Future Generation Computer Systems 186 (2027) 108770

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Random forest algorithm-driven workflow cost optimization scheduling in a multi-cloud environment

Longxin Zhang [a,*], Lili Du [a], Mengying Guo [a], Zhihua Wen [a], Aijia Ouyang [b,*], Jiwu Peng [c], Keqin Li [d]

[a] School of Computer Science and Artificial Intelligence, Hunan University of Technology, Zhuzhou 412007, China
[b] School of Information Engineering, Zunyi Normal University, Zunyi 563006, China
[c] College of Information Technology and Management, Hunan University of Finance and Economics, Changsha 410205, China
[d] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Dataset link: https://github.com/lilidu1026/RFPSO

Keywords:
Cost optimization
Deadline constraints
Multi-cloud environment
Particle swarm optimization
Random forest

ABSTRACT

The high heterogeneity of resource types in multi-cloud computing environments and the complexity and diversity of billing models make the workflow scheduling problem more complex and challenging when scientific workflows need to be completed within strict deadline constraints. Although existing research has progressed in reducing the completion time of workflow scheduling in multi-cloud environments, many difficulties still exist in minimizing scheduling costs while satisfying deadlines. Therefore, a random forest enhanced particle swarm optimization algorithm (RFPSO) is proposed in this study. The RFPSO algorithm implements intelligent initialization of resource allocation through a random forest model, which improves the efficiency of finding optimal solutions. Moreover, it designs a reflective boundary constraint mechanism and a hierarchical task allocation mechanism based on critical path. This design ensures that critical tasks can prioritize access to higher-performance computing resources, which effectively guarantees that tasks within the workflow are scheduled and completed by their deadlines. In addition, the quality of the optimal solution is improved through a local neighborhood search mechanism. Experimental results on scientific workflow datasets such as Epigenomics and Montage show that, compared with existing state-of-the-art methods, RFPSO reduces execution costs by an average of 57.31%.

1. Introduction

In today's rapidly evolving field of cloud computing, the limitations of single-cloud service providers in terms of resource supply capacity and service continuity are becoming increasingly apparent as workflow scales continue to expand [1]. As a result, multi-cloud environments, which integrate resources from multi-cloud service providers, have become the core computing infrastructure for enterprises and research institutions to meet the diverse needs of workflow applications and workloads [2]. In the field of multi-cloud computing, major cloud service providers such as Amazon, Google, and Microsoft can offer users a rich and flexible array of solutions and services; this capability is due to their widely distributed computing and storage facilities [3]. Moreover, when a task set in a multi-cloud environment faces strict deadline constraints, the dynamic changes in resource performance, pricing, and task dependencies cause difficulty for traditional static scheduling methods to achieve optimal scheduling outcomes due to their lack of adaptive flexibility [2]. Therefore, metaheuristic algorithms with global search capabilities and good adaptability have received considerable attention. These algorithms can effectively address optimization problems with complex constraints and demonstrate significant application potential in meeting workflow task deadlines and improving resource utilization [4,5]. However, despite the widespread application of these algorithms in the field due to their strong global search capabilities, they still face challenges such as low resource utilization and insufficient flexibility in parameter tuning [6,7]. By contrast, machine learning algorithms can leverage dynamic prioritization and preemptive task scheduling mechanisms to allocate learning tasks to task queues with varying priorities [8]. As a result, high-priority tasks can access computing resources preferentially, which allow them to begin the training process earlier [9]. For example, Yang et al. [10] proposed a model that combines greedy optimization with machine learning. This model utilizes a multilayer perceptron neural network to

* Corresponding authors.
E-mail addresses: longxinzhang@hut.edu.cn (L. Zhang), lilidu1026@foxmail.com (L. Du), guomengyinging@foxmail.com (M. Guo), wzh@hut.edu.cn (Z. Wen), oyaj@foxmail.com (A. Ouyang), jiwu_peng@hnu.edu.cn (J. Peng), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.future.2026.108770
Received 12 April 2026; Received in revised form 22 July 2026; Accepted 17 August 2026
Available online 22 August 2026

predict task execution times based on historical data, which effectively reduces the execution time of workflows. In addition, Hayat et al. [11] used the random forest (RF) algorithm to evaluate the error rate of machine learning predictors. The results show that, under the marginal error rate, the RF method can enhance resource utilization and reduce execution time.

Ensemble learning models, represented by RF, have been widely applied in various fields due to their outstanding performance [12,13]. For instance, Dinnillah et al. [14] proposed a robust fishing detection method by combining isolation forest with RF, demonstrating the effectiveness of RF in noisy environments. Similarly, the study by Amalia et al. [15] indicates that RF exhibits significant stability when dealing with imbalanced data distributions. This characteristic effectively addresses the challenges of resource load fluctuations and uneven distribution of tasks and resources in multi-cloud scheduling scenarios, providing theoretical support for the application of RF in cloud scheduling optimization and laying a foundation for its use in dynamically changing cloud computing environments. This provides a foundation for applying RF in cloud computing environments characterized by dynamic resource changes. Although the aforementioned studies all employed RF as a classification model, they have not thoroughly explored its joint optimization with metaheuristic search algorithms. Among numerous algorithms, RF stands out due to its comprehensive performance, particularly demonstrating significant advantages in joint optimization with the particle swarm optimization (PSO) algorithm. Its main features include: (1) a task cost prediction method based on heterogeneous resource characteristics and dynamic billing data, which effectively generates low-cost, high-quality initial solutions, thereby enhancing the solution quality and convergence efficiency of the PSO algorithm; and (2) the use of a RF model for task scheduling, which satisfies the real-time requirements and efficient resource utilization demands of cloud environments due to its rapid construction and low computational resource consumption. To this end, this paper proposes a random forest enhanced particle swarm optimization algorithm (RFPSO), aimed at improving the overall performance of cloud computing task scheduling through this integration. This algorithm optimizes the initial solutions of PSO using the RF model, narrowing the search range of the particle swarm from a randomly distributed solution space to a concentrated set of high-quality solutions, thereby enhancing optimization efficiency. Meanwhile, RFPSO incorporates population diversity information to achieve dynamic adaptive adjustment of search parameters and designs a layered optimization mechanism for critical path slack, maximizing cost reduction in workflow scheduling while ensuring task completion deadlines. The main innovations of this paper include:

- An initial population optimization mechanism based on RF is proposed. This mechanism constructs a RF regression model that integrates task computation features and resource heterogeneity characteristics, while introducing a timeout cost penalty scoring mechanism to filter high-quality initial particles. This addresses the issues of low initial solution quality and slow convergence speed caused by random initialization in traditional metaheuristic algorithms. This method not only significantly enhances the overall quality of the initial population but also effectively accelerates the search process for the global optimal solution.
- A dynamic parameter adjustment and reflective boundary constraint strategy driven by population diversity is designed. This strategy quantifies the dispersion between the population and the centroid, dynamically adjusting the nonlinear decay process of inertia weight and learning factors to address the issues of insufficient adaptability of static parameters and ineffective searches caused by particles that exceed boundaries. Simultaneously, a reflective boundary handling mechanism is employed to correct the positions of out-of-bounds particles, significantly improving the search stability of the algorithm in multi-cloud heterogeneous environments.
- A layered optimization framework based on critical path slack is constructed. This framework identifies critical path tasks by calculating task slack and employs a dual-layer resource allocation strategy that balances high performance and low cost to address the cost waste caused by the ambiguity in scheduling priorities between critical and non-critical tasks. By integrating local neighborhood search techniques, this method effectively avoids the algorithm from getting trapped in local optima while prioritizing the satisfaction of critical task deadline constraints, thereby achieving global minimization of execution costs.

The rest of the study is structured as follows. Section 2 provides a systematic review of research in related fields, which highlights the current research progress. Section 3 formalizes the definition of the problem under study and systematically establishes models for workflows, resources, preprocessing, time, and cost. Section 4 provides a detailed introduction to the proposed RFPSO algorithm. Section 5 presents a comprehensive analysis of the experimental process and results. Section 6 concludes the study and discusses future research directions.

2. Related works

In multi-cloud environments, workflow scheduling under deadline constraints is challenging to optimize within a limited timeframe due to the heterogeneity of computational resources and the complexity of task dependencies [16]. Current research primarily focuses on heuristic, metaheuristic, and machine learning-based scheduling methods.

Heuristic scheduling methods. Early studies often employed heuristic algorithms, which are based on experience to establish rules for allocating appropriate computational resources to specific types of tasks within a defined deadline. Kamanga et al. [17] improved the limitations of the classical heterogeneous earliest finish time (HEFT) algorithm [18] in multi-objective optimization by proposing a heuristic method that integrates multi-criteria decision-making. This method minimizes execution costs by introducing central processing unit frequency as a core regulating factor. However, its application is limited to theoretical scenarios without time constraints due to the lack of a deadline constraint mechanism. Bano et al. [19] discussed a hierarchical heterogeneous earliest finish time model for multiple workflows, which combines a two-stage scheduling framework, level partitioning, and upward rank. In each partition, they select the virtual machine (VM) that yields the optimal completion time while considering inter-task communication, thereby achieving simultaneous optimization of workflow makespan and resource utilization. Alam et al. [20] devised a security-driven dynamic level scheduling algorithm for precedence-constrained task scheduling in IaaS clouds. By incorporating security overhead into the task prioritization and VM selection phases, and by matching tasks with high security demands to highly reliable VMs to reduce scheduling risk probability, the algorithm can simultaneously optimize workflow makespan and security risk under deadline constraints, achieving a synergistic improvement in both scheduling performance and security performance. Zhang et al. [21] devoted an energy-aware reliability enhancement scheduling strategy. This strategy improves the accuracy of time constraints through task priority sorting and dynamic sub-deadline allocation. Meanwhile, it optimizes VM selection in real time using a combination of pre-allocated energy consumption and dynamic correction mechanisms. As a result, it promotes resource allocation collaboratively throughout the entire task scheduling process, which significantly enhances system reliability. Notably, early research methods typically relied on manually established rules and static strategies. However, in dealing with the complex scheduling scenarios of multi-cloud environments with multiple constraints, these methods generally suffer from a lack of adaptability.

Meta-heuristic scheduling methods. This type primarily balances global exploration and local exploitation to avoid premature convergence, which generates global optimal solutions. Zhang et al. [22]

explored a budget-aware scheduling algorithm based on snake optimization, which effectively prevents premature convergence of the algorithm through a negative offset mechanism. In this way, the success rate of finding feasible solutions under budget constraints is improved. Li et al. [23] suggested a two-stage improved firefly algorithm (IFA). In the first stage, based on the theorem of secure service selection, IFA prioritizes the allocation of high-security-level services for tasks with small data volumes to meet security and encryption time constraints. The second stage minimizes computational costs through a distance mapping operator and an optimized position update method. Huang et al. [24] addressed the issue that the traditional gray wolf optimization algorithm experiences a drastic expansion of its search space when the task size increases, which leads to a decrease in search efficiency. To tackle this issue, they proposed a gray wolf optimization algorithm using a new encoding mechanism (GWOEM), which effectively compresses the search space and eliminates the dependence of the algorithm on population size parameters. However, this method is primarily suitable for independent task scheduling and is relatively insufficient for handling workflow tasks with complex dependencies. These metaheuristic methods reduce reliance on specific problem structures by simulating mechanisms found in nature. This way allows for a more effective escape from local optima in global search and enables the discovery of global optimal solutions. However, methods such as GWOEM still have limitations when dealing with workflow tasks that have complex dependencies. On the contrary, the RFPSO algorithm proposed in this study fully considers the dependency constraints between workflow tasks. Its boundary reflection mechanism ensures that particles search within the feasible solution space that satisfies these dependencies. Thus, it is suitable for more complex directed acyclic graph (DAG) workflow scenarios.

Machine learning-based scheduling methods. In multi-cloud computing scenarios, the volume of workflow data and the complexity of structures are growing explosively. This situation drives the deep integration of machine learning technologies into scheduling systems. By training models on historical data, the algorithms can capture the implicit relationships between tasks and resources, which enables dynamic resource allocation and real-time adjustments. The three-stage machine learning framework proposed by Patil et al. [25] innovatively combines three core modules: task classification, resource matching, and optimization algorithms. This framework effectively reduces task execution time and resource costs in multi-cloud computing environments. It works through the collaborative optimization of the gravitational search algorithm and genetic algorithm, which achieves a dual enhancement of efficiency and cost-effectiveness. The hybrid machine learning model designed by Zhou [26] focuses on accurately predicting resource loads and task demands. By integrating enhanced PSO [27] and graph attention neural networks, the objective is to simultaneously improve resource utilization and task execution efficiency during dynamic scheduling. This approach opens new avenues for intelligent management of cloud resources. To address the challenge of estimating task execution time, Yang et al. [10] combined regression models with greedy strategies. This combined approach uses time predictions to support resource allocation, which effectively reduces scheduling deviations and resource costs caused by uncertainty. Although machine learning-based methods significantly enhance the intelligence of scheduling decisions through the analysis of historical data, the training of these models often requires substantial costs. Moreover, their performance heavily depends on high-quality real-time data. Compared with the aforementioned scheduling methods, the RFPSO algorithm proposed in this study efficiently initializes using a RF model. It cleverly combines the data-driven advantages with the global search capabilities of metaheuristic algorithms. RFPSO can significantly enhance the efficiency and robustness of finding high-quality scheduling solutions under strict deadline constraints while avoiding the heavy training burden.

Table 1
Symbol definition table.

Notation	Definition
$\mathcal{N}$	Set of tasks in the scientific workflow
E	Set of dependency edges in the workflow
$cp(vm_z)$	Processing capacity of the vm_z
$T_{exc}(t_i)$	Execution time of task t_i
D	Deadline constraint for a specific workflow
C_{cal}	Computation cost of a specific workflow
C_{comm}	Cross-cloud transmission cost of a specific workflow
TEC	Total execution cost of a specific workflow
TET	Total execution time of a specific workflow
$EST(t_i, vm_z)$	Earliest start time of task t_i on vm_z
$LST(t_i, vm_z)$	Latest start time of task t_i on vm_z
$AST(t_i, vm_z)$	Actual start time of task t_i on vm_z
$AFT(t_i, vm_z)$	Actual finish time of task t_i on vm_z
$T_{slack}(t_i, vm_z)$	Slack time of task t_i on vm_z
w	Inertia weight in PSO
c_1, c_2	Learning factors in PSO
X^*_{new}	Global optimal solution in PSO

3. System model and problem description

This section mainly elaborates on the modeling of workflows and resources, the preprocessing model, the cost model, and the description of the optimization problem. Table 1 provides a detailed introduction to the key terminology used in this study.

3.1. Workflow and resource model

A workflow can be described as $\mathcal{W} = (\mathcal{N}, E)$, where $\mathcal{N} = \{t_1, t_2, \ldots, t_n\}$ is the set of tasks and $E = \{e(t_i, t_j) \mid t_i, t_j \in \mathcal{N}\}$ is the set of dependency edges. The weight $DT(t_i, t_j)$ of edge $e(t_i, t_j)$ represents the amount of data transferred from task t_i to task t_j, while the computational load of task t_i is denoted as $cd(t_i)$. All tasks need to run on designated VMs.

A multi-cloud environment consists of m providers $H = \{h_1, h_2, \ldots, h_m\}$, where the mth provider offers n types of VMs, which are denoted as $\mathcal{R} = \{vm_1, vm_2, \ldots, vm_n\}$. Their attributes include computational capacity cp, which represents the amount of computation that can be processed per second. As shown in Fig. 1, the cloud service provider integrates all physical resources of the data center to form a shared resource pool. Then, the data center utilizes virtualization technology to provide cloud services to users. This setup is a specific cloud resource architecture. Users then select and lease specific types of VM instances based on their needs, and the resource orchestrator dynamically allocates the corresponding resources. The tasks submitted by users are executed on the allocated VM and interact with the global distributed storage system for data exchange. Ultimately, users are billed based on their actual resource consumption. In terms of pricing models, Amazon EC2 charges on an hourly basis, Microsoft Azure utilizes a per-minute billing system, while Google Compute Engine employs a fixed fee plus overage billing model [28].

3.2. Preprocessing model

The average execution time $\overline{w(t_i)}$ of task t_i across all available resources is the arithmetic mean of the execution times of that task on each resource, which can be determined by

$$\overline{w(t_i)} = \frac{1}{r} \sum_{i=1}^{r} \frac{cd(t_i)}{cp(vm_z)}, \tag{1}$$

where r represents the total number of VM types, which is the sum of the VM types offered by all providers.

Definition 1 (*Task Priority (rank)*)**.** The priority value of task t_i is the weighted sum of its average execution time and the maximum $rank(t_i)$ among all its immediate successor tasks t_j. Starting from the exit task,

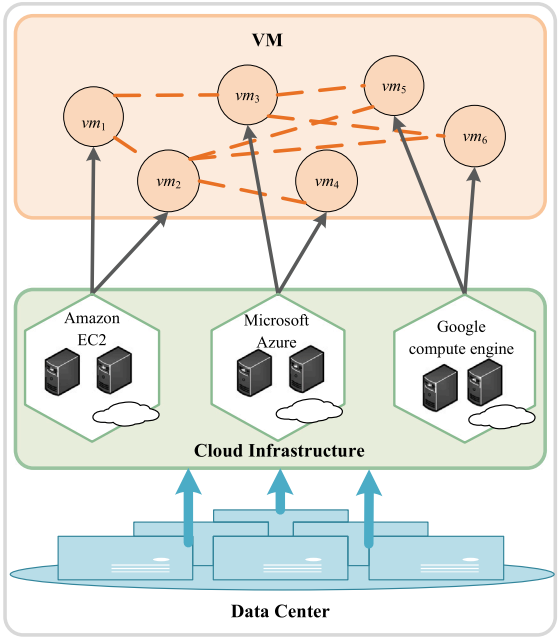

Fig. 1. Typical structure of multi-cloud resource model.

the DAG is traversed recursively in an upward direction. Thereafter, the tasks are organized in descending order of their calculated values to establish the scheduling sequence. The expression is computed as follows [22]:

$$rank(t_i) = \overline{w(t_i)} + \max_{t_j \in succ(t_i)} \left\{ rank(t_j) \right\}, \tag{2}$$

where $succ(t_i)$ represents the set of successor tasks of t_i.

3.3. Timing model

Based on the ratio of the computational load of the task to the computational capacity of the resource, the execution time $T_{exc}(t_i, vm_z)$ of each task t_i on the resource vm_z can be determined [29], and the expression is

$$T_{exc}(t_i, vm_z) = \frac{cd(t_i)}{cp(vm_z)}. \tag{3}$$

Definition 2 (*Communication Time (T_{comm})*)**.** When task t_i and task t_j are allocated to different VMs vm_z and vm_l, the communication time $T_{comm}(t_i, t_j, vm_z, vm_l)$ between them indicates the time required for data to be transmitted from task t_i to task t_j [30]. The calculation expression of $T_{comm}(t_i, t_j, vm_z, vm_l)$ is estimated by

$$T_{comm}(t_i, t_j, vm_z, vm_l) = \frac{DT(t_i, t_j)}{B_{intra} \cdot \vartheta + B_{inter} \cdot (1 - \vartheta)}. \tag{4}$$

Here, when vm_z and vm_l are from the same cloud provider, ϑ is set to 1. When vm_z and vm_l are from different cloud providers, ϑ is set to 0. B_{intra} represents the bandwidth within the same cloud provider, while B_{inter} denotes the bandwidth between different cloud providers.

Definition 3 (*Earliest Start Time (EST)*)**.** The earliest start time of task t_i is equal to the maximum value of the earliest completion times ($EFT(t_j, vm_l)$) of all its predecessor tasks plus the corresponding communication delays. This means that t_i can only begin execution after all predecessor tasks have been completed and all required data has arrived. The expression of $EST(t_i, vm_z)$ is estimated by [31]

$$EST(t_i, vm_z) = \max_{t_j \in pred(t_i)} \left\{ EFT(t_j, vm_l) + T_{comm}(t_i, t_j, vm_z, vm_l) \right\}. \tag{5}$$

Definition 4 (*Latest Start Time (LST)*)**.** The latest start time $LST(t_i, vm_z)$ refers to the latest permitted start time for task t_i after it is deployed to vm_z. The LST values for each task can be determined through reverse topological recursion. For exit tasks, their latest start times follow independent timing determination rules. Other tasks are calculated progressively based on the scheduling boundary conditions of their successor nodes. The specific expression for $LST(t_i, vm_z)$ is as follows:

$$LST(t_i, vm_z) = \begin{cases} EFT(t_i, vm_z), & succ(t_i) = \emptyset; \\ \min\limits_{t_j \in succ(t_i)} \left\{ LST(t_j, vm_l) \right. \\ \left. - T_{comm}(t_i, t_j, vm_z, vm_l) \right\} & otherwise. \\ - T_{exc}(t_i, vm_z), \end{cases} \tag{6}$$

Definition 5 (*Actual Start Time (AST)*)**.** The actual start time of task t_i on vm_z is defined as the maximum value between the current available time of vm_z and the sum of the actual completion times (AFT) of all predecessor tasks plus the corresponding data transmission times. The calculation expression for $AST(t_j, vm_l)$ is as follows:

$$AST(t_i, vm_z) = \max \Big\{ T_{avail}(vm_z), \max_{t_j \in pred(t_i)} \left\{ AFT(t_j, vm_l) + T_{comm}(t_i, t_j, vm_z, vm_l) \right\} \Big\}. \tag{7}$$

here, $T_{avail}(vm_z)$ is the available time of vm_z, $pred(t_i)$ represents the set of all predecessor tasks of task t_i.

Definition 6 (*Task Slack Time (T_{slack})*)**.** The difference between the latest start time and the earliest start time of task t_i is defined as the slack time $T_{slack}(t_i, vm_z)$ of task t_i on VM vm_z. The calculation expression for $T_{slack}(t_i, vm_z)$ is given as follows:

$$T_{slack}(t_i, vm_z) = LST(t_i, vm_z) - EST(t_i, vm_z). \tag{8}$$

Definition 7 (*Actual Completion Time*)**.** The actual completion time $AFT(t_i, vm_z)$ of task t_i on VM vm_z is obtained by adding the start time of task t_i on VM vm_z and its execution time, which represents the moment when the task execution ends. The calculation expression of $AFT(t_i, vm_z)$ is evaluated by

$$AFT(t_i, vm_z) = AST(t_i, vm_z) + T_{exc}(t_i, vm_z). \tag{9}$$

If task t_i is assigned to resource vm_z for execution, then the available time $T_{avail}(vm_z)$ of vm_z must be updated to the completion time of the task once the task is finished. The expression is given as follows [32]:

$$T_{avail}(vm_z) = AFT(t_i, vm_z). \tag{10}$$

Definition 8 (*Resource Usage Duration (T_{usage})*)**.** The total time duration from when vm_z starts until it completes all assigned tasks is referred to as its usage duration $T_{usage}(vm_z)$, and the calculation expression is

$$T_{usage}(vm_z) = AFT(vm_z) - T_{avail}^{init}(vm_z). \tag{11}$$

Definition 9 (*Deadline of a Task (D)*)**.** A linear interpolation strategy based on the HEFT algorithm is employed to determine a task's deadline by calculating the linear relationship between its fastest and slowest execution times. The definition of D is given as follows [33]:

$$D = T_{fast} + \alpha \cdot (T_{slow} - T_{fast}). \tag{12}$$

where T_{fast} is the fastest completion time, T_{slow} is the slowest completion time, and α is the deadline slack factor.

3.4. Cost model

Considering the varying billing policies of service providers, the definitions of computational and communication costs are given as follows.

(1) Hourly Billing C_{Az} (Amazon EC2 provider): Resources are settled on an hourly basis, and the cost equals the hourly rate of the resource multiplied by the actual usage duration. The expression of C_{Az} is estimated by [34]

$$C_{Az} = \sum_{i=1}^{n_{Az}} \left\{ \left\lceil \frac{T_{usage}(vm_i)}{T_h} \right\rceil \times P_{hour}(vm_i) \right\}, \tag{13}$$

where n_{Az} is the number of instances provided by Amazon EC2, $P_{hour}(vm_i)$ represents the hourly price of the ith instance, $T_h = 3600$, $\lceil\rceil$ denotes the ceiling function.

(2) Billed by the minute C_{Ma} (Microsoft Azure provider): Billing is calculated on a per-minute basis, with the cost being the product of the minute rate of the resource and the actual usage duration. The calculation expression for C_{Ma} is obtained by

$$C_{Ma} = \sum_{i=1}^{n_{Ma}} \left\{ \left\lceil \frac{T_{usage}(vm_i)}{T_m} \right\rceil \times P_{minute}(vm_i) \right\}. \tag{14}$$

where $T_m = 60$, n_{Ma} is the number of instances provided by Microsoft Azure, and $P_{minute}(vm_i)$ is the per-minute price for the ith instance.

(3) Hybrid Billing C_{Gce} (Google compute engine provider): A fixed fee is charged for the first T_{fixed} minutes, and any additional usage is billed a per-minute basis. The calculation expression of C_{Gce} can be presented as [28]

$$C_{Gce} = \sum_{i=1}^{n_{Gce}} \left\{ P_{fixed} + \max\left(0, \left\lceil \frac{T_{usage}(vm_i) - T_{fixed}}{T_m} \right\rceil \right) \times P_{ondemand} \right\}, \tag{15}$$

here, n_{Gce} denotes the number of instances provided by google compute engine, $P_{ondemand}$ represents the on-demand price of the ith instance, and T_{fixed} is set to 10 min [2].

Definition 10 (*Computational Cost (C_{cal})*)**.** The expenses incurred from resource rental for a specific workflow are categorized into three types based on the billing methods, resulting in the total computation cost, which is defined as follows:

$$C_{cal} = C_{Az} + C_{Ma} + C_{Gce}. \tag{16}$$

Definition 11 (*Communication Cost (C_{comm})*)**.** C_{comm} represents the additional cost for a specific workflow associated with cross-cloud data transfers, while data transfer within the same cloud is free of charge. It can be expressed as

$$C_{comm} = \begin{cases} \sum\limits_{t_i, t_j \subseteq \mathcal{N}} R_{cross} \times DT(t_i, t_j), & \text{acrosscloud;} \\ 0, & \text{otherwise,} \end{cases} \tag{17}$$

where R_{cross} denotes the communication cost rate between different providers.

3.5. Problem formulation

In multi-cloud environments, the workflow scheduling problem can be formally defined as a quadruple $S = (\mathcal{R}, \mathcal{M}, TEC, TET)$, where $\mathcal{R}$ denotes the collection of all available cloud resources, $\mathcal{M}$ specifies the mapping of tasks to VMs, TEC represents the total execution cost (including the expenses for utilizing computing resources and data transmission), and TET signifies the overall completion time of the workflow. Specifically, the mapping comprises multiple tuples $(t_i, vm_z, AST(t_i, vm_z), AFT(t_i, vm_z))$, where t_i represents a task within the workflow and vm_z is the VM assigned for execution. The primary goal of this study is to minimize the total execution cost TEC of the specific workflow while adhering to the deadline constraint $TET \leq D$. This constrained optimization problem can be formulated as follows:

$$Minimize: \quad TEC = C_{Cal} + C_{comm}, \tag{18}$$

$$subject\ to: \quad TET \leq D. \tag{19}$$

4. RFPSO algorithm

The RFPSO algorithm integrates the predictive strengths of RF with the search capabilities of PSO, which provides a comprehensive enhancement solution for optimization problems that covers the entire process from initialization to search. The framework consists of five components: priority queue-based topological sorting, particle encoding, RF initialization, adaptive parameter adjustment, and local search enhancement. The specific implementation of the RFPSO algorithm is detailed in Algorithm 1, while its workflow scheduling process is illustrated in Fig. 2. This section systematically elaborates the principles of the RFPSO algorithm and concludes with an analysis of its time complexity.

Algorithm 1 RFPSO Algorithm

Input: workflow $\mathcal{W}$, resource pool $\mathcal{R}$, deadline D, number of iterations g.
Output: X_{best}.

```
1: Construct VM mapping M(W, R);
2: Select the fastest resource using sort VMs in descending order based on the cp attribute;
3: Compute execution time for each task using Eq. (3);
4: Call Algorithm 2 to optimize scheduling for Topological sequence O;
5: for each task t_i in O do
6:    Compute the actual start time of t_i using Eq. (7);
7:    Compute the actual time of t_i using Eq. (9);
8:    Update resource availability using Eq. (10);
9: end for
10: Train Random Forest model RF using Algorithm 3 with input M;
11: Initialize swarm S_0 ← null;
12: Define each particle as a position vector with length equal to the number of tasks, where each element records the VM index assigned to the corresponding task;
13: for i ← 1 to g do
14:    Initialize the i-th particle-vector;
15:    for t_i in N do
16:       Score all candidae VMs via the trained RF model, and select the one with the optimal predicted score;
17:       X* ← FE = [f_task, f_res];
18:       Write X* into the corresponding position of the particle vector;
19:       S_0 ← X*;
20:    end for
21:    Add the completely-constructed particle into the initial swarm S_0;
22: end for
23: Generate diversity-enhancing particles and add them into S_0 to form the complete initial population;
24: for k ← 1 to g do
25:    Execute adaptive particle swarm optimization using Algorithm 4;
26: end for
27: return X_best.
```

The pseudocode of the RFPSO algorithm is provided in Algorithm 1. Line 1 parses the input workflow to extract the computational load and dependencies of each task, which forms the task set. Line 2 selects the nodes with the highest computational performance from the available resources. Subsequently, Line 3 iterates through each task, which involves estimating its execution time on the selected resources

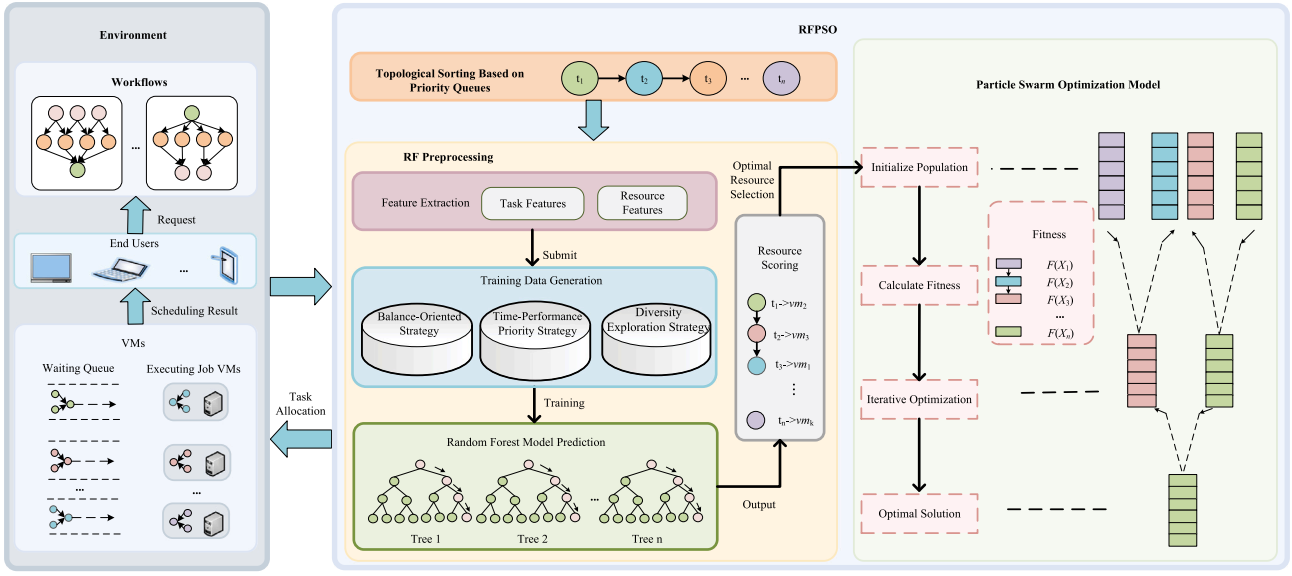

Fig. 2. Workflow scheduling process based on RFPSO.

using Eq. (3). Line 4 directly invokes Algorithm 2 to perform the topological sorting and generate the task scheduling sequence. These procedures ensure that all task dependencies are properly handled. Next, the process enters the scheduling phase (Lines 5–9), where it iterates through the sorted tasks and sequentially allocates resource. For each task, Line 6 calculates the earliest start time using Eq. (7), with the completion status of predecessor tasks and the current resource load being considered. Line 7 derives the actual completion time using Eq. (9), while Line 8 dynamically updates the resource available time according to Eq. (10) to ensure real-time status acquisition for subsequent scheduling. This cycle continues for each task until all tasks are scheduled. Line 10 executes Algorithm 3, which trains a random forest model using historical VM mapping records. This model predicts the optimal mapping relationships between tasks and resources. Line 11 initializes an empty particle swarm population. Line 12 clarifies the encoding definition of the particles, clearly explaining the structural meaning of a single particle. Lines 13–22 construct the resource allocation scheme for each particle through iterative processing. Line 14 completes the initialization of a single particle. Specifically, for each task within the particle (Lines 15–20), Lines 16–18 fully present the conversion logic from RF prediction results to VM allocation, where the aforementioned model is invoked to recommend the optimal resource (Line 17). This allocation relationship is then written into the position vector of the particle, and the complete allocation scheme of the particle is added to the initial population (Line 21). Line 23 outlines the complete composition of the initial population, clarifying the formation method of the population. Finally, Line 25 executes Algorithm 4 to obtain the optimal solution through the adaptive PSO (APSO) algorithm, and Line 27 outputs the optimal scheduling cost value.

4.1. Topological sorting based on priority queues

In workflow scheduling systems, the sequence of task execution plays a vital role in determining overall performance. To address this challenge, this study introduces the topological order generation with priority queue (TOGPQ) algorithm. The TOGPQ algorithm dynamically assigns priorities to tasks by strategically balancing computational complexity and critical path positioning. All these processes strictly adhere to task dependencies, which generates a globally optimized scheduling sequence.

Algorithm 2 outlines the specific workflow of the priority queue-based topological ordering generation algorithm. Line 1 initializes a

Algorithm 2 TOGPQ Algorithm

Input: workflow $\mathcal{W}$, task in-degree table $de(t_i)$, task priority $rank$.
Output: Topological sequence $\mathcal{O}$.

1: Initialize a priority queue Q prioritized by descending $rank$;
2: Initialize queue $Q \leftarrow \{t_i \in \mathcal{N} \mid de(t_i) = 0\}$;
3: Initialize $\mathcal{O} \leftarrow null$, $rank \leftarrow null$;
4: Compute task priority $rank$ using Eq. (2);
5: **for** each task t_i in N **do**
6: **if** $de(t_i) = 0$ **then**
7: Enqueue t_i into Q with priority $rank$;
8: **end if**
9: **end for**
10: **while** Q is not empty **do**
11: Extract the task t_j at the head of queue Q, remove it from queue Q;
12: Add t_i to the end of topological sequence $\mathcal{O}$;
13: **for** each t_j in $succ(t_i)$ **do**
14: $de(t_j) \leftarrow de(t_j) - 1$;
15: **if** $de(t_i) = 0$ **then**
16: Enqueue t_j into Q with priority $rank$;
17: **end if**
18: **end for**
19: **end while**
20: **return** $\mathcal{O}$.

priority queue for tasks, which involves sorting in descending order of task priority. This design guarantees that the highest-priority tasks are processed first during scheduling. Line 2 adds all tasks with zero in-degree to the queue, as these independent tasks represent the initial candidate set for scheduling. Line 3 creates an empty sequence to store the topological sorting results. Line 4 iterates through all tasks to compute their respective priorities. After scanning the task set (Lines 5–9), tasks with zero in-degree are added to the queue according to their priorities, which constructs the initial candidate set (Lines 6–8). Lines 10–18 form the loop body of the algorithm, which is responsible for sorting task priorities. In each iteration, Line 11 removes the highest-priority task from the queue, Line 12 immediately adds it to the output sequence and marks it as scheduled. Subsequently, all successor tasks of the task are handled (Lines 13–18). For each successor, Line 14 decrements its in-degree and updates the dependency relationships.

When the in-degree of a task is detected to be zero, it is added to the priority queue according to its priority, which ensures that high-priority tasks are incorporated into the scheduling process promptly (Lines 15–17). The main loop terminates at Line 19 when the queue is empty, which indicates that all tasks have satisfied the dependency and priority constraints. Finally, Line 20 outputs the topological sequence of the tasks.

4.2. Particle encoding and solution space

Particles employ integer encoding, where each particle $x = (x_1, x_2, \ldots, x_n) \in X$ represents a scheduling scheme. Here, $x_i \in \{1, 2, \ldots, V\}$ denotes the index of the VM assigned to task t_i. The solution space $X = \{1, 2, \ldots, V\}^n$ has a dimensionality n equal to the number of tasks, with the range of each dimension corresponding to the total number V of available VMs.

4.3. Random forest-assisted initialization

During the particle initialization phase, a random forest model is utilized to directionally optimize particle allocation. To generate training samples, a hybrid approach that incorporates three complementary strategies is implemented. Each strategy targets distinct optimization objectives, which ensures thorough coverage of the sample space across three dimensions: cost optimization, time performance optimization, and exploratory search. The specific strategies are structured as follows:

(1) Balance-Oriented Strategy: This strategy aims to achieve a trade-off optimization between task criticality and resource economy, which is designed based on the pareto equilibrium principle in multi-objective decision-making [35]. First, a threshold is set at $\theta = 0.7$, and tasks satisfying $\rho(t_i) > \theta$ are identified as critical tasks. Next, multi-dimensional screening is performed on VM resources, which involves selecting instances to form a high-performance resource pool a low-cost resource pool. Finally, the allocation criterion is that, if a task is a critical task, then a VM instance is randomly selected from the high-performance resource pool; otherwise, it is randomly selected from the low-cost resource pool. The criticality is determined based on the topological level of the task, and the hierarchical weight $\rho(t_i)$ of task t_i is calculated as follows:

$$\rho(t_i) = l(t_i)/l_{max}, \tag{20}$$

where $l(t_i)$ is the level of task t_i, and l_{max} signifies the maximum level value.

(2) Time-Performance Priority Strategy: This strategy focuses on optimizing workflow completion time by following the "critical path first" principle from the HEFT algorithm. It assigns all tasks to high-performance VMs to reduce the critical path execution time. Particle generation involves randomly selecting a VM instance from the high-performance resource pool for each task. This strategy provides the model with samples that demonstrate superior temporal characteristics, which improves the scheduling capabilities of the algorithm when faced with tight deadline constraints.

(3) Diversity Exploration Strategy: To maintain population diversity and avoid premature convergence, a fully random allocation mechanism is designed as a safeguard for exploration [36]. For each task, a VM instance is randomly chosen from the complete set of available VM instances. This strategy introduces perturbations in the search space, which involves exploring potentially high-quality solutions that are not immediately apparent.

Before training the RF model, all feature vectors are standardized by subtracting the mean and dividing by the standard deviation, ensuring that each feature has a mean of zero and a variance of one. At the feature construction level, task requirements and resource characteristics need to be simultaneously considered. Specifically, the feature vector FE, which consists of task and resource features, can be expressed as

$$FE = \left[f_{task}, f_{res} \right], \tag{21}$$

here, the task feature vector f_{task} comprises the following elements: task computational load, task level in the DAG, task in-degree and out-degree, and critical path identifier. The resource feature vector f_{res} includes the computational capacity cp, unit time price, and unit computational cost.

According to the optimized scoring mechanism that combines cost and timeout penalty, the model can be guided to learn low-cost and timeout-free matching patterns. This way achieves the training objective of the model. The formula for the optimized scoring mechanism is

$$SC = \varpi \cdot TEC + \lambda \cdot \left(\max\{0, TET - D\} \right)^2, \tag{22}$$

where the cost weight ϖ is set higher than the timeout penalty weight, which emphasizes the optimization of economic costs. The squared penalty term increases sensitivity to timeout samples, with λ representing the weight of the timeout penalty. λ is set to 800, which is a comparatively high value, to ensure that the timeout penalty constitutes a substantial portion of the fitness score. In this way, the algorithm is encouraged to prioritize the avoidance of timeouts.

The prediction output of the random forest RF can be derived by averaging the predictions of the decision trees. The expression for the random forest prediction model $RF(FE)$ is estimated by

$$RF(FE) = \frac{1}{b} \sum_{i=1}^{b} m_i(FE), \tag{23}$$

where m_b represents the bth decision tree. This predictive capability can lower the risk of overfitting and improve generalization performance.

Using historical data and a random forest model, each training sample comprises a feature vector of the task paired with a label representing the optimal VM allocation under cost and deadline constraints, allowing for a quantitative estimation of the cost associated with the "task–VM" matching scheduling scheme. By leveraging the predictive capability of the model, the scheme with the lowest score is selected. This way allows for high-quality decision-making $DS_{i,j}$ at the beginning of the algorithm and enhances the overall efficiency and effectiveness of the optimization process. The calculation of $DS_{i,j}$ is expressed as

$$DS_{i,j} = \min\left\{ RF\left(FE_{i,j}\right) \right\}, \tag{24}$$

where $FE_{i,j}$ represents the feature vector of the ith task corresponding to the jth particle.

Algorithm 3 RFRAT

Input: workflow $\mathcal{W}$, number of trees b.
Output: high-quality decision DS.
1: Initialize feature matrix $FE \leftarrow null$, target vector $Y \leftarrow null$;
2: **for** each task t_i in $\mathcal{N}$ **do**
3: Extract task features $f_{task}(t_i)$;
4: Extract resource features $f_{res}(vm_z)$;
5: Extract features $f_{task}(t_i)$ and $f_{res}(vm_z)$, and append them to FE;
6: Compute optimization score SC using Eq. (22);
7: Y.append(SC);
8: **end for**
9: Select the particle with the lowest cost from Y;
10: **for** each j in b **do**
11: Train random forest $RF(FE)$ using Eq. (23);
12: $DS \leftarrow \min\left(RF(FE)\right)$ using Eq. (24);
13: **end for**
14: **return** DS.

The specific workflow of the random forest resource allocation training algorithm (RFRAT) is detailed as follows. Line 1 of the RFRAT algorithm initializes an empty feature matrix and target vector to record the training samples that will be generated later. Lines 2 to 8

perform the following operations: Line 3 extracts the task feature information, while Line 5 extracts the corresponding resource attributes. Line 5 concatenates the two sets of features to create the task feature vector. Line 6 evaluates the task-resource allocation scheme using a scoring mechanism. Line 7 calculates the optimization score for the allocation scheme, and this score is stored in the target vector to build the training sample set. Line 9 selects the particle with the lowest cost. Line 11 then trains a random forest regression model based on normalized features and target values. Line 12 develops a high-quality scheduling decision scheme. Line 14 outputs the trained model for subsequent automated resource allocation predictions.

4.4. Population diversity quantification

The population diversity δ is defined as the ratio of the average Euclidean distance from all particles to the centroid to the dimensionality of the solution space, and it can be expressed as

$$\delta = \frac{1}{N} \cdot \frac{1}{|P|} \sum_{i=1}^{|P|} \left\| x_i - \bar{X} \right\|, \tag{25}$$

here, the number of particles $|P|$ is a dynamic parameter adaptively determined based on the task scale. x_i represents the value of the ith particle, N is the number of tasks, $\bar{X}$ is the population centroid, and $\| \cdot \|$ denotes the Euclidean norm. A larger δ value indicates higher population dispersion and stronger exploration capability. A smaller δ value suggests that the population is more concentrated, with a more obvious convergence trend.

4.5. Adaptive parameter update strategy

The inertia weight $\omega(k)$ regulates the capacity of a particle to maintain its historical velocity, and a nonlinear decay strategy is employed to facilitate a shift from global exploration to local exploitation. $\omega(k)$ can be calculated as follows:

$$\omega(k) = \omega_0 \cdot \xi^{k/k_{\max}}, \tag{26}$$

where k represents the current iteration count, and $k_{\max}$ indicates the maximum number of iterations, which is determined through dynamic adjustment. The initial value of the inertia weight ω_0 is set to 0.9, while the empirically determined decay coefficient ξ is set to 0.3. This configuration provides particles with strong global exploration capabilities in the early stages. These properties allow for a rapid nonlinear decay strategy that facilitates a swift transition from global exploration to local exploitation.

The range of the original diversity value δ varies with the task size N and the number of VMs V, and using it directly for learning factor adjustments can lead to unstable parameter updates. To address this, the diversity is normalized in this study. The maximum diversity value that the population can achieve is $\delta_{\max}$. Thus, the normalized diversity $\delta'(k)$ is defined as [36]:

$$\delta_{\max} = \frac{V-1}{2\sqrt{N}}, \tag{27}$$

$$\delta'(k) = \frac{\delta(k)}{\delta_{\max}}, \tag{28}$$

where the range of $\delta'(k)$ is [0,1].

Accordingly, the individual cognitive factor c_1 and the social learning factor c_2 are dynamically adjusted based on the normalized diversity $\delta'(k)$, with the calculation expressions given by

$$c_1(k) = c_{1,\min} + (c_{1,\max} - c_{1,\min}) \cdot (1 - \delta'(k)), \tag{29}$$

$$c_2(k) = c_{2,\min} + (c_{2,\max} - c_{2,\min}) \cdot \delta'(k), \tag{30}$$

where the dynamic adjustment of c_1 and c_2 ensures the stability of the algorithm's optimization results.

4.6. Position update and local enhancement strategy

By integrating the velocity-position model of classical PSO with a heuristic local search, an efficient approach for discovering optimal solutions is established through the synergy of global guidance and local refinement. The particle velocity update expression can be defined as [37]

$$v_{i,j}^{k+1} = \omega v_{i,j}^{k} + c_1 r_1 \left(q_{i,j} - x_{i,j}^{k}\right) + c_2 r_2 \left(g_j - x_{i,j}^{k}\right), \tag{31}$$

where $v_{i,j}^{k}$ refers the velocity of particle k in dimension j at the ith iteration. $q_{i,j}$ denotes the individual optimal position of the particle, while g_j represents the global optimal position in dimension j for the population, which corresponds to the dimension of the task. $r_1, r_2 \in (0, 1)$ is a random disturbance factor that increases search randomness.

Using the current position and updated velocity of the particle, its position at the next time step is calculated to facilitate the search for the optimal solution. The update of the position of the particle can be represented as

$$x_{i,j}^{k+1} = x_{i,j}^{k} + v_{i,j}^{k+1}, \tag{32}$$

where $x_{i,j}^{k}$ represents the position of the ith particle in the jth dimension at the kth iteration, which corresponds to the VM index assigned to the jth task for the ith particle. $v_{i,j}^{k+1}$ denotes the velocity of particle i at the $(k+1)$th iteration. $x_{i,j}^{k+1}$ is the new position reached by particle i after the $(k+1)$th iteration.

We employ a reflective boundary strategy to handle boundary overflow situations for $x_{i,j} \notin [0, V-1]$, where V represents the total number of available VMs, and the particle position $x_{i,j}$ indicates the VM index assigned to the jth task. The latest particle position $x_{i,j}^{*,k+1}$ calculation expression is as follows:

$$x_{i,j}^{*,k+1} = \begin{cases} x_{i,j}^{k+1}, & 0 < x_{i,j}^{k+1} < V; \\ 2(V-1) - x_{i,j}^{k+1}, & x_{i,j}^{k+1} \geq V; \\ -x_{i,j}^{k+1}, & \text{otherwise.} \end{cases} \tag{33}$$

By applying velocity attenuation correction to overflowing particles, repeated oscillations at the boundaries are prevented, which ensures search effectiveness. The update of $v_{i,j}^{*,k+1}$ proceeds as follows:

$$v_{i,j}^{*,k+1} = \mu \cdot v_{i,j}^{k+1}, \tag{34}$$

where μ is the attenuation coefficient, which is set at 0.3. This operation significantly reduces the particle velocity, which effectively dissipates its kinetic energy upon impacting the boundary and suppresses repeated oscillations.

4.7. Fitness function design

The time performance weight ∂ and cost weight β for tasks can be dynamically adjusted according to their deadline urgency, and they are calculated as follows:

$$\partial = TET - D, \tag{35}$$

$$\beta = 1 - \partial. \tag{36}$$

This design ensures that, when the deadline is tight (D close to TET), the time performance weight ∂ increases. This condition prompts the RFPSO algorithm to prioritize meeting time constraints. When the deadline is more lenient, the cost weight β increases, which allows RFPSO to focus more on cost optimization.

Based on the linear weighted sum method, the comprehensive fitness $\mathcal{H}(X)$ for task scheduling is derived from the weighted sum of time and cost objectives. To eliminate the influence of dimensional differences between time and cost on the optimization results, each objective component is normalized separately. $\mathcal{H}(X)$ can be evaluated as

$$\mathcal{H}(X) = \beta \cdot TEC + \partial \cdot TET. \tag{37}$$

For individuals violating the deadline constraint, a penalty term $FP(X)$ is applied, where the penalty severity is positively correlated with the magnitude of the constraint violation. $FP(X)$ is defined as follows:

$$FP(X) = \max\left\{0, TET - D\right\} \cdot \psi, \tag{38}$$

where ψ is the penalty coefficient.

Based on the sum of the comprehensive fitness $\mathcal{H}(x)$ and the individual penalty $FP(X)$, the final fitness function is obtained as

$$F(X) = \mathcal{H}(X) + FP(X). \tag{39}$$

To further improve the quality of the global optimal solution and effectively guide the search process to escape local optima, we incorporate a local neighborhood search operator into the APSO algorithm. This operator includes modules for the neighborhood structure, movement operations, acceptance criteria, and triggering mechanisms.

Neighborhood Structure. The neighboring solution XN and X differ by only one dimension, meaning that exactly one task has been reassigned to another VM, while strictly adhering to all dependency constraints of the tasks.

Movement Operations. The generation of neighboring solutions employs the following two strategies to enhance search efficiency and optimize the scheduling scheme.

(1) Random Reallocation Strategy: A task is randomly selected from the current scheduling scheme and migrated from its currently assigned VM to a different VM, while keeping the allocations of other tasks unchanged. This strategy aims to increase the diversity of solutions and facilitate the exploration of the search space.

(2) Critical Path Priority Reallocation Strategy: By identifying the set of tasks on the critical path, this strategy prioritizes the reallocation of these tasks to further optimize the scheduling efficiency of the workflow. The identification of critical path tasks is based on the priority values of the tasks, and this strategy effectively reduces the total completion time of the workflow, enhancing overall scheduling performance.

Acceptance Criteria. A deterministic greedy acceptance criterion is employed. If the fitness of the neighborhood optimal solution XN_{best} is better than that of the current global optimal solution X^*, then the global optimum is updated as X^*_{new}. The expression is given as follows:

$$X^*_{new} = \begin{cases} XN_{best}, & \text{if } F(XN_{best}) < F(X^*); \\ X^*, & \text{otherwise.} \end{cases} \tag{40}$$

Triggering Mechanism. The local neighborhood search is triggered at the end of each iteration of the APSO to further optimize the current global optimal solution X^*. Specifically, the APSO algorithm tries assigning each task in the current solution to every available VM sequentially, while keeping track of the best neighborhood solutions found during this process. If a solution better than the current one is found, the global optimal solution X^*_{new} is updated. This mechanism aims to enhance the quality of solutions and avoid getting stuck in local optima by thoroughly exploring the neighborhood space of task assignments.

The pseudocode of the APSO algorithm is shown in Algorithm 4. Line 1 parses the input DAG workflow to extract the computational load and dependencies for each task, which forms the task set. In each iteration, Line 1 computes the diversity of the current particle swarm to assess population distribution. The inertia weight, individual cognitive factor, and social learning factor are dynamically adjusted based on the swarm state to balance the global search and local exploitation capabilities of the algorithm (Lines 2–4), while iterating through all dimensions of each particle (Lines 6–18). Specifically, Line 5 initializes the particle's individual best and the population's global best, establishing a benchmark for subsequent iterative optimization and refining the algorithm's initial logic. Line 7 incorporates the parameter recalculation logic for each generation's iteration to ensure that the adaptive parameters are updated in real time based on the population's state, thereby accurately reflecting the algorithm's adaptive characteristics. Line 10 updates the particle velocity, and Line 11 updates the particle position to continuously explore better solutions. Next, boundary constraint handling is applied to ensure that the allocation scheme represented by the particles complies with the constraint conditions (Line 13). The fitness function is used to evaluate the quality of each allocation scheme, update the individual historical optimal solution, and revise the global optimal solution (Lines 14). Line 16 specifies the iteration rules for the individual optimal solution, completing the core iterative process of standard particle swarm optimization and ensuring that each particle preserves its historical best information. Line 17 updates the optimal solution, and Line 20 reiterates the optimal solution obtained.

Algorithm 4 APSO Algorithm

Input: workflow $\mathcal{W}$, resource pool $\mathcal{R}$, deadline D, number of iterations g, population size P.
Output: X^*_{new}.

```
1: Compute diversity δ using Eq. (25);
2: Compute inertia weight w using Eq. (26);
3: Compute cognitive factor c_{1,t} using Eq. (29);
4: Compute social factor c_{2,t} using Eq. (30);
5: Initialize personal best for each particle and global best with initial
   swarm fitness values;
6: for each iterations k in g do
7:    Recompute population diversity and adaptive parameters for the
      current generation;
8:    for each particle i in P do
9:       for j ← 1 to n do
10:         Update velocity v_{i,j}^{k+1} Eq. (31);
11:         Update position x_{i,j}^{k+1} Eq. (32);
12:      end for
13:   end for
14:   Apply boundary constraints to all particles position using Eq.
      (33);
15:   Evaluate fitness F(x_i^{k+1}) for all i using Eq. (39);
16:   Update personal best for each particle by comparing current
      fitness with its historical optimal value;
17:   Update Global Optimum X*_{new} using Eq. (40);
18: end for
19: return X*_{new}.
```

4.8. Time complexity of RFPSO

The time complexity of the RFPSO algorithm mainly depends on the maximum number of iterations g, the population size P, the total number of tasks N, the number of dependency edges e, the number of VM types V, and the number of decision trees b. The initialization module performs VM performance ranking, execution time precomputation, and initial population generation, with a time complexity of $O(V \log V + NV + P \cdot N \cdot V \cdot b)$. Specifically, $V \log V$ is the time required for sorting the VMs, NV is the complexity of the execution time precomputation for tasks, and $P \cdot N \cdot V \cdot b$ is the time overhead for generating the population. The topological sorting module is shown in Algorithm 2, which traverses all tasks and dependency edges using a priority queue, with a time complexity of $O(N + e)$. The random forest training module is illustrated in Algorithm 3, which extracts task and resource features to construct training samples and trains a regression model composed of b decision trees, with a time complexity of $O(b \cdot NV \cdot \log(NV))$. The adaptive parameter update module is shown in Algorithm 4, which calculates population diversity and dynamically adjusts parameters in each iteration, with a time complexity of $O(P \cdot N)$.

Table 2
Microsoft Azure [39].

VM type	Per minute	$cp(vm)$
B2MS	0.0015	2
B4MS	0.003	4
B8MS	0.006	8
B16MS	0.012	10

Table 3
Amazon EC2 [2,40].

VM type	Per hour	$cp(vm)$
m1.small	0.06	2
m1.medium	0.12	4
m1.large	0.24	8
m1.xlarge	0.45	10

Table 4
Google compute engine [28,41].

VM type	Ten minutes	Per minute	$cp(vm)$
n1-highcpu-2	0.014	0.0012	2
n1-highcpu-4	0.025	0.0023	4
n1-highcpu-8	0.05	0.0047	8
n1-highcpu-10	0.1	0.0093	10

Table 5
Parameter settings [37,45].

Parameters	Value
Population size P	100
Maximum number of iterations g	50
Inertia weight w	[0.3, 0.95]
Cognitive factor c_1	[0.5, 2.5]
Social learning factor c_2	[0.5, 2.5]
r_1, r_2	[0, 1]
Number of tree b	50

The particle update module, also illustrated in Algorithm 4, updates the velocity and position of all particles and corrects any out-of-bounds particles, with a time complexity of $O(P \cdot N)$. The fitness evaluation module calculates the actual start times, completion times, and cross-cloud communication costs for all tasks based on topological sorting, with a time complexity of $O(P \cdot (N + e))$. The local search module performs neighborhood search on the global optimal solution at the end of each iteration, with a time complexity of $O(V \cdot N \cdot (N + e))$. Since the algorithm performs g iterations, the total complexities of each module are $O(g \cdot P \cdot N)$, $O(g \cdot P \cdot (N + e))$, and $O(g \cdot V \cdot N \cdot (N + e))$, respectively. In addition, the time complexity for comparing and updating the optimal individuals is $O(g \cdot P)$. Therefore, the total time complexity of the RFPSO algorithm is: $O(N + e + V \log V + NV + b \cdot NV \cdot \log(NV) + P \cdot N \cdot V \cdot b + g \cdot P \cdot (N + e) + g \cdot V \cdot N \cdot (N + e))$, namely $O(g \cdot V \cdot N \cdot (N + e))$.

5. Experiments and analysis

This section establishes the experimental environment and comparison schemes. It provides details on the design and configuration of the experiments, baseline algorithms, performance evaluation metrics, and a comprehensive comparative experimental design.

5.1. Experimental design and environment configuration

The experiments are programmed in Python, with the simulated experimental environment built using CloudSim [38]. The simulation runs on a hardware platform equipped with an Intel Core i7-12700K processor, 256 GB of RAM, and a 1 TB hard disk, operating on Ubuntu 22.04. The simulation environment includes a heterogeneous multi-cloud resource pool composed of Microsoft Azure, Amazon EC2, and Google compute engine cloud service providers. The pricing and billing models of each cloud service provider are based on publicly available data from mainstream cloud service providers [39–41], which cover hourly billing, minute-based billing, and data transfer costs, among other factors. By utilizing various resource configuration combinations, the experimental environment effectively captures the real characteristics of resource heterogeneity and billing diversity present in multi-cloud environments. Specific data for each service provider are detailed in Tables 2 to 4. The startup time for each VM is set to 97 s [42]. In addition, the parameter tuning strategy and the RF hyper-parameter settings used in the experiments are shown in Table 5. the average bandwidth within the same provider is set to B_{inter} = 20, while the average bandwidth between different providers is set to B_{intra} = 100, and the communication cost rate R_{cross} between different providers is set to 0.0001 [28]. To comprehensively evaluate the applicability and generalization performance of the proposed algorithm, this study selects four representative real workflow datasets from the field of scientific computing, including CyberShake, Epigenomics, LIGO, and Montage. The structures of the four types of real workflows are illustrated in Fig. 3, and each workflow type is configured with four task quantity scales (N): 30, 50, 100, and 1000 tasks. Moreover, the DAG structures of all workflows are generated using the Pegasus workflow generator [43,44]. To examine the scheduling performance of the algorithm under different time constraints, a progressive set of deadline coefficients $\alpha = \{0.05, 0.1, 0.15, 0.2, 0.25\}$ is designed.

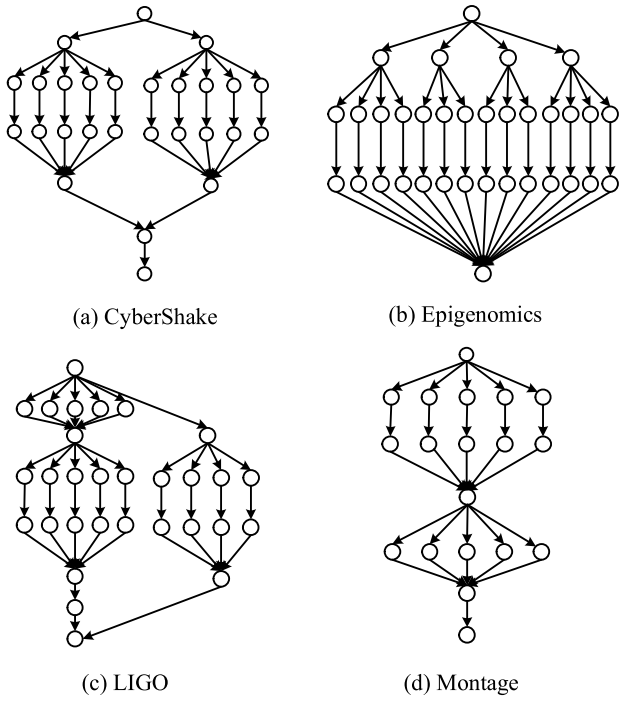

Fig. 3. Structure of four realistic scientific workflows.

5.2. Selection of baseline algorithms

The experiment selects the simulated annealing deep Q-network (SADQN [46]), GWOEM [24], IFA [23], and traditional PSO [27] as baseline algorithms for comparative evaluation to assess the scheduling performance of the proposed algorithm. The main principles of each algorithm are explained as follows:

(1) SADQN Algorithm: This algorithm is designed for cloud real-time workflow scheduling and combines simulated annealing with deep

Q-networks. It utilizes simulated annealing to solve the optimal execution sequence of subtasks and extracts workflow temporal features as input for the deep Q-network. The deep Q-network autonomously learns dynamic scheduling strategies, which can reduce workflow response time, improve scheduling success rates, and decrease resource usage costs.

(2) GWOEM Algorithm: This scheduling algorithm is based on gray wolf optimization and employs a one-dimensional encoding mechanism, where each individual corresponds to a task-to-VM mapping. Multiple individuals collaboratively form a complete scheduling solution. By combining a greedy replacement operator with the gray wolf predation operator, the algorithm iteratively updates individual positions to efficiently search for the optimal solution.

(3) IFA Algorithm: This algorithm adopts a two-stage scheduling strategy. In the first stage, it utilizes the safe service selection theorem to assign appropriate safe services to each task, which involves prioritizing high-security-level services for tasks with smaller data volumes. This approach aims to meet the security thresholds and encryption time constraints of the system simultaneously. In the second stage, an IFA is applied, which maps positions in continuous space to discrete task sequences using a distance mapping operator. Combined with an optimized position update method, this algorithm uses a distance-related random search range to enhance task scheduling.

(4) PSO Algorithm: This scheduling algorithm is based on PSO, where particle positions are encoded as task-to-VM mappings. The algorithm iteratively updates velocity and position to facilitate the search for the optimal solution. Total cost serves as the fitness evaluation metric, with a focus on solutions that meet the deadlines. Dynamic parameter adjustment is employed to strike a balance between global and local search.

5.3. Performance comparison metrics

The experiment employs execution cost, resource utilization (RU, %), success rate (SR, %), and average cost reduction percentage ($ACRP$, %) to evaluate the performance of the proposed RFPSO algorithm.

(1) Execution Cost: This cost refers to the total cost incurred during the workflow scheduling process, including computational resource usage and cross-cloud data transmission costs. It serves as a crucial metric for evaluating the cost optimization capabilities of the algorithm.

(2) Resource Utilization: This measures the efficiency of a scheduling scheme in utilizing VM resources, defined as the ratio of the total computational load of all tasks to the total computational time resources provided by all utilized VMs during the entire workflow execution.

(3) Success Rate: This metric represents the ratio of the number of scheduling solutions (ns) that meet the workflow deadline constraint to the total number of scheduling attempts (nt). It is used to evaluate the reliability of the algorithm in adhering to deadline constraints. The SR can be expressed as

$$SR = \frac{ns}{nt}. \tag{41}$$

The SR metric directly indicates whether the scheduling solutions produced by the algorithm can complete workflow execution within the specified time. A higher SR indicates a stronger capability of the algorithm to meet time constraints, which makes it particularly suitable for latency-sensitive workflow scenarios.

(4) Average Cost Reduction Percentage: This metric quantifies the cost advantage of RFPSO in comparison to baseline algorithms such as GWOEM, IFA, and traditional PSO. It is calculated as the percentage of the cost difference between RFPSO and the baseline algorithm relative to the cost of the baseline algorithm [47]:

$$ACRP(\text{RFPSO}) = \left[\frac{(TEC(U) - TEC(\text{RFPSO}))}{TEC(U)}\right] \times 100\%, \tag{42}$$

here, U represents a specific baseline algorithm. This metric clearly demonstrates the extent of cost optimization improvement achieved by RFPSO. A higher value indicates more significant cost savings compared to the baseline algorithms.

5.4. Performance comparison experiments

This section assesses the performance of the algorithm in terms of cost control and deadline success rate. Comparative experiments are primarily designed based on dimensions such as workflow scale, the strictness of deadline constraints, and the number of VMs. The experiments involve running each algorithm independently 50 times, with the results analyzed using their average values. These experiments encompass the following aspects.

Experiment 1: The study systematically compares the cost control capabilities of the RFPSO algorithm against three scheduling algorithms SADQN, GWOEM, IFA, and PSO across various workflows, N, and α. Specifically, four types of workflows are selected, which correspond to node scales of 50, 100, and 1000, respectively. The parameter α is varied from 0.05 to 0.25 to comprehensively assess the execution cost performance of each algorithm. The experimental results are shown in Figs. 4–7.

Analysis of the CyberShake workflow. As shown in Fig. 4, under various conditions and N values, the boundary constraint strategy of RFPSO effectively addresses the issues of poor adaptability of static parameters and ineffective searches by introducing reflective boundaries to correct out-of-bound particles. This strategy significantly enhances the capability of RFPSO to expand the search space and accelerate global convergence. These conditions allow it to consistently achieve the lowest execution cost. Specifically, when the workflow node scale is 50, RFPSO maintains lower costs than SADQN, GWOEM, IFA, and PSO within the range of 0.05 to 0.25, as shown in Fig. 4(a). In contrast to SADQN, GWOEM, IFA, and PSO, RFPSO achieves average cost reductions of 57.66%, 64.48%, 79.75%, and 88.67%, respectively. When the number of nodes increases to 50, 100, and 1000, RFPSO continues to exhibit significant robustness and cost control capability.

Epigenomics Workflow Analysis. RFPSO demonstrates superior cost performance in experiments using the Epigenomics workflow, as depicted in Fig. 5. Across three different workflow node scales 50, 100, and 1000, RFPSO achieves the lowest cost across all α values, demonstrating that RFPSO consistently outperforms other algorithms in terms of cost minimization efficiency. Fig. 5(b)–(d) further illustrate that, as the node scale increases to 50, 100, and 1000, RFPSO continues to maintain the lowest cost across all α values with relatively low volatility. In summary, across various α values, RFPSO consistently achieves the lowest average scheduling cost compared with SADQN, GWOEM, IFA, and PSO. Similar to Fig. 4, the data in Fig. 5 show that RFPSO effectively minimizes resource costs by leveraging its exceptional global scheduling capabilities.

Montage Workflow Analysis. RFPSO also exhibits significant cost minimization performance under the Montage workflow, as illustrated in Fig. 6. In Fig. 6(a) and (b), when the number of nodes increases to 50 and 100, PSO shows low sensitivity to changes in α, with its cost barely decreasing as the deadline constraint is relaxed. IFA experiences significant cost fluctuations and demonstrates weaker adaptability to changes in α. SADQN and GWOEM maintain costs within a moderate to low range, exhibiting overall relative stability. Notably, SADQN outperforms GWOEM by achieving lower costs under most α values. By contrast, RFPSO showcases strong cost minimization control across the entire α range and demonstrates robust adaptability to varying task deadline requirements. As shown in Fig. 6, when faced with scenarios characterized by discrete task distributions and high demands for dynamic resource allocation, RFPSO can flexibly adjust its population structure to better align with global resource supply. Thus, it consistently maintains the lowest cost.

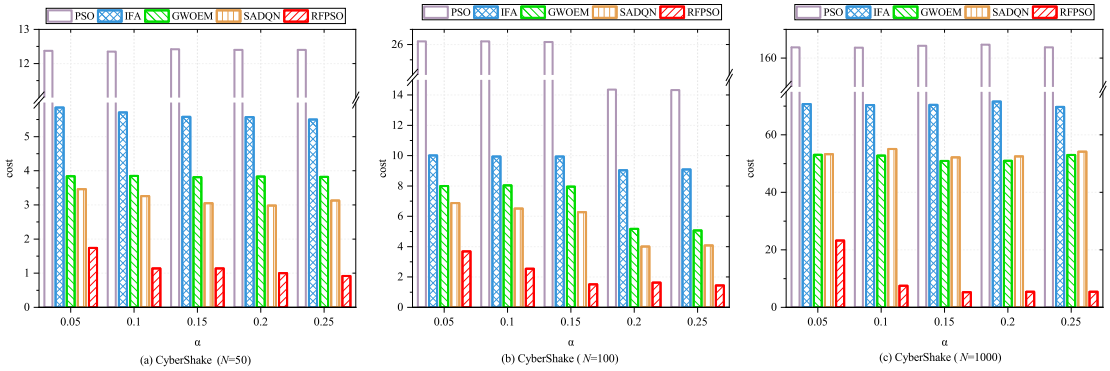

Fig. 4. Cost comparison of the CyberShake workflow with various deadline factors.

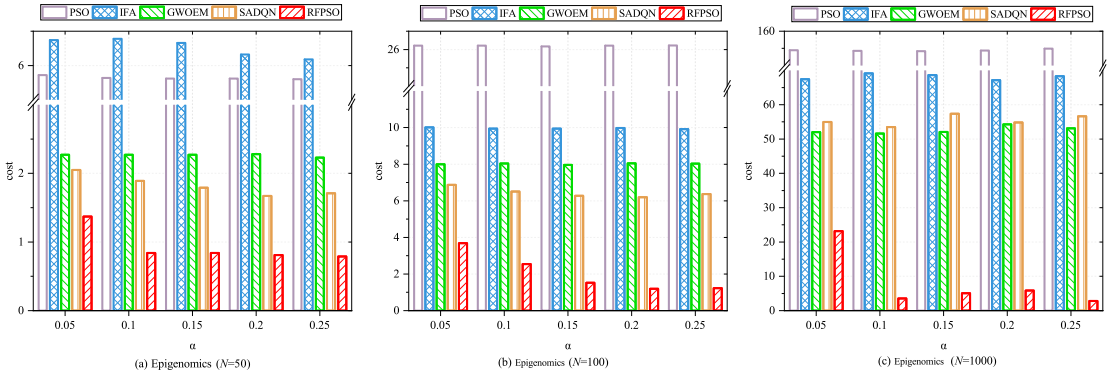

Fig. 5. Cost comparison of the Epigenomics workflow with various deadline factors.

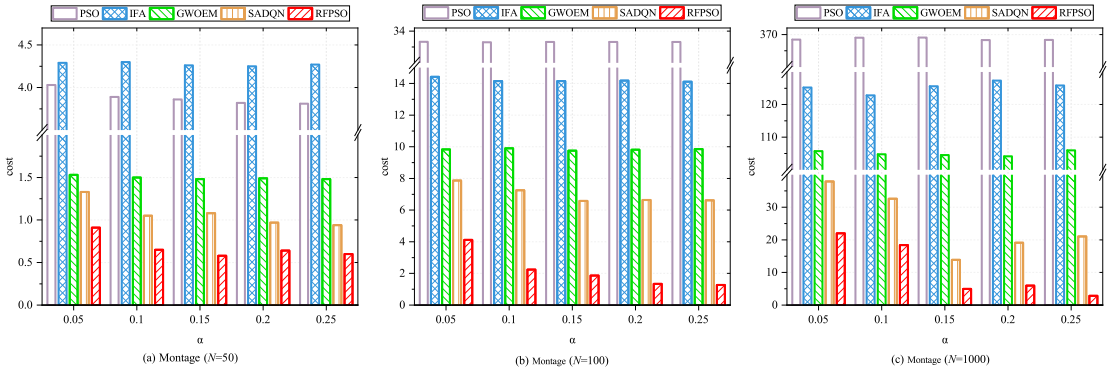

Fig. 6. Cost comparison of the Montage workflow with various deadline factors.

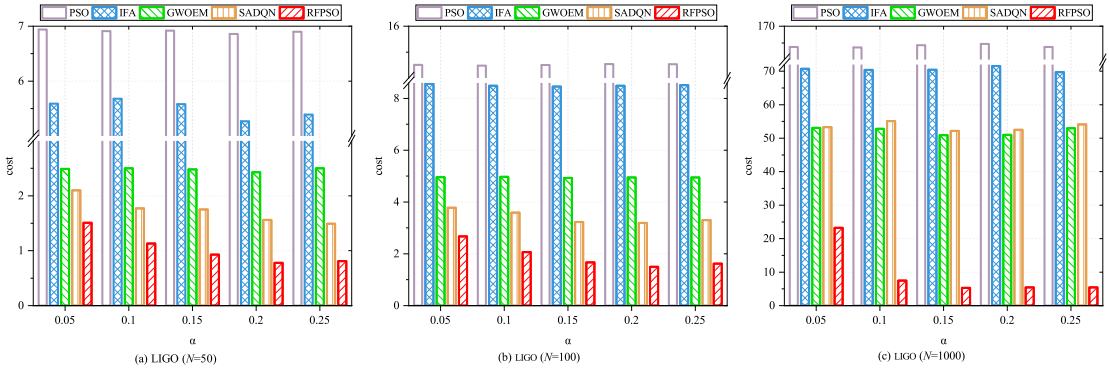

Fig. 7. Cost comparison of the LIGO workflow with various deadline factors.

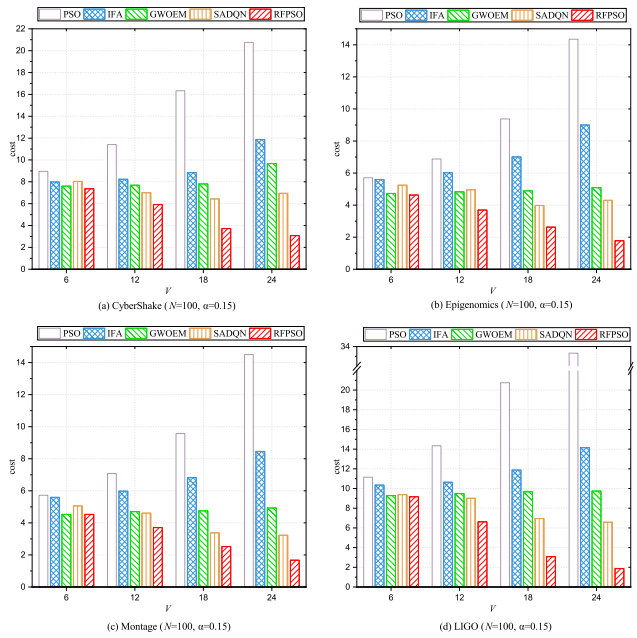

Fig. 8. Cost comparison across four workflows with different VM number.

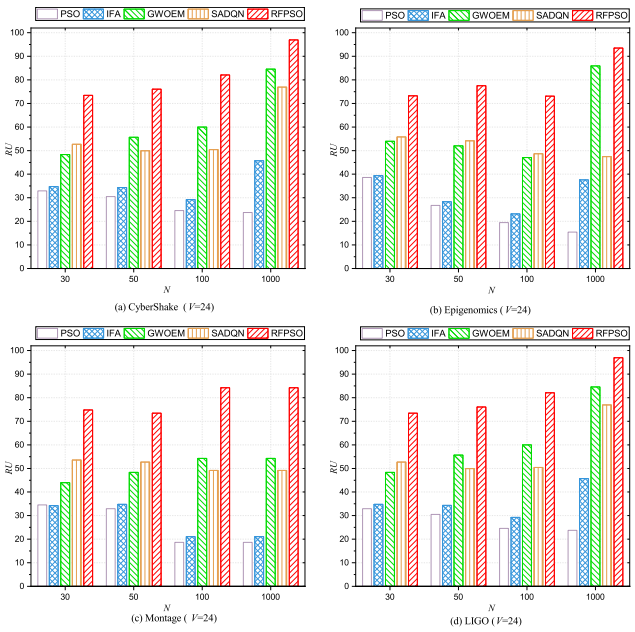

Fig. 9. *RU* (%) over four workflows with different numbers of nodes.

LIGO Workflow Analysis. As shown in Fig. 7, RFPSO achieves the lowest execution cost across all experiments with varying α values, regardless whether the node scale is 50, 100, or 1000. For example, Fig. 7(a) indicates that, at 50 nodes, the cost of RFPSO exhibits minimal variation. When compared with SADQN, GWOEM, IFA, and PSO, RFPSO reduces costs by 37.06%, 54.81%, 84.18%, and 82.59%, respectively. Even with a larger node scale, RFPSO continues to maintain the lowest cost. Fig. 7 reveals a trend similar to that observed in Fig. 6. The cost minimization realized by RFPSO is largely attributed to its hierarchical task allocation mechanism based on critical path slack. This mechanism prioritizes tasks on the critical path for higher-performance computing resources, which effectively reduces costs.

Experiment 2: A systematic comparative analysis is conducted to examine the impact of varying numbers of VMs on workflow scheduling costs. The experiment set the number of VMs within the range of 6 to 24, with α fixed at 0.15. On this basis, the execution costs of the RFPSO, SADQN, GWOEM, IFA, and PSO algorithms under different VM configurations are separately compared, and the relevant results are shown in Fig. 8.

Across different workflows, RFPSO achieves the lowest scheduling costs under all VM configurations, as shown in Fig. 8. Taking CyberShake100 as an example, as the number of VMs increases, the scheduling cost of RFPSO demonstrates an ideal trend of continued decline and eventual convergence. Meanwhile, its cost control capability significantly outperforms those of the comparison algorithms. Specifically, compared with SADQN, GWOEM, IFA, and PSO, the execution costs are reduced by an average of 30.40%, 36.67%, 41.92%, and 57.06%, respectively, as shown in Fig. 8(a). In the Epigenomics100 scenario, RFPSO consistently maintains the lowest costs compared with GWOEM, IFA, and PSO, with its scheduling cost reaching the minimum when the number of VMs is 24. The experimental results indicate that the adoption of the dual-layer resource pool allocation strategy with high performance and low cost proposed by RFPSO can effectively adapt to VM environments of different scales and stably reduce the execution cost of workflows. In Fig. 8(c) and (d), RFPSO continues to maintain its leading position under the Montage100 and LIGO100 workflows. In the Montage100 environment, RFPSO does not show a significant advantage in scheduling cost as the number of VMs increases, with its costs remaining relatively stable across the four VM scales. However, when compared with SADQN, GWOEM, IFA, and PSO, RFPSO demonstrates considerable superiority in workflow scheduling cost reduction. Specifically, it achieves average decreases of 39.03%, 45.10%, 52.50%, and 62.81% compared with the three algorithms, respectively.

Experiment 3: The experiment varies the number of nodes from 30 to 1000, with α ranging from 0.05 to 0.25 and the number of VMs fixed at 24. The average execution costs for the four workflow types (CyberShake, Epigenomics, Montage, and LIGO) are analyzed and compared, as illustrated in Fig. 9.

As shown in Fig. 9, RFPSO demonstrates significant *RU* advantages across all four workflows. For the CyberShake workflow, RFPSO achieves average improvements in resource utilization of 29.98%, 24.35%, 56.16%, and 66.00% compared with SADQN, GWOEM, IFA, and PSO, respectively. Similarly, in the Epigenomics workflow, RFPSO maintains the highest *RU* among all comparison algorithms. Especially when the number of nodes reaches 1000, the *RU* of RFPSO is particularly outstanding compared with the three other benchmark algorithms, as detailed in Fig. 9(a) and (b). Therefore, in large-scale workflow scenarios, RFPSO consistently enhances resource utilization by adopting a balance-oriented strategy during the RF-assisted initialization phase, which allocates high-performance VM resources to critical tasks.

Experiment 4: The deadline factor α is set in the range of 0.05 to 0.25, while N is fixed at 1000 and V at 24. Based on this, the resource utilization of the RFPSO, SADQN, GWOEM, IFA, and PSO algorithms under different deadline factors is compared, with the corresponding results shown in Fig. 10.

As shown in Fig. 10, RFPSO generally demonstrates pronounced and highly stable resource utilization advantages across all workflow scenarios, with its average *RU* exceeding those of PSO, IFA, SADQN, and GWOEM by 79.45%, 56.28%, 34.67%, and 10.47% of its own value, respectively. For example, as illustrated in Fig. 10(a) for the Epigenomics workflow, RFPSO's average UR surpasses that of PSO, IFA, SADQN, and GWOEM by 75.50%, 52.87%, 20.62%, and 12.76% of its own value, respectively. The trend of data changes suggests that, regardless of the deadline constraint, the resource utilization of RFPSO consistently remains the highest. The curve shows minor fluctuations but remains stable overall, which demonstrates its robust adaptability to varying deadline requirements. Compared with other algorithms,

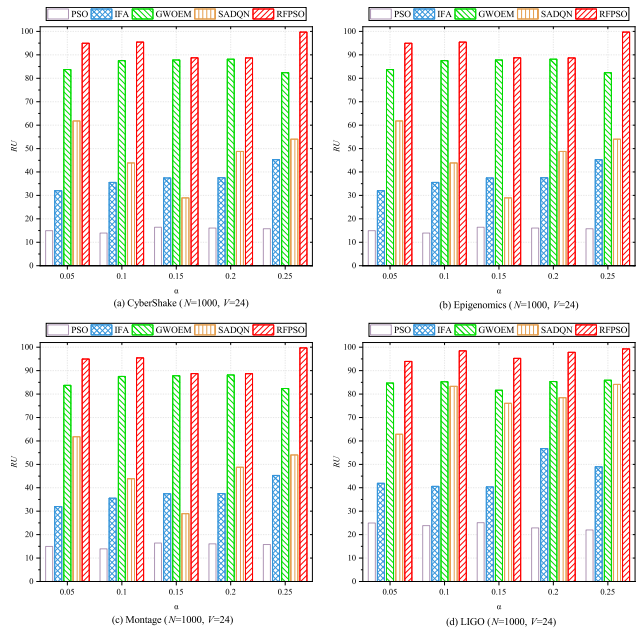

Fig. 10. *RU* (%) of four workflows under different deadline factor.

Table 6
Success rate (%) of different algorithms at $\alpha = 0.25$.

Workflow	Scale	RFPSO (%)	SADQN (%)	GWOEM (%)	IFA (%)	PSO (%)
CyberShake	30	100	100	87.92	90.54	100
	50	100	100	62.98	91.15	100
	100	100	100	100	100	100
	1000	100	100	100	100	100
Epigenomics	30	100	87.33	69.97	94.67	100
	50	100	94.03	97.85	93.93	100
	100	100	100	100	100	100
	1000	100	100	100	100	100
Montage	30	100	74.10	37.92	85.24	95.39
	50	100	95.99	82.94	96.27	100
	100	100	94.89	100	100	100
	1000	100	100	100	100	100
LIGO	30	100	95.99	82.94	96.27	100
	50	100	100	94.79	96.93	100
	100	100	100	100	100	100
	1000	100	100	100	100	100

RFPSO not only achieves higher *RU* values but also exhibits smaller fluctuations. This performance indicates its stronger resource utilization capability under diverse workflow and resource configuration scenarios.

Experiment 5: The number of nodes is varied from 30 to 1000, with α fixed at 0.25. The average success rates (SRs) for the four workflow types – CyberShake, Epigenomics, Montage, and LIGO – are analyzed, as presented in Table 6.

As illustrated in Table 6, RFPSO demonstrates outstanding performance in the Montage and LIGO workflows. Regardless of the task scale, RFPSO maintains the highest SRs, while the SADQN, GWOEM, IFA, and PSO algorithms exhibit significant fluctuations. The table and data trends demonstrate that the SADQN, GWOEM and IFA algorithms exhibit lower SRs at small task scales. Although their SRs gradually improve as the task scale increases, they consistently fail to reach the stable performance level of RFPSO. By prioritizing critical tasks and optimizing resource utilization, RFPSO mitigates resource contention and effectively avoids getting trapped in local optima.

Experiment 6: This experiment aims to quantify the optimization effect of RFPSO in scheduling costs compared to SADQN, GWOEM, IFA, and PSO. The execution cost data generated by different workflows under various α configurations are processed, and the $ACRP$ of RFPSO relative to each comparative algorithm is calculated, as shown in Table 7.

Table 7
Comparative analysis of $ACRP$ achieved by RFPSO over baseline algorithms across different workflows and scale.

Workflow	Scale	SADQN (%)	GWOEM (%)	IFA (%)	PSO (%)
CyberShake	30	57.66	64.48	79.75	88.67
	50	62.59	68.98	78.94	90.42
	100	60.99	68.41	77.46	89.92
	1000	82.44	82.02	86.71	94.28
Epigenomics	30	35.2	47.36	87.48	79.33
	50	48.96	58.92	85.16	84.02
	100	68.44	74.63	79.57	92.24
	1000	85.41	84.61	88.13	94.76
Montage	30	61.58	67.11	89.73	88.48
	50	37.06	54.81	84.18	82.59
	100	68.99	77.93	84.73	93.49
	1000	56.49	89.68	91.36	97.06
LIGO	30	37.06	54.81	84.18	82.59
	50	40.48	58.39	81.24	85.06
	100	44.24	61.52	77.59	86.87
	1000	82.44	82.02	86.71	94.28

Under experimental conditions with varying task node scales, the performance comparison of mainstream scheduling algorithms SADQN, GWOEM, IFA, and PSO on the $ACRP$ metric is shown in Table 7. The experimental results demonstrate that the performance ranking of the four algorithms remains consistently stable. Among them, RFPSO, by leveraging its reflective boundary constraint mechanism, dynamically updates the available resource range to effectively manage the solution space search process. Combined with its hierarchical allocation strategy for critical paths, it ensures that critical tasks are prioritized for computing resources. As a result, RFPSO exhibits the lowest cost advantage than SADQN, GWOEM, IFA, and PSO. As the scale of workflow nodes continues to expand, the $ACRP$ values of RFPSO relative to SADQN, IFA, and GWOEM both show a declining trend, which indicates that the performance gap between the algorithms narrows in large-scale task scenarios. Compared with SADQN, IFA, GWOEM, and PSO, RFPSO demonstrates the lowest cost across all workflow types, which showcases a stable advantage.

6. Conclusions

In this study, the RFPSO algorithm is proposed to address the challenges of uncertain task execution times in multi-cloud computing environments caused by strict deadline constraints and dynamic variations in resource performance, pricing, and task dependencies. RFPSO integrates random forest with an improved PSO algorithm, which effectively minimizes workflow scheduling costs under deadline constraints. RFPSO introduces a random forest regression model to assist in generating high-quality initial populations and proposes dynamic resource pool adjustment and boundary reflection mechanisms. This mechanism enhances the effective utilization of the solution space and improves algorithm stability by dynamically updating the available resource range and applying reflective correction to out-of-bounds particles. These procedures reduce inefficient search behavior. Experimental results demonstrate that the RFPSO algorithm achieves superior execution cost minimization while meeting deadline requirements compared with all baseline algorithms. However, the reliance on machine learning models during the initial particle generation phase may lead to biases in the distribution of training data samples, which can in turn affect the diversity and global optimality of the scheduling solutions.

Our future research will explore a mechanism for distributed collaborative training of a global model. The core of this mechanism lies

in moving away from reliance on a potentially biased centralized data source. Instead, it will aggregate multiple local data nodes from diverse workflow scenarios and resource environments to collaboratively train a global model. This approach fundamentally mitigates the issue of biased data sample distribution, as it integrates diverse and decentralized local data, which results in a global model with stronger generalization capabilities and robustness.

CRediT authorship contribution statement

Longxin Zhang: Writing – review & editing, Writing – original draft, Methodology, Funding acquisition, Formal analysis. **Lili Du:** Writing – review & editing, Writing – original draft, Methodology. **Mengying Guo:** Methodology, Data curation. **Zhihua Wen:** Software, Methodology. **Aijia Ouyang:** Software, Data curation. **Jiwu Peng:** Validation, Methodology. **Keqin Li:** Supervision, Methodology.

Declaration of competing interest

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

Acknowledgments

The authors would like to thank three anonymous reviewers for their suggestions to improve the manuscript. This work was partially funded by the Postgraduate Scientific Research Innovation Project of Hunan Province (Grant No. CX20251653, CX20261658), the Natural Science Foundation of Hunan Province, China (Grant Nos. 2023JJ50204, 2024JJ7143), the Scientific Research Foundation of Hunan Provincial Education Department, China (Grant No. 25A0422), the National Key R&D Program of China (Grant No. 2018YFB1003401), the National Natural Science Foundation of China (Grant Nos. 61702178, 62372495, 62402165, 62572186), and the State Key Laboratory Program for Novel Software Technology (Grant No. KFKT2024B52).

Data availability

The source code that support the findings of this study are available in/from hyperlink to data source https://github.com/lilidu1026/RFPSO.

References

[1] C. Li, H. Sun, H. Tang, Y. Luo, Adaptive resource allocation based on the billing granularity in edge-cloud architecture, Comput. Commun. 145 (2019) 29–42.
[2] X. Cai, Y. Zhang, M. Li, L. Wu, W. Zhang, J. Chen, Dynamic deadline constrained multi-objective workflow scheduling in multi-cloud environments, Expert Syst. Appl. 258 (2024) 125168.
[3] J. Barbosa, B. Fonseca, M. Ribeiro, J. Correia, L.D. da Silva, R. Gheyi, D. Baia, Evaluating the noise tolerance of cloud NLP services across amazon, microsoft, and google, Comput. Ind. 164 (2025) 104211.
[4] M.A.N. Saif, S.K. Niranjan, B.A.H. Murshed, CSO-ILB: chicken swarm optimized inter-cloud load balancer for elastic containerized multi-cloud environment, J. Supercomput. 79 (1) (2023) 1111–1155.
[5] Z. Sun, Y. Mei, F. Zhang, H. Huang, C. Gu, M. Zhang, Multi-tree genetic programming hyper-heuristic for dynamic flexible workflow scheduling in multi-clouds, IEEE Trans. Serv. Comput. 17 (5) (2024) 2687–2703.
[6] P. Suresh, P. Keerthika, R.M. Devi, G.K. Kamalam, K. Logeswaran, K.K. Sadasivuni, K. Devendran, Optimized task scheduling approach with fault tolerant load balancing using multi-objective cat swarm optimization for multi-cloud environment, Appl. Soft Comput. 165 (2024) 112129.
[7] P. Shobeiri, M.A. Rastaghi, S. Abrishami, PCP-ACO: a hybrid deadline-constrained workflow scheduling algorithm for cloud environment, J. Supercomput. 80 (6) (2024) 7750–7780.
[8] S.S. Sefati, A.M. Nor, B. Arasteh, A probabilistic approach to load balancing in multi-cloud environments via machine learning and optimization algorithms, J. Grid Comput. 23 (1) (2025) 16.
[9] X. Lu, C. Liu, S. Zhu, Y. Mao, P. Lio, P. Hui, RLPTO: A reinforcement learning-based performance-time optimized task and resource scheduling mechanism for distributed machine learning, IEEE Trans. Parallel Distrib. Syst. 34 (12) (2023) 3266–3279.
[10] Y. Yang, H. Shen, H. Tian, Scheduling workflow tasks with unknown task execution time by combining machine-learning and greedy-optimization, IEEE Trans. Serv. Comput. 17 (3) (2024) 1181–1195.
[11] A. Hayat, Y.N. Khalid, M.S. Rathore, A machine learning-based resource-efficient task scheduler for heterogeneous computer systems, J. Supercomput. 79 (11) (2023) 15700–15728.
[12] Y. Jin, An effective method for prospective scheduling of tasks in cloud-fog computing with an energy consumption management approach based on Q-learning, Eng. Appl. Artif. Intell. 151 (2025) 110705.
[13] F.S. Alsubaei, A.Y. Hamed, M.R. Hassan, M. Mohery, M.K. Elnahary, Machine learning approach to optimal task scheduling in cloud communication, Alex. Eng. J. 89 (2024) 1–30.
[14] A.H. Dinnillah, M. Widiar, A. Wijaya, Robust phishing detection using random forest with isolation forest based outlier filtering, Int. J. Adv. Comput. Inform. 1 (2) (2025) 99–105.
[15] C.I. Amalia, N. Rahmawati, Handling multiclass imbalance in diabetes, cancer, and pneumonia classification using NR-clustering SMOTE, Int. J. Adv. Comput. Inform. 2 (2) (2026) 83–95, http://dx.doi.org/10.71129/ijaci.v2i2.pp83-95.
[16] N.M.A. Tabalvandani, M.H. Shirvani, H. Motameni, Reliability-aware web service composition with cost minimization perspective: a multi-objective particle swarm optimization model in multi-cloud scenarios, Soft Comput. 28 (4) (2024) 5173–5196.
[17] C.T. Kamanga, E. Bugingo, S.N. Badibanga, A multi-criteria decision making heuristic for workflow scheduling in cloud computing environment, J. Supercomput. 79 (1) (2023) 243–264.
[18] H. Topcuoglu, S. Hariri, M.-Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274.
[19] M. Alam, M. Shahid, S. Mustajab, F. Ahmad, R.A. Haidri, Security driven cost-effective deadline aware workflow allocation strategy in cloud computing environment, in: 2023 3rd International Conference on Technological Advancements in Computational Sciences, ICTACS, IEEE, 2023, pp. 1–7.
[20] F. Bano, F. Ahmad, M. Shahid, M. Alam, F. Hasan, M. Sajid, A levelized multiple workflow heterogeneous earliest finish time allocation model for infrastructure as a service (IaaS) cloud environment, Algorithms 18 (2) (2025) 99.
[21] L. Zhang, M. Ai, K. Liu, J. Chen, K. Li, Reliability enhancement strategies for workflow scheduling under energy consumption constraints in clouds, IEEE Trans. Sustain. Comput. 9 (2) (2024) 155–169.
[22] Y. Zhang, L. Zhang, B. Cao, J. Liu, W. Zhao, J. Chen, K. Li, Adaptive-oriented mutation snake optimizer for scheduling budget-constrained workflows in heterogeneous cloud environments, Future Gener. Comput. Syst. 175 (2026) 108118.
[23] L. Li, C. Zhou, P. Cong, Y. Shen, J. Zhou, T. Wei, Makespan and security-aware workflow scheduling for cloud service cost minimization, IEEE Trans. Cloud Comput. 12 (2) (2024) 609–624.
[24] X. Huang, M. Xie, D. An, S. Su, Z. Zhang, Task scheduling in cloud computing based on grey wolf optimization with a new encoding mechanism, Parallel Comput. 122 (2024) 103111.
[25] B. Patil, S. Ket, Machine learning based secure and efficient task allocation in multi-cloud, Concurr. Comput.: Pract. Exp. 35 (27) (2023) e7848.
[26] H. Zhou, A novel approach to cloud resource management: hybrid machine learning and task scheduling, J. Grid Comput. 21 (4) (2023) 68.
[27] M.A. Rodriguez, R. Buyya, Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds, IEEE Trans. Cloud Comput. 2 (2) (2014) 222–235.
[28] J. Xu, H. Yu, G. Fan, J. Zhang, Uncertainty-aware scheduling of real-time workflows under deadline constraints on multi-cloud systems, Concurr. Comput.: Pract. Exp. 35 (5) (2023) e7562.
[29] L. Zhang, L. Du, Y. Zhang, J. Peng, J. Chen, B. Cao, Cost-aware multiple workflow scheduling algorithm under deadline constraints in cloud edge environments, in: 2025 IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA, IEEE, Shenyang, China, 2025, pp. 124–131.
[30] Z. Li, H. Yu, G. Fan, J. Zhang, Cost-efficient fault-tolerant workflow scheduling for deadline-constrained microservice-based applications in clouds, IEEE Trans. Netw. Serv. Manag. 20 (3) (2023) 3220–3232.
[31] L. Zhang, M. Ai, R. Tan, J. Man, X. Deng, K. Li, Efficient prediction of makespan matrix workflow scheduling algorithm for heterogeneous cloud environments, J. Grid Comput. 21 (4) (2023) 75.
[32] J. Liu, Y. Huang, C. Deng, L. Zhang, C. Chen, K. Li, Efficient resource allocation algorithm for maximizing operator profit in 5G edge computing network, J. Grid Comput. 23 (1) (2025) 1–18.
[33] B. Lin, C. Lin, X. Chen, M. Lin, G. Huang, Z. Xu, Cost-driven scheduling for workflow decision making systems in fuzzy edge-cloud environments, IEEE Trans. Autom. Sci. Eng. 22 (2025) 3756–3771.
[34] X. Tang, Reliability-aware cost-efficient scientific workflows scheduling strategy on multi-cloud systems, IEEE Trans. Cloud Comput. 10 (4) (2022) 2909–2919.

[35] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.
[36] W. Hu, G.G. Yen, Adaptive multiobjective particle swarm optimization based on parallel cell coordinate system, IEEE Trans. Evol. Comput. 19 (1) (2015) 1–18.
[37] G. Singh, A.K. Chaturvedi, A cost, time, energy-aware workflow scheduling using adaptive PSO algorithm in a cloud–fog environment, Comput. Arch. Inform. Numer. Comput. 106 (10) (2024) 3279–3308.
[38] X. Tang, Cost-efficient workflow scheduling algorithm for applications with deadline constraint on heterogeneous clouds, IEEE Trans. Parallel Distrib. Syst. 33 (9) (2022) 2079–2092.
[39] Microsoft Corporation, Microsoft azure pricing, 2023, https://azure.microsoft.com/ec2/pricing/.
[40] Amazon Web Services, Inc., Amazon EC2 pricing, 2023, https://aws.amazon.com/ec2/pricing/.
[41] Google LLC, Google compute engine pricing, 2023, https://cloud.google.com/ec2/pricing/.
[42] M.A. Rodriguez, R. Buyya, Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds, IEEE Trans. Cloud Comput. 2 (2) (2014) 222–235.
[43] Pegasus Workflow Management System, Pegasus workflow generator, 2023, https://confluence.pegasus.isi.edu/.
[44] L. Zhang, R. Tan, B. Cao, L.A.K. Li, K. Li, EP-MUSTO: Entropy-enhanced DRL-based task offloading in secure multi-UAV-assisted collaborative edge computing, IEEE Internet Things J. 12 (16) (2025) 34459–34470.
[45] Z. He, J. Wang, M. Jiang, L. Hu, Q. Zou, Random subsequence forests, Inform. Sci. 667 (2024) 120478.
[46] Y. Gu, F. Cheng, L. Yang, J. Xu, X. Chen, L. Cheng, Cost-aware cloud workflow scheduling using DRL and simulated annealing, Digit. Commun. Netw. 10 (6) (2024) 1590–1599.
[47] H. Mikram, S. El Kafhali, CHPSO: An efficient algorithm for task scheduling and optimizing resource utilization in the cloud environment, J. Grid Comput. 23 (1) (2025) 15.