algorithms

MDPI

Article

D3QN-Guided Sand Cat Swarm Optimization with Hybrid Exploration for Multi-Objective Cloud Task Scheduling

Minghao Shao [1,2], Ying Guo [1,2], Jibin Wang [1,2] and Hu Zhang [1,2,*]

[1] Key Laboratory of Computing Power Network and Information Security, Ministry of Education, Shandong Computer Science Center (National Supercomputer Center in Jinan), Qilu University of Technology (Shandong Academy of Sciences), Jinan 250013, China; 18266061700@163.com (M.S.); guoying@sdas.org (Y.G.); wangjb@sdas.org (J.W.)

[2] Shandong Provincial Key Laboratory of Computing Power Internet and Service Computing, Shandong Fundamental Research Center for Computer Science, Jinan 250103, China

* Correspondence: zhanghu@sdas.org

Abstract

Task scheduling in cloud computing environments is a complex NP-hard problem that requires maximizing resource utilization while satisfying quality-of-service (QoS) constraints. Traditional meta-heuristic algorithms often become stuck in local optima, while single deep reinforcement learning (DRL) models exhibit instability when exploring large-scale solution spaces. To address this, this paper proposes a hybrid scheduling algorithm based on multi-objective sand cat colony optimization (MoSCO). This algorithm utilizes a D3QN network to extract task features and guide population initialization, followed by a multi-objective Sand Cat Swarm Optimization (SCSO) algorithm for refined local search. Results from 50 independent replicate experiments conducted in a simulated cloud environment, coupled with an analysis of the dynamic convergence process, demonstrate that MoSCO exhibits significant superiority and robustness. Scatter plot convergence analysis further confirms that MoSCO's knowledge injection mechanism effectively overcomes the blind exploration phase of traditional algorithms and successfully breaks through the local optimum bottleneck in the late iteration stages of single reinforcement learning, achieving higher-quality, denser, and more stable convergence. Furthermore, 3D and 2D Pareto front analyses show that MoSCO generates highly competitive, well-distributed non-dominated solutions, offering flexible trade-off options for conflicting objectives. Compared to PureD3QN, H-SCSO, and NSGA-II, MoSCO exhibits the smallest performance fluctuations in box plots. Specifically, MoSCO elevates the average resource utilization of clusters to 92.20%, while reducing the average maximum Makespan and Tardiness to 528 and 4187, respectively. Experimental data confirm that MoSCO effectively balances global exploration with local exploitation, delivering stable, high-quality solutions for dynamic cloud task scheduling.

Keywords: cloud computing; task scheduling; multi-objective optimization; hybrid algorithm

Check for updates

Academic Editors: Seyed Jalaleddin Mousavirad and Mehran Pourvahab

Received: 5 March 2026
Revised: 3 April 2026
Accepted: 15 April 2026
Published: 20 April 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

1.1. Background and Motivation

Cloud computing pools computing, storage, and networking resources through virtualization technology [1], providing users with on-demand, elastic, and scalable services. It has become the core infrastructure driving the development of the digital economy. With the rapid expansion of IoT and big data applications, cloud data centers encounter

massive, heterogeneous, and dynamically changing task requests [2]. Maximizing resource utilization for cloud service providers while satisfying user QoS requirements—such as reducing latency and completion time—has become an urgent NP-hard optimization problem [3]. The main goal of task scheduling is to effectively assign a variety of heterogeneous tasks onto virtual machine resources to improve specific performance metrics [4,5].

1.2. Related Work

To address this scheduling problem, various approaches have been proposed in recent years. These methods can be broadly categorized into heuristic or meta-heuristic algorithms, learning-based approaches, and hybrid methods.

1.2.1. Heuristic and Meta-Heuristic Methods

Early heuristic rules are computationally simple but often lack global search capabilities in large-scale heterogeneous environments. To address this, meta-heuristic algorithms have been widely adopted. For instance, Chatterjee et al. proposed the TOPSIS-based CEFT-LB algorithm to optimize workflow scheduling and ensure load balancing among processors [6]. Mahmoud et al. combined decision trees with TOPSIS-EWM to construct a multi-criteria decision framework that effectively optimizes metrics such as makespan, cost, and energy consumption in heterogeneous environments [7]. In the meta-heuristic domain, Cui et al. proposed a strategy based on weighted vector decomposition to transform complex multi-objective problems into subproblem solutions [8]. Devi and Valli designed the GEC-DRP method, using K-means clustering and exponential smoothing models to predict load and dynamically allocate resources [9]. Mishra et al. introduced an Optimal Backward Learning mechanism to enhance the Tasmanian Devil Optimization algorithm, significantly improving convergence speed during the exploration and exploitation phases [10]. However, despite having global search capabilities, the performance of these meta-heuristic algorithms heavily relies on random initialization [11]. Without prior knowledge, they often operate in a "blind" state during the initial phases, which can lead to slow convergence and fluctuations in the solution quality [12].

1.2.2. Hybrid Algorithm

To mitigate the limitations of single approaches, hybrid algorithms attempt to combine different strategies. Hemanth et al. employed a multi-queue strategy integrated with a multi-objective ant colony optimization to optimize task allocation [13]. Malti et al. combined the flower pollination algorithm with grey wolf optimization to address multi-objective challenges in heterogeneous environments [14]. Amer et al. proposed SMO_ACO, which fuses spider monkey optimization with ant colony optimization, to mitigate the local optimum problem [15]. However, most of the aforementioned hybrid algorithms rely on simple sequential combinations and lack deep, knowledge-guided mechanisms, which limit their performance in large-scale solution spaces.

In addition, recent research has explored various combinations of swarm intelligence, genetic algorithms (GA), deep neural networks (DNNs), and clustering techniques to enhance optimization capabilities. For example, in single-objective continuous optimization, hybrid multi-population methods, such as the Hybrid Multi-Swarm Particle Swarm Optimization (HMSPSO) [16], have proven highly effective in modeling complex systems. Similarly, the integration of genetic operators into the field of swarm intelligence has demonstrated tremendous potential; Afrasyabi et al. proposed a multi-objective discrete particle swarm optimization model based on crossover operators, which successfully solved the multi-peak routing problem [17]. In the broader context of multi-objective optimization, state-of-the-art algorithms such as MA-HCAGA [18], MORCGA-MOPSO-II, and CBHPSO further demonstrate the strong potential of hybrid evolutionary algorithms to balance

global search and local exploration. Although these advanced hybrid methods have significantly improved search efficiency, their direct application to dynamic cloud scheduling often lacks a deep understanding of real-time conditions. This further motivated the development of our MoSCO algorithm, which integrates prior knowledge derived from deep reinforcement learning into the swarm optimization process.

However, despite these advancements, challenges persist: while these advanced hybrid methods greatly enhance search efficiency, applying them directly to dynamic cloud scheduling remains difficult due to the need for deep, real-time state awareness. Moreover, many of the hybrid algorithms mentioned earlier tend to rely on sequential combinations, which may not fully leverage deep knowledge-guided mechanisms, potentially limiting their effectiveness in large solution spaces. This observation drives the development of our MoSCO algorithm, which incorporates DRL-based prior knowledge into the swarm optimization process to address these issues and limitations.

1.2.3. Learning-Based Approach

With advances in deep learning, learning-based scheduling methods have attracted considerable attention. Hao et al. used GCNs to extract high-level features from DAG tasks and combined them with adaptive evolutionary algorithms for dynamic allocation [19]. Mangalampalli et al. proposed an enhanced A3C framework for multi-cloud environments. By incorporating residual convolutional neural networks to capture nonlinear relationships between tasks and hosts, and leveraging an asynchronous parallel architecture, they significantly improved training efficiency and scheduling robustness [20]. Similarly, recent studies have proposed dedicated Deep Reinforcement Learning Based Task-Scheduling Algorithms (DRLBTSAs) to learn optimal resource allocation policies [21]. Furthermore, to address increasingly complex scenarios, Fan et al. introduced a dual-agent DRL framework to jointly optimize service placement and task scheduling in multi-user edge computing networks [22], and Cui et al. proposed a novel hierarchical DRL framework that dynamically allocates tasks across virtual machine clusters to balance cost and load [23]. Nevertheless, pure DRL models (e.g., DQN, D3QN) often show instability during training [24]. Agents are sensitive to hyperparameters and tend to become stuck in local optima when exploring large discrete state spaces, resulting in high variance in scheduling policies [25].

1.3. Main Contributions

Faced with these challenges, relying on a single method often struggles to achieve both high-quality solutions and optimization stability. Therefore, exploring a scheduling architecture that leverages DRL's perceptual capabilities to guide metaheuristic algorithms, while using the fine-grained search of metaheuristics to compensate for DRL's instability, presents a promising research direction. Motivated by these considerations, this paper proposes a hybrid scheduling algorithm based on multi-objective sand cat colony optimization (MoSCO). The main contributions of this work are summarized as follows:

- We propose the MoSCO algorithm, a hybrid framework that combines the rapid environmental perception of a D3QN agent with the multi-objective search capabilities of the Sand Cat Swarm Optimization algorithm.
- We design a knowledge-injection mechanism in which the D3QN network extracts task features to guide population initialization, aiming to reduce blind exploration commonly observed in traditional metaheuristics.
- We formulate a multi-objective optimization model for dynamic cloud environments that accounts for node failure probabilities and simultaneously optimizes the maximum Makespan, task Tardiness, and Average Resource Utilization.

- We conduct extensive simulations to evaluate the proposed MoSCO algorithm. The results show that it can effectively balance global exploration with local exploitation, demonstrating competitive performance and stability when compared to several existing baseline algorithms.

2. Materials and Methods

2.1. System Model

This section describes the task-scheduling architecture for cloud computing, including the task-scheduling process and a multi-objective optimization model.

2.1.1. Environment Model

To evaluate the performance of our proposed scheduling algorithm, we constructed a dynamic and heterogeneous simulated cloud computing environment. This environment consists of two components: the resource model and the task model.

1. ***Resource model:*** The data center contains M heterogeneous virtual machines, represented by the set $VM = vm_1, vm_2, \ldots, vm_M$. Resource heterogeneity manifests itself as differences in processing capabilities, meaning that the same task takes different processing times across different virtual machines. Each virtual machine maintains a state object recording its current task queue and start/end times (Start, End). To simulate real-world uncertainty, a random failure mechanism is introduced. During task scheduling, a virtual machine has a base failure probability, BP, and incurs a random repair time, Repair_time. Notably, high-load nodes (those in the top 90% by completion time) are assigned a higher failure probability.
2. ***Task Characteristics:*** Task flows arrive dynamically following a stochastic process. The arrival intervals of the tasks follow an exponential distribution, and each task j_i has a distinct arrival time A_i. The processing relationship between tasks and resources is defined by a two-dimensional matrix $P_{i,k}$, representing the execution time of task j_i on virtual machine vm_k ($P_{i,k} = -1$ indicates incompatibility). QoS objectives are defined by the desired completion time D_i, which is the sum of the arrival time A_i and the estimated processing duration. If the task completion time C_i exceeds D_i, a latency delay occurs.

2.1.2. Multi-Objective Optimization Model

In this study, our objective is to identify an optimal scheduling policy that maps a sequence of dynamically arriving tasks onto virtual machines. To rigorously formulate this mathematically, we define a binary decision variable matrix $X = [x_{i,k}]$, where:

$$x_{i,k} = \begin{cases} 1, & \text{if task } j_i \text{ is assigned to } vm_k \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

The scheduling policy must strictly satisfy two constraints: Unique Assignment and Resource Compatibility. First, each task must be assigned to exactly one virtual machine:

$$\sum_{k=1}^{M} x_{i,k} = 1, \quad \forall i \in \{1, 2, \ldots, |J|\} \tag{2}$$

Second, tasks cannot be scheduled on incompatible nodes, and are indicated by a processing time of -1 in the two-dimensional execution matrix $P_{i,k}$:

$$x_{i,k} = 0, \quad \forall (i,k) \text{ where } P_{i,k} = -1 \tag{3}$$

Based on this decision matrix X, the dynamic features of the cloud environment are incorporated into the formulation of the objectives. The actual start time ST_i of task j_i depends on its dynamic arrival time A_i and the completion time of the immediately preceding task scheduled on the same virtual machine:

$$ST_i = \sum_{k=1}^{M} x_{i,k} \cdot \max\left(A_i, \max_{j \in Pre(i,k)} (C_j \cdot x_{j,k})\right) \tag{4}$$

where $Pre(i,k)$ represents the set of tasks assigned to vm_k before task j_i. Consequently, the actual completion time C_i incorporates the processing time $P_{i,k}$ and the stochastic repair time delay Δ_i^{repair} triggered by the dynamic failure probability BP:

$$C_i = ST_i + \sum_{k=1}^{M} x_{i,k} \cdot P_{i,k} + \Delta_i^{repair} \tag{5}$$

The utilization u_k of virtual machine vm_k is defined as the ratio of its actual active processing time to the overall Makespan of the system:

$$U_k = \frac{\sum_{i=1}^{|J|} (x_{i,k} \cdot P_{i,k})}{C_{max}} \tag{6}$$

By optimizing the decision variable matrix X, we aim to balance the following three conflicting goals to find the Pareto optimal scheduling matrix X^*:

1. ***Minimize Maximum Completion Time:*** The maximum completion time $C_{\mathbf{max}}$ is defined as the maximum value among all task completion times, reflecting the system throughput.

$$f_1 = \min(C_{\max}) = \min(\max(C_i)), i \in J \tag{7}$$

where J is the set of all tasks, and C_i is the completion time of task j_i.

2. ***Minimize Timeliness Deviation:*** This metric aims to measure the deviation of scheduling strategies from these predefined objectives, specifically by minimizing the cumulative lag between the actual completion times of all tasks and their expected times:

$$f_2 = \min(T_{\text{lag}}) = \min\left(\sum_{i \in J} \max(0, C_i - D_i)\right) \tag{8}$$

where D_i is the expected completion time for task j_i. The function $\max(0, C_i - D_i)$ ensures that the difference is only included in the cumulative deviation when the task falls behind the expected time; if the task is completed ahead of schedule or on time, this term is zero, indicating full compliance with expectations.

3. ***Maximize Average Utilization:*** This metric measures resource utilization efficiency and is defined as the average of the proportion of busy time across all virtual machines:

$$f_3 = \max(U_{\text{ave}}) = \max\left(\frac{1}{M} \sum_{k=1}^{M} U_k\right) \tag{9}$$

where U_k is calculated by Equation (6).

4. ***Comprehensive Optimization Objective:*** In summary, this paper aims to identify the Pareto optimal strategy π^* by optimizing the vector objective function:

$$\text{Optimize } F(\pi) = \{f_1(\pi), f_2(\pi), -f_3(\pi)\} \tag{10}$$

2.2. The Proposed MoSCO Algorithm

To effectively address multi-objective task scheduling in dynamic cloud computing environments, we propose a novel hybrid intelligent algorithm named MoSCO. The core concept of this algorithm lies in deeply integrating DRL's rapid environmental perception and decision-making capabilities with SCSO's powerful, fine-grained, multi-objective search capabilities. The D3QN agent acts as an experienced "guide," providing high-quality initial directions for SCSO's search. SCSO, functioning as the "optimizer," then explores these directions to identify the Pareto-optimal solution set.

2.2.1. Overall Algorithm Framework

The MoSCO algorithm constructs an online closed-loop scheduling framework that deeply integrates the rapid perception capabilities of D3QN with the global search advantages of SCSO. As shown in Figure 1, this framework primarily consists of the following three core modules working in concert:

- ***D3QN Lead Module:*** This module is responsible for perceiving the environmental state s and, based on the current policy, outputs a candidate action probability distribution for the current task generated by the D3QN agent's policy, serving as the 'elite individual' to be passed to the optimization module. Simultaneously, the module interacts with the Replay Memory Buffer, continuously training the network using historical experience tuples $(s, a, R_t, s', done)$ to enhance guidance quality
- ***SCSO Optimization Module:*** This module receives a_t^{D3QN} and incorporates it into the initial population via a Knowledge Injection mechanism, combining it with randomly generated individuals to balance solution quality and diversity. Subsequently, SCSO iterative optimization and Pareto non-domination sorting are performed to select the optimal scheduling policy a_t^* from the resulting Pareto frontier.
- ***Execution and Feedback Loop:*** The optimal policy a_t^* is applied to the cloud environment. The environment's multidimensional performance metrics are weighted and normalized to produce a scalar reward R_t. Finally, the new state s' and reward R_t are fed back into the D3QN module, forming a complete adaptive closed-loop of "guidance-optimization-feedback."

2.2.2. D3QN Guidance Module

This module is designed to leverage the strengths of DRL to provide a fast and effective macro-level decision direction for scheduling problems.

1. ***State and Action Space Definition:***
   - ***State Space:*** To accurately characterize the dynamic cloud environment, the state s_t is defined as a six-dimensional continuous vector

$$s_t = \{U_{ave}, U_{std}, CRJ_{ave}, CRJ_{std}, P_{pending}, P_{due}\}$$

     encompassing resource load characteristics and task queue status. To enhance training stability, all state features are normalized before being fed into the network. Among these, U_{ave} represents the average resource utilization rate; U_{std} denotes the standard deviation of resource utilization; CRJ_{ave} is the average task completion rate; CRJ_{std} indicates the standard deviation of task completion rate; $P_{pending}$ signifies the proportion of pending tasks; P_{due} represents the proportion of urgent tasks, reflecting the percentage of tasks approaching their expected completion time.
   - ***Action Space:*** To address the convergence challenges arising from large-scale combinatorial action spaces, this paper proposes a dynamic candidate action

generation mechanism based on ϵ-greedy. At decision time t, the algorithm constructs a fixed-dimensional candidate action set $A_{t_candidate}$ of size K_{pool}, following the strategy outlined below:

- ***Heuristic Development:*** Select the top $K_{pool} \times (1 - \epsilon)$ tasks with the tightest deadlines for the candidate pool, ensuring priority response to urgent demands.
- ***Random Exploration:*** Sample randomly from the remaining feasible tasks with probability ϵ to maintain diversity in the solution space and avoid becoming stuck in local optima.

The output action a_t^{D3QN} represents the index of the selected virtual machine at step t. The discrete operation is then converted into a "one-hot" probability vector, which serves as the initial elite individual for the subsequent SCSO micro-optimization phase.

2. ***D3QN Network Architecture and Learning Process:*** We employ the Dueling Double Deep Q-Network (D3QN) as the value function approximator. Its architecture consists of a shared feature extraction layer and two independent output streams: the state value stream $V(s;\theta,\alpha)$ and the action advantage stream $A(s,a;\theta,\alpha)$. The final Q-value is aggregated using the following formula:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|A|} \sum_{a'} A(s,a';\theta,\alpha) \right) \tag{11}$$

where α, θ, and β are the network parameters for the shared layer, value flow, and advantage flow, respectively.
The D3QN agent learns by minimizing the temporal difference (TD) error. After sampling an experience tuple

$$(s_t, a_t^{D3QN}, R_t, s_{t+1}, done)$$

from the experience replay pool, the loss function is defined as the mean squared error between the predicted Q-value and the target value y_t:

$$L(\theta_{\text{online}}) = \mathbb{E}[(y_t - Q(s_t, a_t; \theta_{\text{online}}))^2] \tag{12}$$

In particular, y_t is estimated using the Double DQN mechanism, which combines immediate rewards with future value estimation.

$$y = R_t + (1 - \text{done}) \cdot \gamma \cdot Q(s_{t+1}, \arg\max_{a'} Q(s_{t+1}, a'; \theta_{\text{online}}); \theta_{\text{target}}) \tag{13}$$

θ_{online} and θ_{target} represent the parameters of the online network and target network, respectively. The scalar reward R_t is obtained by linearly weighting the normalized multi-objective reward vector $\mathbf{r}_t$ according to preset weights.

2.2.3. SCSO Multi-Objective Optimization Module

This module is responsible for conducting a refined multi-objective search under the guidance of D3QN.

1. ***Knowledge-Guided Population Initialization:*** Instead of mapping a sequence of actions for a batch of tasks, the proposed method operates dynamically step-by-step. At each decision time step t, the D3QN provides a macro-level assignment direction for the current task, which is then injected into the SCSO population to accelerate the local search.

- ***Elite Solution Construction:*** At time step t, given the current system state s_t, the D3QN agent selects the optimal target virtual machine a_t^{D3QN} by maximizing the Q-value:

$$a_t^{D3QN} = \arg\max_a Q(s_t, a; \theta_{online}) \tag{14}$$

  To map this discrete decision into the continuous search space of the SCSO algorithm, we define each individual p_i in the population as an M-dimensional continuous probability distribution vector. This vector represents the preference or probability of assigning the current task to each of the M available virtual machines. The optimal action recommended by the D3QN is transformed into a one-hot encoded vector and directly injected as the first elite individual p_1:

$$p_1 = \text{OneHot}(a_t^{D3QN}) \tag{15}$$

- ***Diversity Preservation:*** To maintain search space diversity and prevent premature convergence, the remaining $N-1$ individuals $(p_2, \ldots, p_N)$ are randomly generated. Specifically, they are sampled from a Dirichlet distribution, $p_i \sim \text{Dirichlet}(\mathbf{1})$, which naturally ensures that each generated vector satisfies the probability constraint.

2. ***Multi-Objective Evaluation and Pareto Elite Selection:*** To accurately select high-quality solutions during the iteration process, this section introduces a decoding evaluation and diversity preservation mechanism.
   - ***Fitness Evaluation:*** Since individual p_i is a continuous probability distribution vector, the algorithm first employs an Argmax strategy to decode it into a discrete scheduling action a_i. Subsequently, this action is simulated in an isolated, temporary replica of the current cloud environment. By executing this single assignment, we compute its immediate impact on the global system state across three dimensions: the updated maximum Makespan, the newly accumulated total Tardiness, and the real-time average Utilization. This ensures that the global consequences of a single-step decision can be accurately evaluated without interfering with the DRL main environment state.
   - ***Pareto Elitism:*** Perform non-dominated sorting based on fitness vectors to identify the Pareto frontier. To maintain a fixed population size N and prevent premature convergence, a Crowding Distance mechanism is introduced: when the number of non-dominated solutions exceeds the population capacity, individuals with low crowding distance are prioritized for removal. Sparsely distributed solutions are retained to preserve population diversity. The resulting elite set P_{elite} guides the evolution of the next generation.
3. ***Population Renewal (Exploration and Attack):*** Based on the selected elite population P_{elite}, the SCSO module generates the next generation population P_{t+1} by simulating the predatory behavior of sand cats. This process is regulated by the adaptive sensitivity parameter r_G, which linearly decreases from r_{G_init} to r_{G_min} with the iteration count l. To balance solution quality and diversity, the algorithm employs a strategy combining elite retention with evolutionary generation: First, the top K_{elite} optimal individuals from P_{elite} are directly copied into P_{t+1}. Subsequently, the remaining elite individuals serve as parents, and the remaining population slots are filled with a 50% probability by randomly executing one of the following two operators:

- ***Global Exploration:*** Simulate the wide-area search behavior of sand cats. Apply random perturbations with amplitude controlled by R_t to selected elite individuals p_i to generate new individuals p_{new}, thereby escaping local optima:

$$p_{new} = p_i + r_G \cdot \mathcal{N}(0,1) \tag{16}$$

  Here, $\mathcal{N}(0,1)$ denotes the standard normal distribution vector. A larger value of R_t ensures that the algorithm possesses a broad search horizon during its initial stages.
- ***Localized attack:*** Simulating the sand cat's precise attack on its prey. Due to the existence of multiple non-dominated Pareto-optimal solutions in multi-objective optimization, there is no single globally optimal solution. Therefore, the algorithm employs a random guidance strategy: it randomly samples an individual from the current elite set P_{elite} as a temporary "prey" target, p_{target}. Individual p_i will converge toward this target according to the following formula:

$$p_{new} = p_i + r_G \cdot (p_{target} - p_i) \cdot r \tag{17}$$

  Here, $r \sim U(0,1)$. This mechanism not only enhances local development capabilities but also guides the population to distribute along the entire Pareto frontier, preventing the solution set from becoming overly concentrated.

2.2.4. DRL Feedback and Learning Mechanism

This section establishes a "guidance-optimization-feedback" closed-loop system that converts the multi-objective optimization results from SCSO into scalar rewards for DRL, thereby training the agent to identify high-quality search starting points.

1. ***Reward Definition and Scaling:*** To transform lagging system performance into immediate feedback, we define a multi-objective reward vector r_t composed of delay penalties, utilization rewards, and completion time penalties:

$$r_t = \left\{ -\frac{\sum \max(0, C_i - D_i)}{|J_{done}|}, \quad \frac{1}{M}\sum U_k, \quad -\max(C_i) \right\} \tag{18}$$

   Given the substantial differences in the dimensionality of each component, the algorithm employs dynamic Z-score normalization based on exponential moving averages. This is combined with preference weights W to compute the final scalar reward R_t:

$$R_t = W \cdot \frac{r_t - \mu_t}{\sigma_t + \epsilon} \tag{19}$$

   where W is as follows:

$$W = \{w_{tard}, w_{util}, w_{make}\} \tag{20}$$

   By setting different values for W, we can simulate scenarios where users have varying needs in real-world applications. In the following experiment, W is set to a fixed value to simulate a specific application scenario.
2. ***Experience Replay and Meta-Learning Strategies:*** We store the experience tuple $(s_t, a_t^{D3QN}, R_t, s_{t+1}, done)$ into the replay pool. In this mechanism, D3QN serves as a "meta-learner": although SCSO executes the action a_t^*, the system reward R_t is attributed to the state-action trajectory that generated the initial solution. Through this approach, the intelligent agent can learn to identify high-potential initial search directions, thereby achieving a deep integration of macro-level guidance and micro-level optimization.

2.3. Overall Procedure of the MoSCO Algorithm

The complete execution workflow of MoSCO is detailed in Algorithm 1. It operates as a closed-loop synergy in which D3QN provides macro-guidance via knowledge injection, and SCSO performs micro-optimization to identify the Pareto-optimal scheduling action. By feeding the scalarized execution rewards back to update the DRL agent, this architecture robustly overcomes local optima while maintaining high search stability.

Algorithm 1: MoSCO: D3QN-Guided Multi-Objective Sand Cat Swarm Optimization

Input: Cloud environment; D3QN params (ϵ, $BatchSize$); SCSO params (N, T_{max}, r_{G_init}, r_{G_min}); Weights $W = [w_{tard}, w_{util}, w_{make}]$.
Output: Pareto optimal scheduling strategy, trained D3QN model.

Initialize D3QN online network θ_{online}, target network θ_{target}, Replay Buffer D
Initialize RunningMeanStd for state and reward normalization
for *episode* $= 1$ **to** L **do**
  Reset environment, set clock $t = 1$, $done = False$
  while ***not*** *done* **do**
    Get a valid action set A_{valid}. **if** $A_{valid} = \emptyset$ **then** $t \leftarrow t + 1$, **continue**
    `/* Stage 1: D3QN Macro-Guidance */`
    $s_t \leftarrow$ Normalize(Current System State)
    $a_t^{D3QN} \leftarrow \epsilon$-greedy$(s_t, \theta_{online})$ `// Recommended action`
    `/* Stage 2: Knowledge Injection */`
    $p_1 \leftarrow \text{OneHot}(a_t^{D3QN})$ `// D3QN-guided elite`
    $P_0 \leftarrow \{p_1\} \cup \{p_i \sim \text{Dirichlet}(1)\}_{i=2}^{N}$ `// Random individuals`
    `/* Stage 3: SCSO Micro-Optimization */`
    $P_{current} \leftarrow P_0$
    for *iter* $= 1$ **to** T_{max} **do**
      Evaluate multi-objective fitness $F(p)$ for $\forall p \in P_{current}$
      $P_{elite} \leftarrow$ NonDominatedSorting$(P_{current})$
      $r_G(t) \leftarrow r_{G_init} - (r_{G_init} - r_{G_min}) \times (iter/T_{max})$
      $P_{new} \leftarrow P_{elite}$
      while $|P_{new}| < N$ **do**
        Select $p_i \in P_{elite}$
        if $rand() < 0.5$ **then**
          $p_{new} \leftarrow p_i + r_G(t) \times \mathcal{N}(0,1)$ `// Global Explore`
        else
          Select $p_{target} \in P_{elite}$
          $p_{new} \leftarrow p_i + r_G(t) \times (p_{target} - p_i) \times \mathcal{U}(0,1)$ `// Local Attack`
        Normalize p_{new} and add to P_{new}
      $P_{current} \leftarrow P_{new}$
    `/* Stage 4: Execution & Feedback Learning */`
    $a_t^* \leftarrow \arg\max_{a \in P_{elite}}(W \cdot F(a))$ `// Best action via weights`
    Execute a_t^*, observe raw multi-rewards r_t, next state s_{t+1}, and $done$
    $R_t \leftarrow W \cdot \text{Z-Score}(r_t)$ `// Reward scalarization`
    Store tuple $(s_t, a_t^{D3QN}, R_t, s_{t+1}, done)$ into D
    if $|D| \geq BatchSize$ **then**
      Sample mini-batch from D, update θ_{online} via Gradient Descent
    $t \leftarrow t + 1$
  Periodically update target network: $\theta_{target} \leftarrow \theta_{online}$

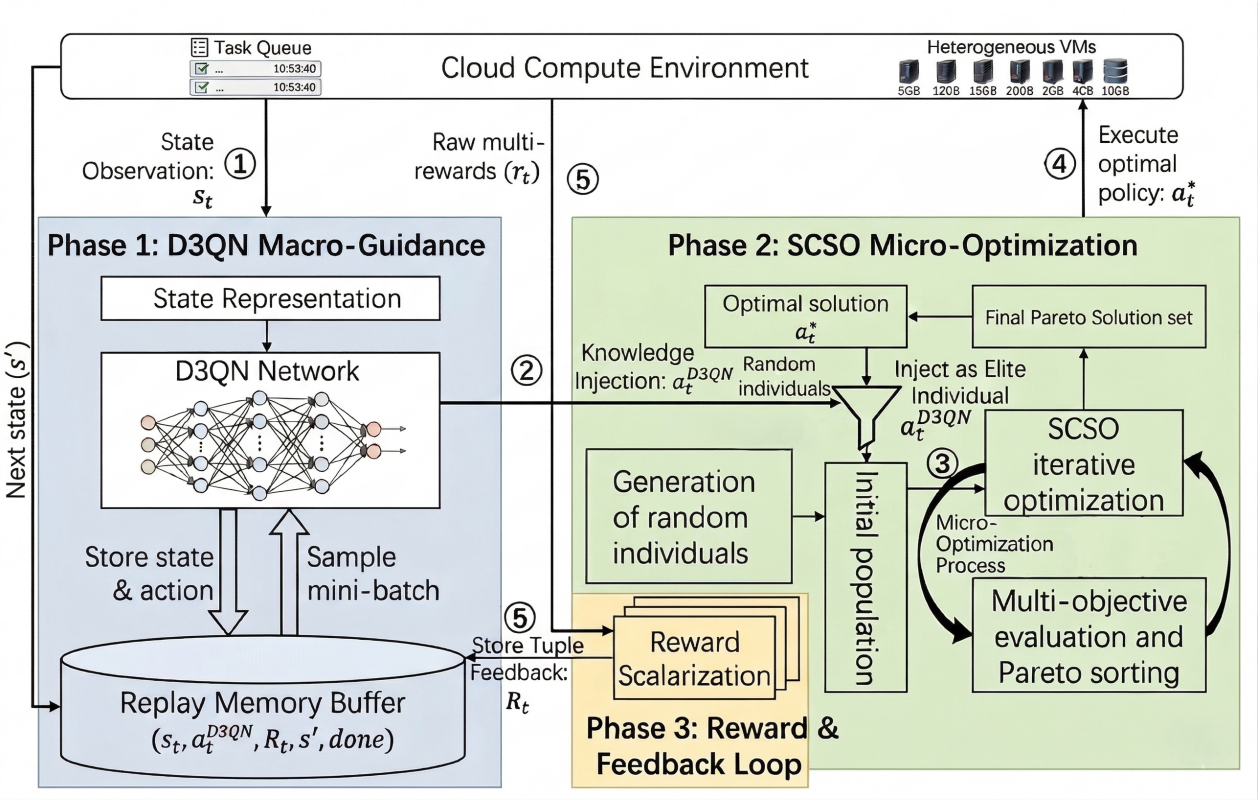


Figure 1. The architecture of MoSCO.

3. Experimental Simulation and Result Analysis

To validate the effectiveness of the MoSCO algorithm, we developed a dynamic, multi-objective task-scheduling simulation environment. We compared it with the PureD3QN, H-SCSO, and the classical multi-objective algorithm NSGA-II. Among these, H-SCSO is an SCSO algorithm guided by the rule of "sorting actions based on the urgency of their task deadlines." It is contrasted with MoSCO to illustrate D3QN's "guiding" role within the MoSCO algorithm.Furthermore, NSGA-II is recognized as the industry-standard benchmark for evaluating the overall search capabilities of multi-objective evolutionary algorithms. PureD3QN serves as another essential ablation baseline to demonstrate the importance of the SCSO micro-optimization module for escaping local optima, a challenge pure DRL models often face in complex combinatorial spaces. Together, these baselines create a comprehensive multi-dimensional comparison framework.

3.1. Experimental Setup

The experiment was implemented using Python 3.9 and the PyTorch 2.5.1 framework. The simulated data center comprises 50 heterogeneous virtual machines processing 500 dynamically arriving tasks. Task arrivals follow an exponential distribution with rate $\lambda = 50$, and processing times are randomly generated between $[1, 30]$. To simulate real-world uncertainty, the base failure probability of virtual machines is set to 0.1. The maximum iteration limit L_{max} for all algorithms is uniformly set to 200 to ensure convergence. Meanwhile, the weight settings for each sub-metric in the MoSCO algorithm are as follows:

$$W = \{w_{tard}, w_{util}, w_{make}\} = \{0.5, 0.3, 0.2\} \tag{21}$$

The purpose of setting these weights is to simulate a common scenario: since total delay directly measures Service Level Agreement (SLA) fulfillment and user satisfaction, the Tardiness weight is set at the highest; resource utilization directly affects return on investment

and operational costs, making it the service provider's top priority after ensuring user SLA compliance; hence, Utilization is weighted second highest. Makespan reflects the system's overall processing efficiency but is less critical than the other two factors; its weight is set to the minimum. All subsequent experiments are based on this hypothetical scenario.

3.2. Multi-Objective Optimization Performance Evaluation

To visually compare the final performance of the four algorithms, we summarize their average performance metrics after 200 episodes in Table 1. CPI (Comprehensive Performance Index) is a weighted score where a lower value indicates better overall algorithm performance. For other metrics: Tardiness is better when total delay is lower; Utilization is better when average utilization is higher; and Makespan is better when maximum completion time is lower.

Table 1 and Figure 2 present the average performance of each algorithm after 200 iterations. Experimental data indicate that MoSCO performs best across all metrics. Specifically, MoSCO's CPI is 0.2698, which is 11.5% lower than PureD3QN and 16.0% lower than H-SCSO, demonstrating that the "knowledge injection" mechanism effectively improves solution quality. Regarding individual metrics, MoSCO not only reduced Makespan to 528 but also increased cluster resource utilization to 92.2%, significantly outperforming the worst-performing NSGA-II, which achieved only 86.9% utilization. This indicates that the algorithm did not sacrifice any single objective but successfully identified a higher-quality Pareto equilibrium point. In summary, the MoSCO algorithm does not merely sacrifice one objective to improve another. Instead, it achieves optimal or joint-optimal performance across all three key metrics, demonstrating its robust multi-objective balancing capability.

The improved CPI of MoSCO is due to its hybrid design. While traditional methods like NSGA-II depend heavily on stochastic operators, and PureD3QN may struggle with fine-grained policy adjustments, MoSCO uses the D3QN output as an initial elite individual. This "warm start" mechanism guides the search earlier toward potential Pareto-optimal regions, reducing the time spent exploring low-quality solutions and helping to achieve balanced improvements across all three metrics.

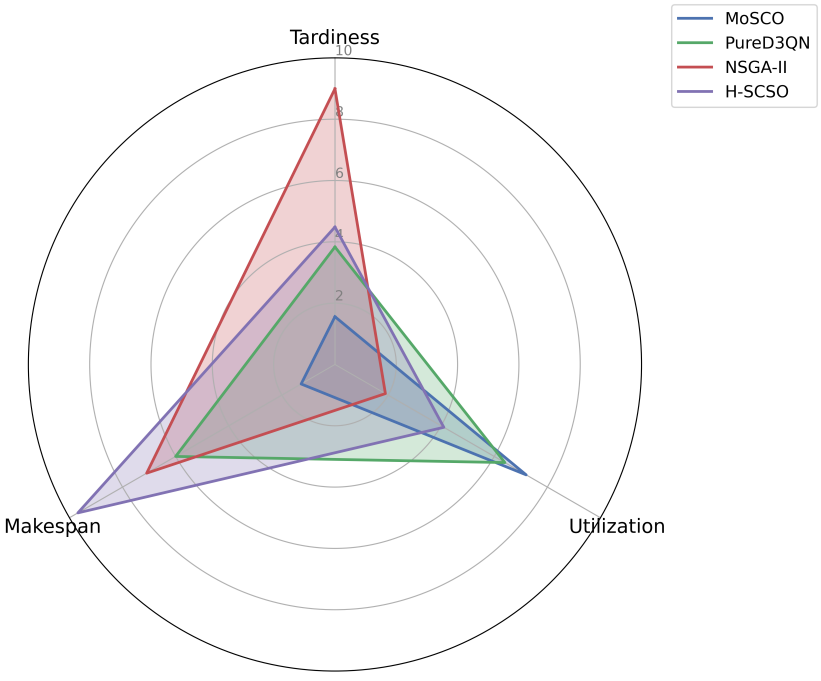


Figure 2. Radar chart comparing sub-objective performance of various algorithms.

Table 1. Performance comparison of different algorithms.

Algorithms	Tardiness	Utilization	Makespan	CPI
MoSCO	4187	0.922	528	0.2698
PureD3QN	4604	0.914	632	0.3053
NSGA-II	5079	0.869	656	0.3227
H-SCSO	4538	0.891	713	0.3576

3.3. Dynamic Convergence Analysis of Tardiness

Figure 3 illustrates the scatter plot distribution of total tardiness across 200 episodes for four algorithms. It is evident that the traditional NSGA-II algorithm, which lacks guidance from prior knowledge, exhibits highly blind initial search behavior and severe global fluctuations. PureD3QN and H-SCSO later reveal bottlenecks: PureD3QN struggles to escape local minima, while H-SCSO exhibits a discrete solution distribution. In contrast, the proposed MoSCO demonstrates significant advantages. Benefiting from D3QN's knowledge injection, MoSCO avoids inefficient exploration in the early stages. During the late iteration phase, its scatter points not only dip below 4500 but also converge with the densest distribution. This visually demonstrates MoSCO's ability to effectively integrate macro-level guidance with fine-grained local exploration, achieving optimal performance in both solution quality and optimization stability.

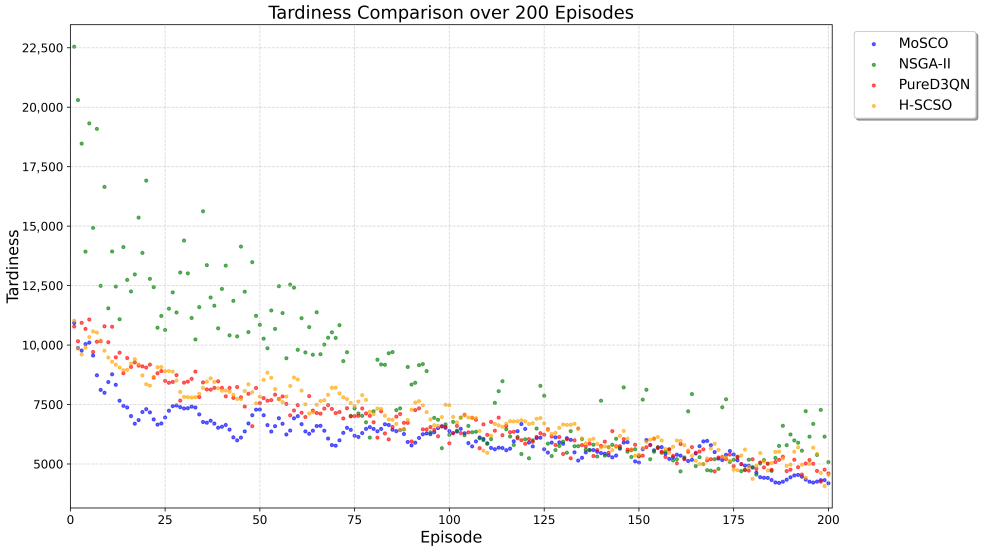


Figure 3. Scatter plots of the Tardiness for the four algorithms.

Minimizing Tardiness requires appropriately prioritizing urgent tasks in dynamic queues. MoSCO's convergence behavior is partly facilitated by the state representations provided to the D3QN network, particularly the proportion of urgent tasks (P_{due}). This allows the agent to recognize delay-sensitive tasks during initial scheduling. The local attack mechanism in SCSO further refines this sequence, helping mitigate scheduling fragmentation that can lead to cascading delays in baseline algorithms.

Figure 4 illustrates the dynamic convergence process of four algorithms toward the average resource utilization. It is evident that PureD3QN lacking heuristic micro-optimization, struggles to perform fine-grained adjustments in complex combination allocations. Despite achieving an average utilization of around 0.9, its optimization trajectory exhibits higher variance and a slower stabilization speed compared to MoSCO, highlighting the limitations of value-driven deep reinforcement learning models in large discrete spaces. NSGA-II exhibits extreme instability, experiencing a sharp decline in performance mid-iteration.

While H-SCSO maintains an acceptable overall level, its data points are loosely scattered with noticeable oscillations. In contrast, the proposed MoSCO algorithm demonstrates overwhelming superiority: it rapidly surpasses 0.85 in the early iterations and converges stably and densely within the high range of from 0.90 to 0.95+ in the later stages. This demonstrates that MoSCO's hybrid architecture significantly enhances the compactness of task mapping, effectively escapes local oscillations, and maximizes resource utilization for cloud service providers.

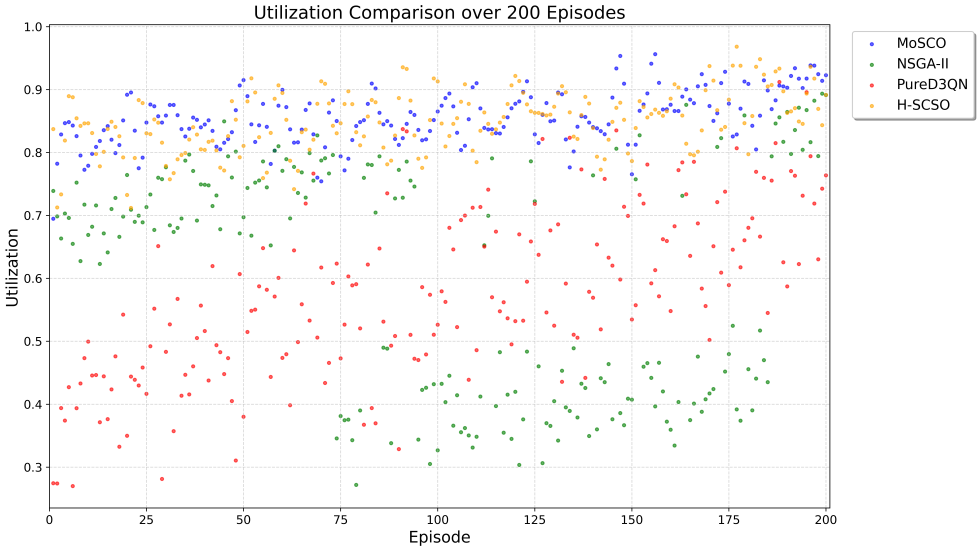


Figure 4. Scatter plots of the Utilization for the four algorithms.

3.4. Dynamic Convergence Analysis of Utilization

Maximizing resource utilization in heterogeneous clouds remains a complex combinatorial problem. PureD3QN's difficulty in maintaining high utilization suggests that pure value-based DRL models may struggle to navigate large discrete assignment spaces efficiently. MoSCO addresses this by incorporating the evolutionary generation and spatial perturbation operations of the SCSO module. Even if the D3QN's initial policy contains fragmented assignments, SCSO explores alternative allocations around this baseline, aiming to find more compact task arrangements that keep virtual machines active.

3.5. Dynamic Convergence Analysis of Makespan

Figure 5 illustrates the convergence of various algorithms in the scatter plot for Makespan. Observation reveals that traditional NSGA-II exhibits extreme blindness in its initial search iterations and exhibits highly volatile optimization trajectories. While PureD3QN achieves a steady decline, its convergence rate significantly slows in the later stages, struggling to break through the local bottleneck around 600. H-SCSO similarly becomes stuck near 700, exhibiting loose oscillations in solution distribution. In contrast, the proposed MoSCO algorithm, leveraging high-quality guidance from D3QN, keeps Makespan low from the outset. In the late iteration phase, MoSCO demonstrates exceptional sustained optimization capability, with solutions densely and stably distributed between 500 and 650. This clearly demonstrates that MoSCO not only efficiently avoids local extremum traps but also identifies more compact task-scheduling sequences, significantly enhancing the system's overall processing throughput.

The plateauing effect seen in PureD3QN and H-SCSO may indicate premature convergence into local optima. Reducing the overall Makespan often requires balancing the global load across heterogeneous nodes, a trade-off that pure learning models can find difficult. MoSCO attempts to mitigate this by employing a global exploration operator controlled

by the adaptive sensitivity parameter (R_t). By applying standard normal-distribution perturbations to elite individuals, the algorithm is better positioned to escape local traps and identify execution sequences that reduce the overall Makespan.

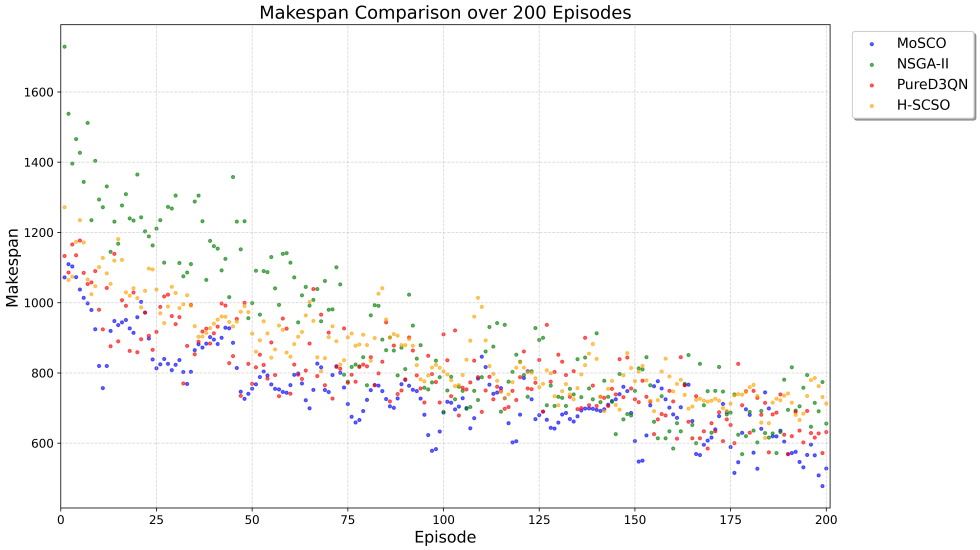


Figure 5. Scatter plots of the Makespan for the four algorithms.

3.6. Multi-Objective Trade-Off and Pareto Front Analysis

To intuitively evaluate the algorithms' overall search capabilities in solving multi-objective cloud scheduling problems, we extracted and visualized non-dominated solutions, which represent the Pareto front approximations plotted by MoSCO, NSGA-II, and H-SCSO at the end of each iteration. As shown by the 3D Pareto surface in Figure 6 and the 2D multidimensional projection in Figure 7, the distribution of solutions exhibits a discrete, stair-step pattern. This is fully consistent with the discrete combinatorial nature of real-world cloud task scheduling.

As shown in the two-dimensional projection plots, the proposed MoSCO algorithm demonstrates highly competitive performance compared to baseline algorithms. In the multi-objective trade-off space, the solutions found by MoSCO—indicated by red markers—typically lie in more favorable regions, such as the lower-left region in the "Total Duration vs. Delay" projection. Although algorithms such as H-SCSO and NSGA-II often face severe trade-offs—such as sacrificing task delays to achieve an acceptable total duration or reducing resource utilization—MoSCO demonstrates more robust and balanced optimization capabilities. By keeping delays below 4600 and limiting the total duration to a reasonable range, MoSCO still maintains a system resource utilization rate exceeding 87%. These visualizations demonstrate that integrating the D3QN network for state awareness and cluster initialization effectively reduces the risk of becoming stuck in local optima, thereby generating a high-quality Pareto front.

The discrete, step-like distribution of Pareto front approximations naturally reflects the indivisibility of cloud tasks and the heterogeneous capacities of virtual machines. The "crowding distance" mechanism employed by MoSCO during the Pareto elite phase helps identify solutions within the favorable trade-off region. By prioritizing solutions with sparse distributions, MoSCO helps prevent the solution set from converging to a single extreme, thereby providing diverse scheduling options to balance operational costs with SLA requirements.

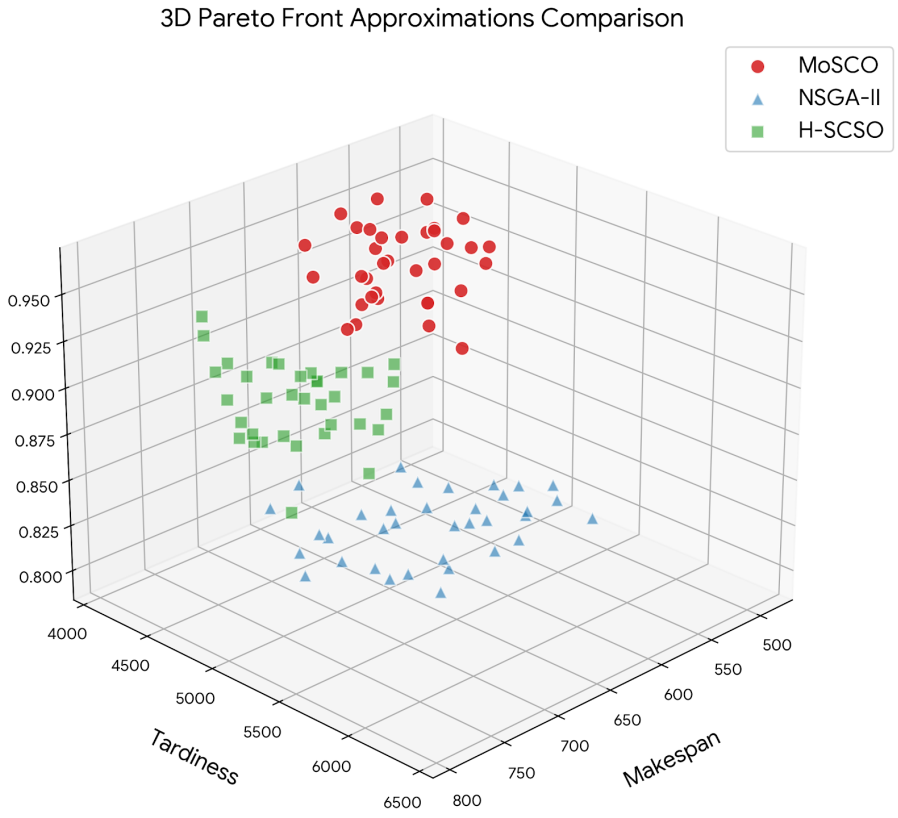


Figure 6. Comparison of 3D Pareto front approximations generated by MoSCO, NSGA-II, and H-SCSO.

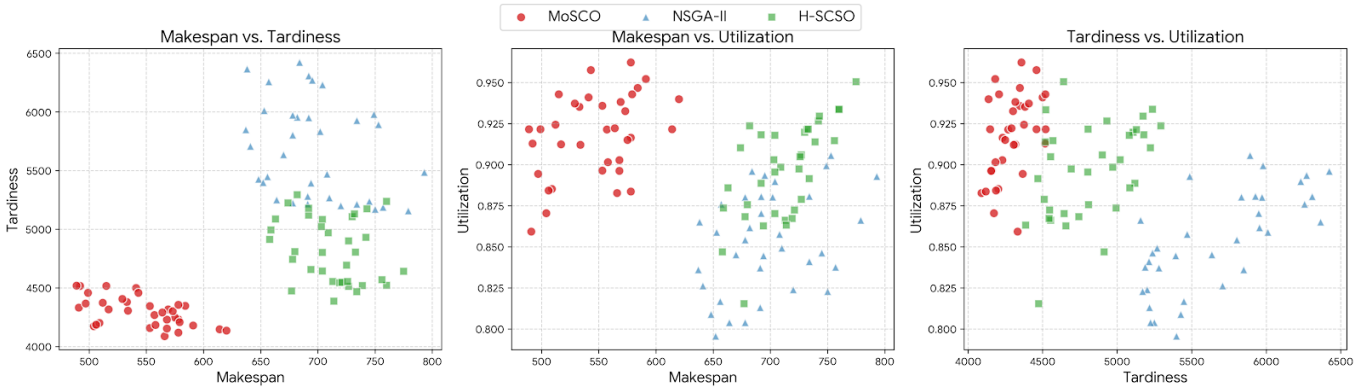


Figure 7. 2D multidimensional projections of the Pareto fronts illustrating the trade-offs among Makespan, Tardiness, and Resource Utilization.

3.7. Stability Analysis

To validate the algorithm's robustness, Figure 8 displays the distribution of the CPI across 50 independent experiments.

As shown in Figure 8, MoSCO features the lowest box position and narrowest box height. Its scatter points are extremely compact, indicating that the algorithm is insensitive to initial random seeds and can converge stably to high-quality solution regions across different runs, with minimal performance fluctuations.

In contrast, although PureD3QN outperforms H-SCSO in median performance, its box plot shows a wider range and more dispersed data points. This visually reflects the instability inherent in pure DRL models during exploration, which tends to generate scheduling policies with high variance. H-SCSO and NSGA-II exhibit higher CPI values and broader

distributions, confirming that traditional meta-heuristic algorithms, lacking prior guidance, tend to become stuck in local optima and produce solutions of inconsistent quality.

The narrower interquartile range and compact data distribution for MoSCO suggest that the "guidance-optimization-feedback" loop improves algorithmic stability. The higher variance observed in PureD3QN often stems from exploration noise, while variance in meta-heuristics typically arises from random initialization. MoSCO seeks to mitigate these issues by using D3QN to provide a more stable initial starting point, while the sorting and local attack procedures of SCSO iteratively refine these solutions, helping to smooth out the fluctuations often seen in pure DRL approaches.

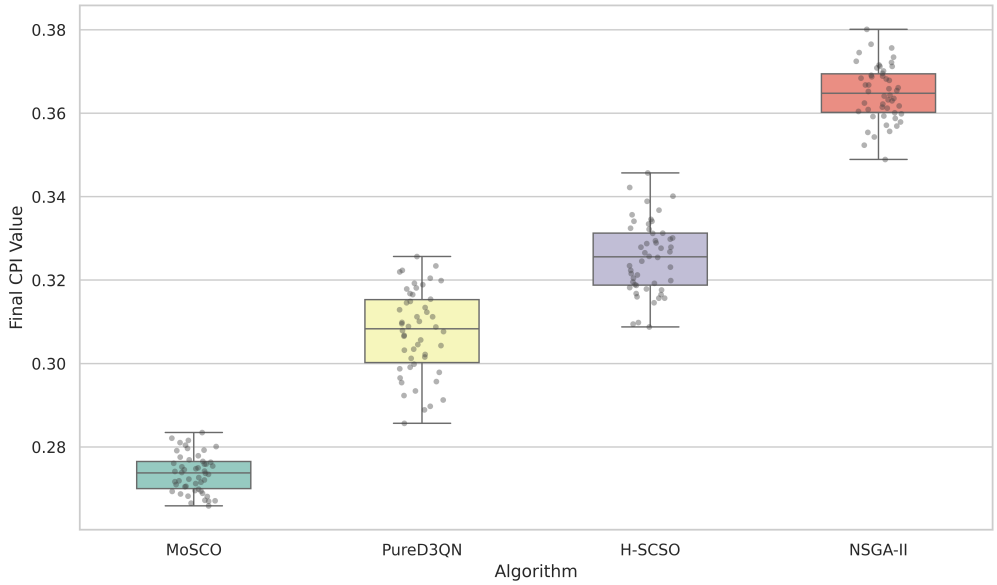


Figure 8. Distribution of scheduling performance metrics for MoSCO and baseline algorithms over 50 independent runs.

4. Discussion

The experimental results suggest that integrating deep reinforcement learning with swarm intelligence can help mitigate the limitations often observed in isolated scheduling algorithms. The improved performance of MoSCO across the evaluated metrics appears to stem from its "knowledge injection" mechanism. By using the D3QN agent to provide an initial elite action distribution, the algorithm can begin its search in a more promising region of the solution space, thereby reducing the initial blind exploration typical of traditional metaheuristics like NSGA-II. Additionally, the population-based iterative process of SCSO serves as a complementary search mechanism, helping the DRL agent escape local optima in later iterations.

In practical cloud environments, balancing conflicting goals can be quite difficult. The distinct distribution of Pareto fronts produced by MoSCO yields a set of non-dominated policies, giving cloud service providers flexible options for controlling operational costs while meeting SLA constraints. The results demonstrate that MoSCO can sustain a high resource utilization rate, averaging 92.20%, while keeping Makespan and Tardiness within competitive limits. This ability is especially important in dynamic environments where task arrival rates and resource availability vary.

Despite these promising results, the proposed hybrid architecture introduces certain computational trade-offs. Offline training of the D3QN network and maintaining experience replay buffers incur additional computational overhead compared to simpler heuristic rules. While this overhead may be acceptable in scenarios with longer task execution times,

it remains a factor to consider for highly time-sensitive applications. Future research could explore the scalability of this framework in more complex edge-cloud environments or integrate energy-aware constraints to support sustainable green computing.

5. Conclusions

This paper presents MoSCO, a hybrid multi-objective scheduling algorithm that integrates deep reinforcement learning with swarm intelligence for dynamic cloud environments. By incorporating DRL-guided initialization into the meta-heuristic optimization process, the proposed method aims to alleviate the slow convergence of traditional swarm algorithms and the instability often seen in pure DRL models. Experimental results indicate that MoSCO achieves competitive performance, increasing the average resource utilization to 92.20% while reducing the maximum Makespan and Tardiness to 528 and 4187, respectively.

However, this study has certain limitations that should be acknowledged. First, the hybrid framework introduces additional computational overhead for training and querying the D3QN network, which may pose challenges in strictly time-sensitive scheduling scenarios. Second, the study could benefit from a more granular ablation analysis regarding the algorithm's sensitivity to individual state features and reward weight combinations. Third, the current performance evaluations are based on a simulated cloud computing environment with a bounded scale. The scalability of MoSCO under extremely large-scale workloads—which often exacerbate the combinatorial curse of dimensionality—as well as its robustness on real-world cloud platforms characterized by unpredictable network latency and complex hardware fault cascades, remains to be empirically validated.

Therefore, future research will focus on addressing these limitations. Primary directions include conducting a fine-grained parameter sensitivity analysis and exploring lightweight neural network architectures to reduce computational overhead. Furthermore, evaluating the algorithm's convergence boundaries under massive-scale workloads and integrating more advanced state-representation architectures, such as Graph Neural Networks (GNNs), are critical next steps. Finally, deploying the MoSCO framework on physical cloud testbeds, extending it to accommodate energy-aware constraints in edge-cloud environments, and conducting comparative analyses with advanced swarm algorithms like MOPSO will further validate its real-world viability.

Author Contributions: Conceptualization, M.S. and H.Z.; methodology, J.W. and M.S.; software, Y.G. and M.S.; validation, H.Z. and M.S.; formal analysis, M.S.; investigation, M.S.; resources, Y.G.; data curation, J.W.; writing—original draft preparation, M.S. and H.Z.; writing—review and editing, M.S. and H.Z.; visualization, Y.G.; supervision, J.W.; project administration. Y.G.; funding acquisition, H.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the following projects: Shandong Provincial Key Research and Development Program (2023CXGC011101), Major Innovation Project of the Science-Education-Industry Integration Pilot Program at Qilu University of Technology (Shandong Academy of Sciences) (2024ZDZX08), Taishan Scholars Program: NO.tspd20240814 and the "20 Articles" Project for Universities in Jinan (202534093).

Data Availability Statement: The datasets presented in this article are not readily available because the data are part of an ongoing state-funded research project and are bound by institutional confidentiality policies. Requests to access the datasets should be directed to Hu Zhang (zhanghu@sdas.org).

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

QoS	Quality-of-service
MoSCO	D3QN-Guided Multi-Objective Sand Cat Swarm Optimization
SCSO	Sand Cat Swarm Optimization
H-SCSO	Heuristic Sand Cat Swarm Optimization
NSGA	Non-dominated Sorting Genetic Algorithm
DRL	Deep Reinforcement Learning
D3QN	Dueling Double Deep Q-Network
TD	Temporal Difference
CPI	Comprehensive Performance Index

References

1. Baloni, D.; Bhatt, C.; Kumar, S.; Patel, P.; Singh, T. The Evolution of Virtualization and Cloud Computing in the Modern Computer Era. In Proceedings of the 2023 International Conference on Communication, Security and Artificial Intelligence (ICCSAI), Greater Noida, India, 23–25 November 2023; pp. 625–630. [CrossRef]
2. Abraham, O.L.; Ngadi, M.A.B.; Sharif, J.B.M.; Sidik, M.K.M. Multi-Objective Optimization Techniques in Cloud Task Scheduling: A Systematic Literature Review. *IEEE Access* **2025**, *13*, 12255–12291. [CrossRef]
3. Mansouri, N.; Ghafari, R.; Zade, B.M.H. Cloud computing simulators: A comprehensive review. *Simul. Model. Pract. Theory* **2020**, *104*, 102144. [CrossRef]
4. Arunarani, A.R.; Manjula, D.; Sugumaran, V. Task scheduling techniques in cloud computing: A literature survey. *Future Gener. Comput. Syst.* **2019**, *91*, 407–415. [CrossRef]
5. Behera, I.; Sobhanayak, S. Task scheduling optimization in heterogeneous cloud computing environments: A hybrid GA-GWO approach. *J. Parallel Distrib. Comput.* **2024**, *183*, 104766. [CrossRef]
6. Chatterjee, M.; Setua, S.K. A multi-objective deadline-constrained task scheduling algorithm with guaranteed performance in load balancing on heterogeneous networks. *SN Comput. Sci.* **2021**, *2*, 361. [CrossRef]
7. Mahmoud, H.; Thabet, M.; Khafagy, M.H.; Omara, F.A. Multiobjective task scheduling in cloud environment using decision tree algorithm. *IEEE Access* **2022**, *10*, 10266–10283. [CrossRef]
8. Cui, Z.; Zhao, T.; Wu, L.; Qin, A.K.; Li, J. Multi-objective cloud task scheduling optimization based on evolutionary multi-factor algorithm. *IEEE Trans. Cloud Comput.* **2023**, *11*, 3685–3699. [CrossRef]
9. Devi, K.L.; Valli, S. Multi-objective heuristics algorithm for dynamic resource scheduling in the cloud computing environment. *J. Supercomput.* **2021**, *77*, 8252–8280. [CrossRef]
10. Mishra, A.K.; Mohapatra, S.; Sahu, P.K. Adaptive Tasmanian Devil Optimization algorithm based efficient task scheduling for big data application in a cloud computing environment. *Multimed. Tools Appl.* **2025**, *84*, 26977–26996. [CrossRef]
11. Pradhan, R.; Satapathy, S.C. Energy-Aware Cloud Task Scheduling algorithm in heterogeneous multi-cloud environment. *Intell. Decis. Technol.* **2022**, *16*, 279–284. [CrossRef]
12. Zhou, G.; Tian, W.; Buyya, R. Multi-search-routes-based methods for minimizing makespan of homogeneous and heterogeneous resources in Cloud computing. *Future Gener. Comput. Syst.* **2023**, *141*, 414–432. [CrossRef]
13. Hemanth, S.V.; Kirubha, D.; Reddy, S.R.; Chelladurai, T.; Soundari, A.G.; Amirthayogam, G. Multi objective Ant Colony Optimization Technique for Task Scheduling in Cloud Computing. In Proceedings of the 2024 3rd International Conference on Applied Artificial Intelligence and Computing (ICAAIC), Salem, India, 4–6 July 2024; pp. 830–835.
14. Malti, A.N.; Hakem, M.; Benmammar, B. A new hybrid multi-objective optimization algorithm for task scheduling in cloud systems. *Clust. Comput.* **2024**, *27*, 2525–2548. [CrossRef]
15. Amer, D.A.; Attiya, G.; Ziedan, I. An efficient multi-objective scheduling algorithm based on spider monkey and ant colony optimization in cloud computing. *Clust. Comput.* **2024**, *27*, 1799–1819. [CrossRef]
16. Akopov, A.S. A Hybrid Multi-Swarm Particle Swarm Optimization Algorithm for Solving Agent-Based Epidemiological Model. *Cybern. Inf. Technol.* **2025**, *25*, 59–77. [CrossRef]
17. Afrasyabi, P.; Mesgari, M.S.; El-sayed, M.; Kaveh, M.; Ibrahim, A.; Khodadadi, N. A Crossover-Based Multi-Objective Discrete Particle Swarm Optimization Model for Solving Multi-Modal Routing Problems. *Decis. Anal. J.* **2023**, *9*, 100356. [CrossRef]
18. Akopov, A.S.; Beklaryan, L.A. Evolutionary Synthesis of High-Capacity Reconfigurable Multilayer Road Networks Using a Multiagent Hybrid Clustering-Assisted Genetic Algorithm. *IEEE Access* **2025**, *13*, 53448–53474. [CrossRef]
19. Hao, Y.; Zhao, C.; Li, Z.; Si, B.; Unger, H. A learning and evolution-based intelligence algorithm for multi-objective heterogeneous cloud scheduling optimization. *Knowl.-Based Syst.* **2024**, *286*, 111366. [CrossRef]

20. Mangalampalli, S.S.; Karri, G.R.; Mohanty, S.N.; Ali, S.; Khan, M.I.; Abdullaev, S.; AlQahtani, S.A. Multi-objective Prioritized Task Scheduler using improved Asynchronous advantage actor critic (a3c) algorithm in multi cloud environment. *IEEE Access* **2024**, *12*, 11354–11377. [CrossRef]
21. Mangalampalli, S.; Karri, G.R.; Kumar, M.; Khalaf, O.I.; Romero, C.A.T.; Sahib, G.A. DRLBTSA: Deep Reinforcement Learning Based Task-Scheduling Algorithm in Cloud Computing. *Multimed. Tools Appl.* **2024**, *83*, 8359–8387. [CrossRef]
22. Fan, W.; Chun, X.; Fan, Z.; Zhang, R.; Liu, S.; Liu, Y. Dual-Agent DRL-Based Service Placement, Task Scheduling, and Resource Allocation for Multi-Sensor and Multi-User Edge Computing Networks. *IEEE Trans. Netw. Sci. Eng.* **2025**, *12*, 3416–3433. [CrossRef]
23. Cui, D.; Peng, Z.; Li, K.; Li, Q.; He, J.; Deng, X. An Novel Cloud Task Scheduling Framework Using Hierarchical Deep Reinforcement Learning for Cloud Computing. *PLoS ONE* **2025**, *20*, e0329669. [CrossRef]
24. Zhang, M.; Wang, D.; Cai, Z.; Huang, Y.; Yu, H.; Qin, H.; Zeng, J. EGLight: Enhancing deep reinforcement learning with expert guidance for traffic signal control. *Transp. A Transp. Sci.* **2025**, 1–27. [CrossRef]
25. Wang, Z.; Goudarzi, M.; Buyya, R. TF-DDRL: A transformer-enhanced distributed DRL technique for scheduling IoT applications in edge and cloud computing environments. *IEEE Trans. Serv. Comput.* **2025**, *18*, 1039–1053. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.