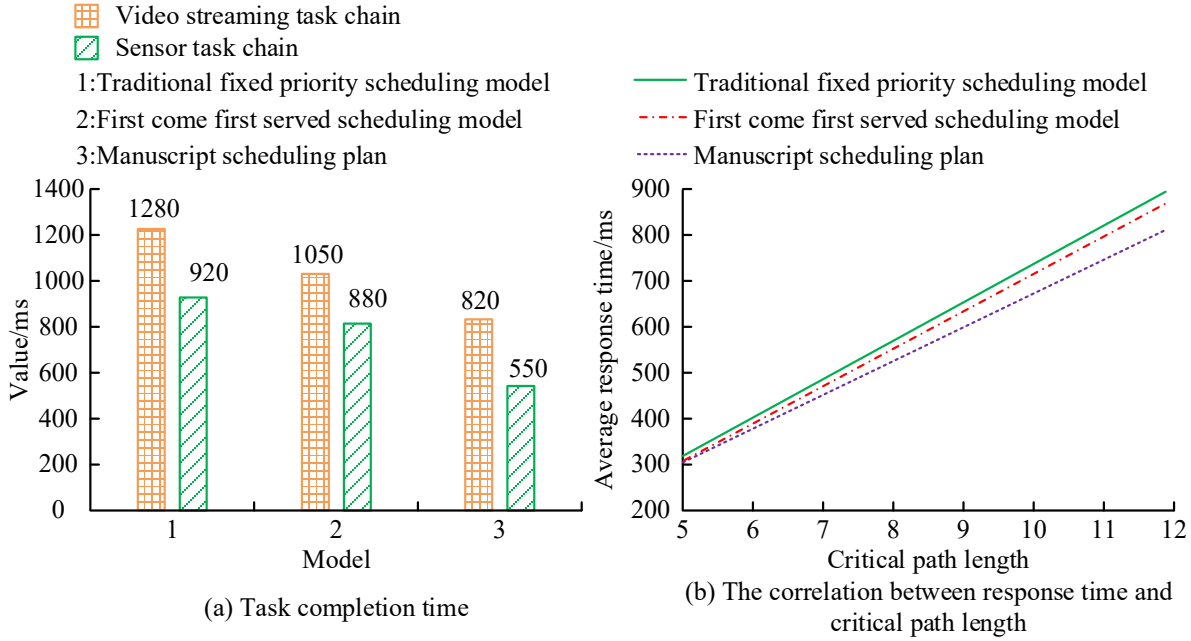
Fig. 10. Comparison of cloud and fog resource utilization rates of different models

Table 6. Comparison of success and failure rates of different models

Model	Success rate/%		Failure rate/%	
	Task type		Task type	
	Compute-intensive	Delay-sensitive type	Compute-intensive	Delay-sensitive type
Traditional fixed priority scheduling model	62.58	55.32	37.42	44.68
First come first served scheduling model	72.15	63.87	27.85	36.13
Manuscript	89.65	84.29	10.35	15.71

The cloud-edge computing TS model developed in this study was based on three core assumptions: the collaboration of a three-tier architecture (“terminal-edge-cloud”), the ability to fully characterize task dependencies through DAGs, and real-time resource load state awareness. From a technical feasibility perspective, current edge computing devices already possess dynamic bandwidth monitoring and task priority identification capabilities, aligning with existing technological trends. From a scenario adaptability standpoint, the dependency relationships among task types in smart city surveillance scenarios exhibited no cyclic characteristics, highly matching the “acyclic” nature of DAGs. Moreover, the topological

sorting of 11 tasks in experiments confirmed no logical conflicts in their dependency chains, further validating the rationality of these assumptions.



(a) Task completion time

(b) The correlation between response time and critical path length

Fig. 11. Comparison of task completion time and system response time for different models

Table 7. Comparison of service quality and energy efficiency among different models

Model	Deadline violation rate/%	Average EC per task/mJ
Traditional fixed priority scheduling model	6.5	40.7
First come first served scheduling model	7.2	42.3
Manuscript scheduling plan	0.8	25.3

4. Conclusion

To address the TS problem in CFC, the study constructed a DAG TS model incorporating slack priority and designed the ACO-IPSO algorithm to solve it. The results revealed that in the performance testing of the ACO-IPSO algorithm, its standard deviation and optimal values under unimodal, multimodal, and composite functions were all smaller than those of the comparative algorithms. For instance, under the $f19$ test function, the maximum standard deviations of ACO-IPSO and the comparative algorithms were 1.48×10^{-18}, 7.14×10^{-18}, 6.87×10^{-18}, 4.99×10^{-18}, and 4.35×10^{-18}, respectively. In the empirical validation, regarding cloud-fog resource utilization, success rate, failure rate, task completion time, and system response time, the scheduling solution corresponding to the study's designed scheduling model consistently performed better. Both the ACO-IPSO algorithm and the scheduling model designed in the study exhibited favorable performance. The scope of application of ACO-IPSO was small and medium-sized, statically dependent cloud and fog scheduling scenarios, especially suitable for scenarios such as smart city monitoring, small and medium-sized industrial Internet of Things with a task quantity of ≤ 100 and a critical path length of ≤ 15. Meanwhile, computational overhead may arise in large-scale scenarios, leading to longer initialization times and reduced real-time performance.

Subsequent research can focus on enhancing innovation in dynamic environment adaptability: firstly, designing a real-time resource perception module to collect real-time data on cloud and fog node load, bandwidth, and other factors. The second is to establish a load forecasting model to predict changes in resource supply and demand based on historical data. The third is to implement a dynamic rescheduling mechanism that adjusts task allocation based on perception and prediction results. Against the background of digital twins, edge intelligence and the rapid popularization of 5G/6G technology, the above directions can not only improve the engineering implementation ability of algorithms in large-scale, high dynamic IoT scenarios, but also provide key support for engineering production management, smart city operation and industrial Internet scheduling from "static optimization" to "real-time autonomy", so that cloud computing TS can be upgraded from simple performance optimization to a closed-loop management system with autonomous perception, intelligent decision-making and adaptive adjustment, and lay a theoretical and technical foundation for subsequent research on cross scenario, scalable and highly robust scheduling systems.

Subsequent research can focus on enhancing innovation in dynamic environment adaptability: firstly, designing a real-time resource perception module to collect real-time data on cloud and fog node load, bandwidth, and other factors. The second is to establish a load forecasting model to predict changes in resource supply and demand based on historical data.

The third is to implement a dynamic rescheduling mechanism that adjusts task allocation based on perception and prediction results. Against the background of digital twins, edge intelligence and the rapid popularization of 5G/6G technology, the above directions can not only improve the engineering implementation ability of algorithms in large-scale, high dynamic IoT scenarios, but also provide key support for engineering production management, smart city operation and industrial Internet scheduling from "static optimization" to "real-time autonomy", so that cloud computing TS can be upgraded from simple performance optimization to a closed-loop management system with autonomous perception, intelligent decision-making and adaptive adjustment, and lay a theoretical and technical foundation for subsequent research on cross scenario, scalable and highly robust scheduling systems.

Table 8. Scalability analysis of cloud and fog computing TS models

Expand scenarios/requirements	Time-consuming	Cloud resource utilization rate	Fog resource utilization rate
The number of tasks has increased from 11 to 100	Single round scheduling calculation time: 11 items ≤ 200ms, 100 items ≤ 500ms	11 items 86.54%, 100 items 84.21%	11 items 93.58%, 100 items 90.17%
Add 8 5G base station edge nodes and 4 edge storage nodes	New node data interaction time: ≤ 8ms with fog nodes, ≤ 12ms with cloud nodes, scene resource configuration time ≤ 40ms	After adding new nodes, the decrease is ≥ 82.3% and ≤ 4.2%	Base station edge nodes ≥ 78.5%, storage nodes ≥ 83.7%
Expanding from smart city scenarios to remote healthcare and Industry 4.0	Medical response time ≤ 20ms, industrial time ≤ 10ms, scene switching configuration time ≤ 30ms	Medical ≥ 85.1%, Industry 4.0 scenarios ≥ 83.7%	Medical ≥ 89.2%, industrial ≥ 92.6%
New EC threshold constraint (total EC of clouds and mist ≤ 1000J/hour)	EC calculation time ≤ 20ms/round	Optimized ≥ 84.7%, decreased ≤ 1.8%	Optimized ≥ 91.3%, decreased ≤ 2.3%

In addition, after reading this article, engineering and production managers will change three core decisions in cloud and fog scheduling. Firstly, the task-priority decision-making: switching from fixed to relaxed priority and dynamic DAG-based critical path determination to prioritize the scheduling of critical tasks. Secondly, resource allocation decisions: shifting from single-cloud/fog resource allocation to cloud-to-cloud, collaborative, layered allocation, balancing resource utilization and latency. Thirdly, algorithm selection decision: Change from a single heuristic algorithm to an ACO-IPSO hybrid algorithm to avoid local optima and improve scheduling stability.

Funding

This research received no specific financial support from any funding agency.

Institutional Review Board Statement

Not applicable.

Declaration of Artificial Intelligence (AI) Tools

The authors used AI tools solely for language editing and readability improvement. The authors reviewed and verified all content and take full responsibility for the accuracy and integrity of the manuscript.

References

Ahmadabadi, J. Z., Mood, S. E., and Souri, A. (2024). Star-Quake: A new operator in multi-objective gravitational search algorithm for task scheduling in IoT-based cloud-fog computing system. *IEEE Transactions on Consumer Electronics*, 70(1), 907-915. https://doi.org/10.1109/TCE.2023.3321708

Ahmed, Y. N., Mohideen, S. P., and Pasha, M. (2023). Task scheduling in cloud-fog computing using discrete binary particle swarm meta-heuristic with modified sigmoid function. *Journal of Information & Optimization Sciences*, 44(6), 1023-1033. https://doi.org/10.47974/JIOS-1226

Ali, A., Shah, S. A. A., Al Shloul, T., Assam, M., Ghadi, Y. Y., Lim, S., and Zia, A. (2024). Multiobjective Harris Hawks optimization-based task scheduling in cloud-fog computing. *IEEE Internet of Things Journal,* 11(13), 24334-24352. https://doi.org/10.1109/JIOT.2024.3391024

Bansal, S., and Aggarwal, H. (2024a). A multiobjective optimization of task workflow scheduling using hybridization of PSO and WOA algorithms in cloud-fog computing. Cluster Computing: *The Journal of Networks, Software Tools and Applications,* 27(8), 10921-10952. https://doi.org/10.1007/s10586-024-04522-3

Bansal, S., and Aggarwal, H. (2024b). An efficient workflow scheduling in cloud-fog computing environment using a hybrid particle whale optimization algorithm. *Wireless Personal Communications*, 137(1), 441-475. https://doi.org/10.1007/s11277-024-11421-8

Daghayeghi, A., and Nickray, M. (2024). Delay-aware and energy-efficient task scheduling using strength Pareto evolutionary algorithm II in fog-cloud computing paradigm. *Wireless Personal Communications,* 138(1), 409-457. https://doi.org/10.1007/s11277-024-11510-8

Goel, G., and Tiwari, R. (2024). IGWOA: Improved grey wolf optimization algorithm for resource scheduling in cloud-fog environment for delay-sensitive applications. *Peer-to-Peer Networking and Applications,* 17(3), 1768-1790. https://doi.org/10.1007/s12083-024-01642-w

Gu, K., Liu, Z. L., and Jia, W. J. (2025). Location-aware reliable task cooperative-computation scheme under fog computing-based IoVs. *IEEE Transactions on Intelligent Transportation Systems,* 26(1), 425-442. https://doi.org/10.1109/TITS.2024.3485241

Hebbi, C., and Mamatha, H. (2023). Comprehensive dataset building and recognition of isolated handwritten Kannada characters using machine learning models. *Artificial Intelligence and Applications,* 1(3), 179-190. https://doi.org/10.47852/bonviewAIA3202624

Kumar, M. S., Reddy, K. G., and Donthi, R. K. (2024). SSKHOA: Hybrid metaheuristic algorithm for resource-aware task scheduling in cloud-fog computing. *International Journal of Information Technology and Computer Science,* 16(1), 1-12. https://doi.org/10.5815/ijitcs.2024.01.01

Li, H., Song, D. Z., and Zhu, J. T. (2024). A research on genetic algorithm-based task scheduling in cloud-fog computing systems. *Automatic Control and Computer Sciences,* 58(4), 392-407. https://doi.org/10.3103/S0146411624700512

Mentzer, K., Price, J., and Singh, J. (2024). Analyzing Reddit discourse surrounding generative AI. *Issues in Information Systems,* 25(3), 277-292. https://doi.org/10.48009/3_iis_2024_122

Mirzapour-Moshizi, M., Sattari-Naeini, V., and Molahosseini, A. S. (2024). Application placement in fog-cum-cloud environment based on a low latency policy-making framework. *Cluster Computing,* 27(1), 199-217. https://doi.org/10.1007/s10586-022-03954-z

Mousavi, S., Mood, S. E., Souri, A., and Javidi, M. M. (2023). Directed search: A new operator in NSGA-II for task scheduling in IoT based on cloud-fog computing. *IEEE Transactions on Cloud Computing*, 11(2), 2144-2157. https://doi.org/10.1109/TCC.2022.3188926

Rao, M. C., and Qin, H. (2024a). Enhanced hybrid equilibrium strategy in fog-cloud computing networks with optimal task scheduling. Computers, *Materials & Continua,* 79(5), 2647-2672. https://doi.org/10.32604/cmc.2024.050380

Rao, M. C., and Qin, H. (2024b). Enhanced hybrid equilibrium strategy in fog-cloud computing networks with optimal task scheduling. CMC–Computers, *Materials & Continua,* 79(2), 2647-2672. https://doi.org/10.32604/cmc.2024.050380

Shreyas, J., Kharat, R. S., Phursule, R. N., and Rao, M. V. B. (2025). Chaotic grey wolf optimization-based framework for efficient task scheduling in cloud-fog computing. *Bulletin of Electrical Engineering and Informatics,* 14(3), 2066-2076. https://doi.org/10.11591/eei.v14i3.8098

Shukla, P., and Pandey, S. (2024). MOTORS: Multi-objective task offloading and resource scheduling algorithm for heterogeneous fog-cloud computing scenario. *Journal of Supercomputing,* 80(15), 22315-22361. https://doi.org/10.1007/s11227-024-06315-2

Wu, L. B., Li, S. B., Wu, F. B., Xie, R. X., and Yuan, P. L. (2025). Multi-strategy enhanced hiking optimization algorithm for task scheduling in the cloud environment. *Journal of Bionic Engineering,* 22(3), 1506-1534. https://doi.org/10.1007/s42235-025-00674-z

Zhang, F. Q., Jiang, H. L., Liu, F., Hou, T., Liu, Y. J., Guan, Y. Q., and Mu, X. T. (2025). Task scheduling strategy for cloud-fog computing system of IoV based on improved quantum genetics and QoS awareness method. *Journal on Communications* (Tongxin Xuebao), 46(4), 91-107. https://doi.org/10.11959/j.issn.1000-436x.2025057

Lun A obtained his master's degree in computer application technology (2011) from Inner Mongolia University of Technology. Presently, he is working as an Associate Professor and the Dean of the Information Technology Teaching and Research Section in the Department of Information Engineering, Inner Mongolia Vocational College of Chemical Engineering. He has published articles in Chinese core journals. His areas of interest include machine learning, data mining, multilayer neural networks and information security.

Huchun Qi obtained his master's degree from Beijing Normal University. Presently, he is working as an Associate Professor in the Information Engineering Department, Inner Mongolia Vocational College of Chemical Engineering. He has published several papers in Chinese core journals. His areas of interest include data analysis and visualization, machine learning, and computer vision processing.

Xiaohai Guo obtained his master's degree from Inner Mongolia University of Technology. Presently, he is working as an Associate Professor and the Vice President of Inner Mongolia Chemical Engineering Vocational College. He has published several papers in Chinese core journals. His areas of interest include mathematical.