LLM-based cost-aware task scheduling for cloud computing systems

Journal of Cloud Computing

RESEARCH Open Access

LLM-based cost-aware task scheduling for cloud computing systems

Haoran Pei[1], Yan Gu[1], Yajuan Sun[2*], Qingle Wang[1], Cong Liu[3], Xiaomin Chen[4] and Long Cheng[1,2]

Abstract
Cloud task scheduling faces significant challenges due to resource heterogeneity, conflicting optimization objectives, and dynamic workload fluctuations. Traditional heuristic algorithms often necessitate comprehensive knowledge of environmental parameters, significantly constraining their efficacy in dynamic cloud computing environments. While Deep Reinforcement Learning (DRL) methods have shown promise in intelligent scheduling via continuous environment interaction, they suffer from limited generalization to diverse cloud scenarios and lack decision interpretability. To address these shortcomings, this paper proposes LarS, a scheduling framework that employs Large Language Models (LLMs) as high-level decision agents for cloud task scheduling. In LarS, DRL agents trained in carefully chosen representative cloud environments generate a high-quality dataset of scheduling decisions, which is used to fine-tune an LLM. By jointly optimizing average response time, task success rate, and average rental cost, LarS achieves strong generalization across heterogeneous cloud deployments. Experimental results demonstrate that LarS surpasses current approaches in average response time, success rate, and average cost, and maintains strong generalization performance under varied experimental settings.

Keywords Cloud computing, Task scheduling, Deep reinforcement learning, Large language models

Introduction

Cloud computing has emerged as a transformative paradigm that enables a wide range of users, including enterprises and individual developers, to access a shared pool of configurable computing resources such as servers, storage, and networks on an on-demand basis. These resources are dynamically allocated and virtualized, allowing for scalable, cost-effective, and flexible service delivery. With its inherent advantages in elasticity, reliability, scalability, and sustainability, cloud computing not only reduces capital and operational costs but also significantly simplifies system management and maintenance [1]. As a result, an increasing number of users are choosing to deploy their applications and services in cloud environments.]. As a result, an increasing number of users are choosing to deploy their applications and services in cloud environments.

Task scheduling plays a central role in cloud computing, as it determines how computational workloads are allocated across virtualized resources. An effective scheduling mechanism must maximize resource utilization, reduce operational expenses, and improve diverse Quality of Service (QoS) [2, 3]. However, meeting these objectives is complicated by the inherently dynamic and heterogeneous nature of cloud environments, which comprise virtual machines (VMs) with varying processing capabilities, fluctuating cost models, and irregular

*Correspondence:
Yajuan Sun
syj@ncepu.edu.cn
[1]School of Control and Computer Engineering, North China Electric Power University, Beijing, China
[2]Network and Information Office, North China Electric Power University, Beijing, China
[3]NOVA Information Management School, Nova University of Lisbon, Lisbon, Portugal
[4]Department of Computer Science, University of Reading, Reading, UK

task arrival patterns [4–6]. Moreover, such variability often leads to resource contention and uneven load distribution, degrading system responsiveness and service reliability [2, 7]. Therefore, it is imperative to develop intelligent and adaptive scheduling strategies to ensure efficient and dependable cloud service delivery.

Traditional scheduling approaches, including heuristic and rule-based methods, frequently fall short in dynamic environments due to their inherent rigidity and inability to adapt swiftly to evolving workloads or unexpected changes in resource states [4]. Recently, deep reinforcement learning (DRL) has emerged as a promising alternative by enabling adaptive scheduling decisions through interactions with the environment [8, 9]. DRL-based scheduling has demonstrated substantial advantages, automatically learning near-optimal strategies to significantly improve metrics such as makespan, operational cost, and resource utilization compared to traditional heuristics [10, 11]. Despite these successes, DRL still faces critical limitations. First, training DRL models typically requires extensive computational resources and large volumes of interaction data, making it computationally expensive and impractical for rapid deployment [4]. Second, DRL schedulers are susceptible to overfitting to specific training scenarios, severely restricting their generalization to new or evolving environments [9]. Third, the performance of DRL models heavily depends on manually crafted reward functions, and inappropriate reward designs can lead to convergence to suboptimal solutions or slow convergence, limiting real-world applicability.

Recent advancements in large language models (LLMs) offer a complementary solution. Pretrained on vast text corpora, LLMs provide powerful semantic reasoning and deep contextual insight, enabling them to generate human-like heuristics and explanatory guidance for complex scheduling decisions [12–14]. However, standalone LLM-based scheduling solutions also face several challenges: their outputs may sometimes lack reliability, producing inaccurate or contextually inappropriate recommendations, thus undermining scheduling robustness [15]. Moreover, directly translating linguistic reasoning to actionable scheduling decisions or quantitative rewards remains non-trivial, limiting the direct applicability of purely LLM-based approaches [12, 13].

To address these challenges, we propose LarS, a cloud task scheduling framework that leverages an LLM as a high-level decision agent. In LarS, GPT-4o generates scheduling decisions with reasoning trajectories for given environments and states. Trained DRL agents evaluate these trajectories, and only the validated ones are retained to form a high-quality dataset. This dataset is then used to fine-tune the LLM via LoRA, enhancing its generalization capability and enabling optimization of cost and QoS across diverse cloud environments. In summary, the key contributions of this paper are as follows:

- We introduce LarS, an efficient framework that integrates LLM-based reasoning with DRL-based verification for intelligent cloud task scheduling.
- We propose a hybrid data generation pipeline where GPT-4o produces reasoning trajectories and DRL agents serve as evaluators to curate high-quality supervision data.
- We present the detailed design and implementation of LarS, and experimental results show that it outperforms existing approaches while maintaining strong generalization across diverse settings.

The remainder of this paper is organized as follows. Section "Related work" reviews related work on cloud scheduling methods. Section "System model and problem formulation" describes the system models and optimization objectives. Section "The proposed LarS" details the design and implementation of LarS. Section "Experiments evaluation" presents the experimental evaluation of LarS's performance, and Section "Conclusion" concludes the paper.

Related work

Conventional methods for cloud task scheduling

Traditional scheduling approaches in cloud computing rely heavily on heuristic and meta-heuristic algorithms to find near-optimal solutions within reasonable time. Evolutionary algorithms and swarm intelligence techniques are prominent in this domain. For instance, Ismayilov and Topcuoglu propose a dynamic workflow scheduling method using a neural-network enhanced evolutionary algorithm to handle multiple objectives under changing conditions [16]. Similarly, Shukri et al. introduce an enhanced Multi-Verse Optimizer that significantly improves task scheduling performance in terms of makespan and resource utilization [17]. On the swarm intelligence side, researchers have leveraged algorithms like Particle Swarm Optimization and Whale Optimization. Nabi et al. present an adaptive PSO-based scheduling approach (AdPSO) which dynamically adjusts to workload changes, achieving better load balancing and reducing completion time [18]. Mangalampalli et al. propose a trust-aware task scheduler based on the Whale Optimization algorithm to jointly minimize execution time and SLA violations, demonstrating superior results over basic heuristics in cloud environments [19]. While these specialized solutions can optimize particular objectives, they typically focus on a restricted set of requirements; consequently, their performance may deteriorate when the cloud environment diverges from

expected conditions or when additional objectives must be incorporated.

DRL methods for cloud task scheduling

Deep reinforcement learning (DRL) has emerged as a promising approach for cloud task scheduling due to its ability to simultaneously optimize multiple objectives, including cost efficiency, makespan minimization, and QoS compliance [20]. Instead of relying on predefined heuristic rules, DRL-based schedulers can adapt to complex dynamics and optimize long-term rewards (such as response time or cost). Siddesha et al. propose a DRL scheme for cloud scheduling that learns to allocate tasks to VMs, yielding improvements in makespan and energy consumption compared to traditional algorithms [21]. In a similar vein, Islam et al. leverage deep Q-learning techniques to develop a scheduling policy for Spark jobs in cloud computing, achieving both performance gains and cost efficiency over baseline scheduling strategies [22]. Advanced variants of DRL have also been explored; for example, Xiu et al. introduce a meta-reinforcement learning framework (MRLCC) that enables a scheduler to quickly adapt to new cloud environments by learning a meta-policy, resulting in higher sample efficiency and robust performance across varying conditions [23].

These works illustrate that DRL approaches can dynamically learn from the cloud system's state and feedback, often outperforming static heuristics especially in large-scale or non-stationary cloud scenarios. However, current DRL approaches face several limitations. These include limited generalization across diverse scenarios, computationally expensive training procedures, and policies that are susceptible to overfitting. As a result, retraining is necessary when workload patterns or cloud configurations change [4]. Furthermore, DRL models often exhibit inadequate explainability, unpredictable worst-case behavior, and difficulties in optimally balancing multiple competing objectives, highlighting the need for more adaptive and robust scheduling paradigms that extend beyond conventional DRL methods.

Table 1 Summary of cloud task scheduling methods and main features

Reference	Method	Success rate	Generalization	Interpretability
[22]	DQN + Policy Gradient	✓	-	-
[18]	Adaptive PSO	✓	-	-
[21]	DQN + LSTM	✓	-	-
[20]	Deep Q-Learning	✓	-	-
[17]	Enhanced Multi-Verse Optimizer	-	-	-
[16]	NN-based Evolutionary Algorithm	✓	✓	-
[19]	Whale Optimization	✓	-	-
[23]	Meta Reinforcement Learning	✓	-	-
[14]	LLM-guided SARSA	-	-	✓
[24]	LLM-assisted RL	✓	-	-
LarS	**DQN + LLM**	✓	✓	✓

Large language models for cloud task scheduling

The rapid advancement of LLMs has created opportunities for addressing complex optimization problems, including scheduling tasks, by leveraging their powerful sequence modeling and reasoning capabilities. Recent studies demonstrate that LLMs, pretrained on extensive corpora, can effectively learn intricate scheduling constraints and objectives. For example, Abgaryan et al. [25] demonstrated that with minimal fine-tuning techniques like LoRA, LLMs achieve competitive performance on static job shop scheduling problems. Krishnamurthy and Shiva propose an LLM-guided approach using a SARSA reinforcement learning agent for dynamic task scheduling in the cloud [14]. Similarly, Tang et al. [24] developed a scheduling expert dataset to fine-tune a lightweight LLM for task assignment decisions in multi-cloud environments, showing that LLM-based agents can learn effective scheduling policies from expert demonstrations. However, current LLM-based schedulers primarily operate in offline or semi-static contexts, providing heuristic guidance or refining existing solutions rather than participating in the continuous, real-time decision-making required for dynamic cloud environments [26].

We summarize the related works mentioned above in Table 1. While task scheduling has been extensively studied, traditional heuristic methods often lack flexibility and adaptability to dynamic conditions. DRL-based schedulers, though adaptive, suffer from limited generalization and high computational costs. Existing LLM-driven approaches exhibit strong generalization capabilities, yet they have not fully demonstrated their potential in handling online adaptive scheduling scenarios with streaming workloads and evolving objectives. To remedy these shortcomings, this paper proposes LarS, an effective framework that leverages LLM as cloud task scheduling agent to achieve adaptive, explainable, and efficient cloud task scheduling.

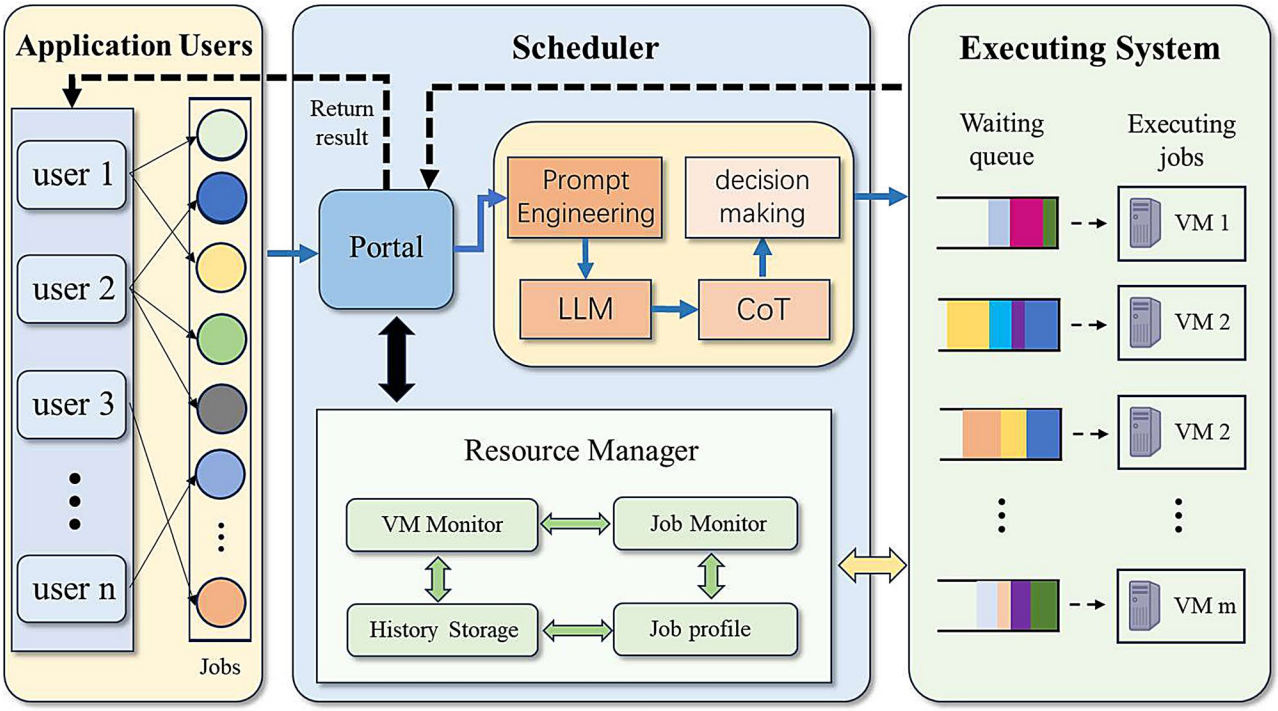

Fig. 1 Task scheduling in cloud computing systems

System model and problem formulation

This section presents the formal mathematical framework underlying our cost-aware cloud job scheduling methodology. We provide definitions of the cloud environment, job characteristics, VM configurations and the job scheduling strategy.

The overall framework

We model a cloud computing environment comprising VMs analogous to commercial IaaS offerings (such as AWS EC2 instances and Google Compute Engine) that operate on a pay-as-you-go pricing model. In this environment, users submit computational jobs to applications hosted on these VMs, while the scheduling system dynamically allocates incoming jobs to suitable VMs for execution.

Fig. 1 illustrates the architecture of our cloud job scheduling framework. Upon job arrival from multiple application users, each job initially enters the scheduling portal, where it undergoes prompt engineering for proper input formatting and parameter extraction. Subsequently, an LLM combined with CoT reasoning executes decision-making to assign each job optimally to an appropriate VM. Each VM maintains a local queue and executes the assigned jobs following a first-come, first-served (FCFS) scheduling policy. The resource manager performs three key functions: processing job metadata, monitoring cloud resource pool status and tracking job execution states. For clarity of presentation, we summarize the key mathematical notations employed in our framework in Table 2.

Table 2 Notations used in our scheduling model

Notation	Meaning
ID_i	The id of the -th job
aT_i	The arrival time of the -th job
$reqCom_i$	The required compute units by job
QoS_i	The QoS requirement by job
$Type_i$	The type of the -th job
T_i^{rep}	The response time of the -th job
T_i^{exe}	The runtime of the -th job
T_i^{wait}	The waiting time of the -th job
VID_i	The id of the -th VM instance
$VCPU_i$	The number of virtual CPUs of VM
$VType_i$	The type of the -th VM instance
VSC_i	The start-up cost of the -th VM instance
VEC_i	The execution cost of VM per time unit

Job model

Our framework models dynamic workloads characterized by unpredictable job arrivals with heterogeneous computational requirements. Formally, we define each job_i through the following parameters:

$$job_i = \{ID_i, aT_i, reqCom_i, QoS_i, Type_i\} \quad (1)$$

where ID_i denotes the job identifier, aT_i represents the arrival timestamp, $reqCom_i$ specifies the required

computational units, QoS_i defines the QoS requirement and $Type_i$ indicates the job classification (I/O-intensive or CPU-intensive).

Virtual machine model

Following the paradigm of commercial cloud providers (such as AWS EC2's memory-optimized and I/O-optimized instances), we characterize each cloud instance's computational capacity by its virtual CPU (vCPU) count. Formally, each VM_j is defined by the following attributes:

$$VM_j = \{VID_j, VCom_j, VCPU_j, VType_j, VSC_j, VEC_j\} \quad (2)$$

where VID_j denotes the VM identifier, $VCom_j$ represents the computational capacity per vCPU, $VCPU_j$ indicates the total vCPU count and $VType_j$ specifies the VMs type (I/O or CPU). The total execution cost for a job on VM_j comprises both a fixed startup cost VSC_j and a time-based execution cost VEC_j.

Problem formulation

The job scheduler dynamically assigns incoming jobs to available VMs. Each job undergoes two phases: queueing time in the VMs waiting queue and processing time on the allocated VM. Formally, the response time T_i^{rep} for job_i is defined as:

$$T_i^{rep} = T_i^{wait} + T_i^{exe} \quad (3)$$

The waiting time T_i^{wait} for job_i in VM_j's queue equals the sum of execution times for all preceding jobs in the queue:

$$T_i^{wait} = \begin{cases} \sum_{k=1}^{n} T_k^{exe}, & \text{if } n > 0 \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

where n denotes the number of jobs preceding job_i in the queue. When the instance's queue is empty, the waiting time becomes zero and the job executes immediately.

The execution time T_i^{exe} of job_i on VM_j is calculated as:

$$T_i^{exe} = \frac{reqCom_i \cdot (Type_i \odot VType_j)}{2VCom_j \cdot VCPU_j} \quad (5)$$

where $reqCom_i$ is the required compute of the job_i, $Type_i$ is the type of the job_i, $VType_j$ is the type of the assigned VM_j, $\odot$ is the type matching operator, $VCPU_j$ is the number of virtual CPUs of the VM_j and $VCom_j$ is the available compute of the VM_j. The type matching operator $\odot$ returns value of 1 when the $Type_i$ matches the $VType_j$, and value of 2 when they do not match, effectively doubling the execution time when there is a type mismatch between the job and VM.

Therefore, the total execution cost $Cost_j$ for processing job_i on VM_j is computed as:

$$Cost_j = VSC_j + VEC_j \cdot T_i^{exe} \quad (6)$$

Optimization objectives

To optimize cloud task scheduling, we establish two key objectives: minimizing total costs and maximizing success rate. We consider the ratio of execution time to response time as QoS. The QoS of job_i is calculated as:

$$\mathrm{QoS}_i = \frac{T_i^{\mathrm{exe}}}{T_i^{\mathrm{rep}}} \quad (7)$$

where T_i^{rep} is the response time of the job and T_i^{exe} is the execution time of the job. The success of a job is measured by the deadline requirements specified by the user:

$$Success_i = \begin{cases} 1, & \text{if } \mathrm{QoS}_i \leq \text{ QoS requirement} \\ 0, & \text{otherwise} \end{cases} \quad (8)$$

The multi-objective optimization problem is formally expressed as:

$$\begin{aligned} &\text{Minimize} \quad \sum_{j=1}^{M} Cost_j \\ &\text{Maximize} \quad SR = \frac{1}{M} \sum_{i=1}^{M} Success_i \end{aligned} \quad (9)$$

where M denotes the total number of jobs in the scheduling process, SR represents the success rate of QoS compliance across all jobs.

The proposed LarS

This section presents LarS, a cloud task scheduling framework that enhances the generalization and interpretability of the scheduling LLM by leveraging DRL-filtered datasets and fine-tuning with LoRA.

Markov decision process formulation

Within our framework, the scheduling problem is formalized as a Markov Decision Process (MDP) with the following components [27]:

State space (𝒮)

In our system, the state space is defined by the combined attributes of tasks and VMs. Accordingly, the system

state at time t, upon the arrival of job job_i, is represented as:

$$S_{t_i} = S_i \cup S_{t_i}^{\text{vm}} \tag{10}$$

where S_i encodes the job-specific features and $S_{t_i}^{\text{vm}}$ encodes the current state of all VMs. Concretely, we set:

$$S_{t_i} = \left[\text{reqCom}_i,\ \text{QoS}_i,\ T_{i1}^{\text{wait}}, \ldots,\ T_{ij}^{\text{wait}},\ \ldots,\ T_{iN}^{\text{wait}}\right] \tag{11}$$

where reqCom_i is the compute requirement of job_i, QoS_i is its QoS metric and T_{ij}^{wait} is the waiting time of job_i in the queue of VM_j.

Action space (𝒜)

The action space $\mathcal{A}$ represents the set of all possible actions available to the agent during the decision-making process. In our scheduling environment, $\mathcal{A}$ corresponds to the set of available VMs that can be selected for each incoming job. Therefore, the action space is defined as:

$$\mathcal{A} = \{\text{VM}_1, \text{VM}_2, \ldots, \text{VM}_N\} \tag{12}$$

where each action corresponds to dispatching the job to a particular VM.

Reward function (ℛ)

The reward is designed to optimize the cost and QoS. Assigning job_i incurs a $Cost_j$ and has QoS_i. The reward is defined as:

$$r = \left(1 + e^{-k\,Cost_i}\right) \text{QoS}_i \tag{13}$$

where $k > 0$ is a hyperparameter used to balance the cost and QoS. As the cost decreases, the reward increases, reflecting a preference for lower-cost objection. Similarly, as QoS_i increases, indicating relatively fewer waiting times and easier to meet QoS requirements, the reward also increases.

Design and implementation of lars

To address the MDP formulation, we propose a two-stage framework (Fig. 2). First, DRL agents are independently trained for distinct cloud scenarios to acquire optimal scheduling policies. Second, these agents filter scheduling trajectories generated by GPT-4o through interaction with the scenarios to create a high-quality dataset

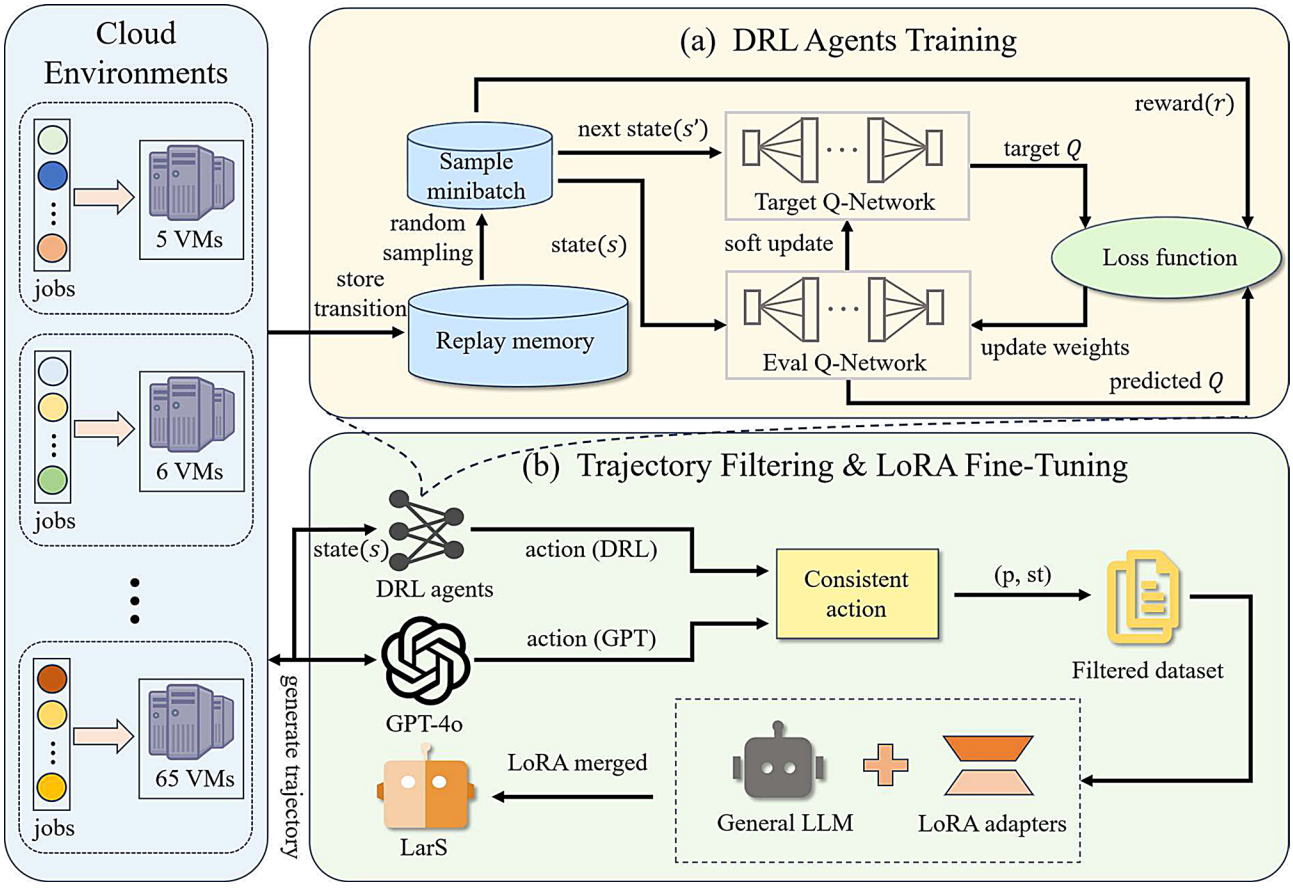

Fig. 2 The proposed LarS architecture

for fine-tuning a lightweight scheduling LLM via LoRA, enabling robust generalization across diverse task scheduling scenarios.

DRL agents training for each scenario

DQN [28] extends traditional Q-learning by employing neural networks to approximate Q-values in high-dimensional state spaces, rendering them particularly suitable for dynamic cloud scheduling applications [29, 30].

While we use the DQN algorithm as the core method, our approach is not limited to a single DQN training. For each cloud task scheduling scenario of varying scale, we train a DRL agent. Each agent is responsible for making scheduling decisions specific to its assigned environment and filters out the scheduling trajectories in that scenario. These scheduling trajectories from different scenarios are aggregated into a larger dataset, which is then used for LoRA fine-tuning of the scheduling LLM. This process allows the scheduling LLM to achieve stronger generalization and robustness across different cloud environments.

The data aggregation process can be expressed as follows:

$$\mathcal{D}_{\text{final}} = \bigcup_{j=1}^{n} \mathcal{D}_j \tag{14}$$

where $\mathcal{D}_j$ represents the dataset of filtered scheduling trajectories from the DRL agent of scenario j, and $\mathcal{D}_{\text{final}}$ is the final dataset aggregated from all scenarios, which is used for LoRA fine-tuning of the scheduling LLM. And the DRL agents' training process is given as follows:

Firstly, we adopt prioritized experience replay [31] to enhance learning efficiency by selectively replaying more informative experiences during training. Specifically, transitions are sampled from the replay buffer with probabilities proportional to their temporal-difference (TD) errors, thus prioritizing experiences that indicate larger deviations between predicted and actual rewards.

Secondly, to stabilize the training of the Q-network and mitigate variance in the Q-value estimations, we maintain a separate target network, updated periodically with parameters derived from the primary evaluation network. This periodic synchronization ensures stable target estimations, significantly enhancing training consistency in dynamic scheduling scenarios. The target value used in training is computed as:

$$y = r_t + \gamma \max_{a\prime} Q(s_{t+1}, a\prime; \theta^-) \tag{15}$$

where γ is the discount factor balancing immediate and long-term rewards, and θ^- represents the target network parameters. The primary network parameters (θ) are optimized by minimizing the mean squared temporal-difference loss:

$$L(\theta) = \mathbb{E}_{i\sim P(i)}\left[(y - Q(s_t, a_t; \theta))^2\right] \tag{16}$$

Lastly, to effectively balance exploration and exploitation during policy learning, we employ a linear epsilon-greedy strategy. In this strategy, the exploration probability ε gradually increases from an initial small value toward a defined maximum, encouraging the agent to systematically explore scheduling decisions initially and progressively shift to exploiting learned policies as training advances. This approach effectively prevents premature convergence and ensures a comprehensive exploration of diverse scheduling strategies.

Trajectory filtering and LLM fine-tuning

DRL agent-guided trajectory filtering: The GPT-4o generates scheduling trajectories based on structured environmental prompts. Concurrently, DRL agents independently produce scheduling decisions for identical states. The system retains only those prompt-trajectory pairs (p, st) where the GPT-4o's proposed action (a_{GPT}) matches the DRL agent's action (a_{DRL}).

LoRA-based fine-tuning of LLaMA-2: The filtered scheduling trajectories are used to fine-tune a lightweight LLM, with the 13-billion-parameter LLaMA-2 causal language model serving as the foundation for the scheduling model, leveraging the parameter-efficient LoRA technique [32]. This method drastically reduces the number of trainable parameters, adapting only two projection matrices per attention head, resulting in less than 1% of total model parameters requiring updates during fine-tuning.

Given a frozen pre-trained weight matrix $W_0 \in \mathbb{R}^{d\times k}$ (such as the query projection matrix in attention layers), LoRA injects a low-rank trainable update defined as:

$$W = W_0 + BA, \quad B \in \mathbb{R}^{d\times r},\ A \in \mathbb{R}^{r\times k},\ r \ll \min(d, k) \tag{17}$$

Initially, we set $B = 0$ and $A = \frac{\alpha}{r} I$, ensuring the fine-tuned model starts with identical behavior to the pre-trained model ($W = W_0$). Throughout the fine-tuning process, only matrices A and B receive gradient updates, while the vast majority of original weights remain unchanged.

To further enhance training efficiency, we apply INT8 quantization prior to LoRA insertion. This converts linear layers' activations to 8-bit precision and maintains FP16 accuracy through freezing LayerNorm statistics. Such quantization reduces GPU memory requirements by approximately fourfold, making single-GPU training feasible.

[Observation]

Current Task (Compute or IO), length, deadline, current time
Multiple VMs (Compute or IO), with different idle times, capacities, and costs

[Task Scenario Description]

A cloud computing scheduler assigns tasks (Compute or IO-intensive) to suitable virtual machines (VMs).

[Task Description]

Which VM should be selected to minimize cost and finish before the deadline?

[Commonsense Knowledge]

Match tasks to the appropriate VM type
Prefer VMs available sooner and with lower cost

[Scheduling Action Space]

All VMs

(a) The prompt template

Step 1: Reasoning

[Task Comprehension]

Assign Compute-heavy tasks to Compute-type VMs.
Prioritize earlier availability and lower cost.

[Scheduling Condition Analysis]

VM 2, 4, and 9 available immediately.
VM 2 is Compute-type, lowest cost , and ready now.
VM 4 and VM 9 have higher cost , despite immediate availability.
VM 7 & 8 available too late , thus unsuitable.

Step 2: Action Execution

Optimal choice is VM ID 2 (Compute, lowest cost, immediate availability).

(b) The reasoning and action execution process.

Fig 3 Prompt template used in LarS and an example of the reasoning process

Given the filtered dataset of prompt-response pairs $(x, y) \in \tau_{\text{filtered}}$, the fine-tuning minimizes the causal language modeling cross-entropy loss defined as:

$$\mathcal{L}_{\text{distill}}(\phi) = \mathbb{E}_{(s,a)\sim\tau_{\text{filtered}}}\left[-\log P_\phi(a|s)\right] \quad (18)$$

where $P_\phi(a|s)$ is the probability distribution over actions produced by the LLM with parameters ϕ. To avoid introducing irrelevant biases into the model, tokens corresponding to the Observation: fields are masked by replacing their token IDs with -100, effectively excluding these from gradient computations.

Through this combined approach, which leverages efficient LoRA fine-tuning and INT8 quantization, we effectively transfer the high-quality trajectories that align with the DRL strategy to the scheduling LLM. The final fine-tuned LLM not only preserves generalization capabilities inherent in LLaMA-2 but also delivers significantly enhanced practical applicability in cloud task scheduling scenarios.

Prompt construction and reasoning strategy

LarS employs a prompt-driven decision-making method. The framework constructs an enriched knowledge prompt that comprehensively encodes the scheduling problem state. Formally, we define:

$$X = \text{Prompt}(o,\ d_{\text{scenario}},\ d_{\text{task}},\ d_{\text{knowledge}},\ A) \quad (19)$$

where o denotes the observed cloud environment state, d_{scenario} characterizes the operational context, d_{task} specifies the scheduling objectives, $d_{\text{knowledge}}$ incorporates relevant domain knowledge, and A represents the action space of scheduling decisions (such as task-to-VM mappings).

The LLM processes the prompt using CoT reasoning to generate scheduling trajectories, performing a two-phase analysis: (1) evaluating the prompt content including VM states, queued tasks and optimization objectives; (2) selecting the optimal task-to-VM mapping. Importantly, the LLM concurrently produces a natural language justification for each decision, enabling human operators to verify and understand the scheduling choices. The specific prompt template and reasoning example are shown in Fig. 3.

The detailed training process of LarS

The detailed algorithm is shown in Algorithm 1 which comprises two phases, beginning with the DRL-guided

Algorithm 1 DRL-Guided GPT Trajectory Filtering and Unified LoRA Fine-Tuning

// DRL-Guided GPT Trajectory Filtering Phase
Initialize an empty list $\tau_{\text{filtered}} \leftarrow [\,]$
for each environment env **do**
  $s \leftarrow env.reset()$
  while not env.terminated **do**
    $p \leftarrow \text{GENERATEPROMPT}(s)$
    $st \leftarrow \text{GPT\_4O.GENERATETRAJECTORY}(p)$
    $a_{\text{DRL}} \leftarrow \text{AGENT.CHOOSE\_BEST\_ACTION}(s)$
    if $a_{\text{GPT}} = a_{\text{DRL}}$ **then**
      $\tau_{\text{filtered}}.\text{append}\big((p,\, st)\big)$
    end if
    $s \leftarrow env.step(a_{\text{GPT}})$
  end while
end for
// LoRA Fine-Tuning Phase
for each tuple $(p,\, st)$ in τ_{filtered} **do**
  $input\_text \leftarrow p$
  $target\_text \leftarrow st$
  $tokens_{\text{in}} \leftarrow \text{TOKENIZER}(input\_text)$
  $tokens_{\text{out}} \leftarrow \text{TOKENIZER}(target\_text)$
  $labels \leftarrow \text{copy}(tokens_{\text{out}})$
  $\text{MASKTOKENS}(tokens_{\text{in}},\, labels)$
end for
Fine-tune scheduling LLM parameters ϕ (with LoRA adapters) on the tokenized examples by minimizing: $\mathcal{L}_{\text{distill}} = \mathbb{E}_{(p,\, st)\sim\tau_{\text{filtered}}}\big[-\log P_\phi(st \mid p)\big]$.

GPT trajectory filtering phase. Initially, an empty filtered trajectory set τ_{filtered} is created. For each environment scenario, the environment state s is reset. While the environment has not terminated, the corresponding prompt p encoding the current state is generated, and GPT-4o generates the scheduling trajectory st including CoT reasoning and action a_{GPT} based on the prompt p. Subsequently, a DRL agent selects the optimal action a_{DRL} for the given state. If the GPT-selected action a_{GPT} matches the DRL agent's action a_{DRL}, the tuple comprising the prompt p and trajectory st is stored into τ_{filtered}. The environment is then updated based on GPTs selected action, advancing the state.

In the subsequent LoRA fine-tuning phase, each tuple (p, st) from the filtered trajectory set τ_{filtered} is utilized for fine-tuning. Specifically, the prompt p serves as the input text, and the corresponding full trajectory st, including the selected action, functions as the target text. Both texts are tokenized, converting the prompt and trajectory into token IDs. During this phase, labels are generated by masking prompt tokens within the tokenized labels to ensure only trajectory tokens contribute to the loss calculation. Finally, the scheduling LLM parameters, enhanced with LoRA adapters, are fine-tuned by minimizing the negative log-likelihood of the trajectory tokens conditioned upon the prompt, thus refining model behavior in alignment with DRL guidance.

In summary, LarS provides notable operational advantages over conventional scheduling methods by effectively integrating DRLs optimization precision with GPT-4o's generalization capability, enhancing adaptability across different scenarios, improving decision transparency through structured prompting, and dynamically optimizing multi-objective goals based on real-time system states, thus significantly improving cost efficiency, QoS compliance, operational scalability, and explainability.

Experiments evaluation

In this section, we compare our approach with several widely used scheduling methods and evaluate the generalization performance of LarS.

Environment setup

To ensure scheduler robustness, we designed training environments encompassing diverse resource configurations. And the job arrival process follows a Poisson distribution, reflecting the stochastic nature of job arrivals in real-world cloud environments. Figure 4 presents a scatter plot of the 10 training environments, with the x-axis and y-axis representing the counts of High CPU and High I/O VMs, respectively. The distributed point pattern demonstrates significant environmental heterogeneity, which is essential to prevent the the scheduling LLM

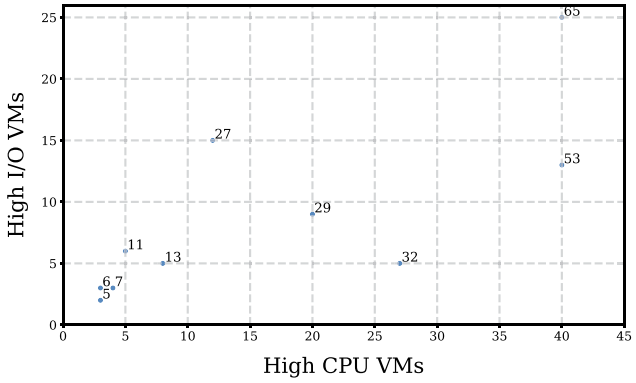

Fig. 4 Scatter plot of the 10 DRL training environments

from overfitting and ensure reliable performance across varied workload scenarios.

In our experiments, the DRL agents were trained using DQN with the following parameters: replay buffer size was set to 1600, batch size was 60, and the target network update interval was configured to occur every 150 steps. We adopted the RMSProp optimizer with a learning rate of 5×10^{-3}, a discount factor (γ) of 0.9, and employed an adaptive epsilon-greedy exploration strategy, where epsilon incrementally increased from 0.01 to 0.9 with increments of 0.003 per training step. For the LLM fine-tuning stage, we utilized the LLaMA-2-13B model, applying LoRA with $r = 8$, $\alpha = 16$, and a dropout rate of 0.05. The training was conducted for 30 epochs using the Adam optimizer with a learning rate of 3×10^{-4}, a batch size of 128, and a validation set comprising 5% of the data.

To conduct comprehensive experiments, we utilized a computing environment equipped with an NVIDIA A800 GPU featuring 80 GB of memory, 14 virtual CPUs based on Intel Xeon(R) Gold 6348 running at 2.60 GHz, and 100 GB RAM. Software-wise, our setup included Python 3.12 running on Ubuntu 22.04, PyTorch 2.3.0, and CUDA 12.1 to optimize neural network computations and ensure efficient GPU utilization during training and evaluation.

Baselines

We evaluate nine scheduling strategies on an 11-VM test environment consisting of 5 High CPU and 6 High I/O VMs. The evaluated strategies comprise:

1. **Random**: Assigns tasks arbitrarily to VMs without considering load, priority, or deadlines.
2. **Round Robin**: Cyclically assigns tasks across all VMs in fixed order, ensuring equal distribution but ignoring task complexity and urgency.
3. **Earliest**: Selects and dispatches tasks with nearest deadlines first, greedily optimizing completion time but potentially neglecting overall system load balance.

Table 3 Performance comparison of scheduling strategies on an 11-VM environment

Strategy	Avg. Resp. (s)	Success Rate (%)	Cost
Naive Methods			
Random	0. 422	31. 6	0. 596
Round-Robin	0. 290	47. 0	0. 602
Earliest	0. 282	50. 0	0. 607
DRL-based Method			
DQN	0. 191	98. 6	0. 421
LLM-based Methods			
GPT-4o	0. 252	79. 4	0. 400
LLaMA-2 (untrained)	9. 064	24. 5	0. 486
LLaMA-2 (+1000 samples)	0. 245	74. 3	0. 437
LLaMA-2 (+2000 samples)	0. 364	43. 3	0. 529
LLaMA-2 (+4700 samples)	0. 211	88. 0	0. 439

4. **GPT-4o**: Leverages advanced LLM to predict optimal scheduling decisions based on workload and task requirements.
5. **DQN**: Utilizes deep Q-learning to learn scheduling policies by exploring state-action rewards, continuously improving task allocation efficiency through training.
6. **LLaMA-2 (untrained)**: Applies pretrained LLaMA-2 model without any task-specific fine-tuning, using generic language capabilities for scheduling decisions.
7. **LLaMA-2 (+1000 samples)**: LLaMA-2 fine-tuned with 1000 samples.
8. **LLaMA-2 (+2000 samples)**: LLaMA-2 fine-tuned with 2000 samples.
9. **LLaMA-2 (+4700 samples)**: LLaMA-2 fine-tuned with 4700 samples.

The performance of these strategies is evaluated using three metrics:

Average Response Time : Lower values indicate faster task completion.

Success Rate : The proportion of tasks meeting the desired QoS requirements.

Cost : Reflects the average resource expenditure.

Results

As demonstrated in Table 3 and Fig. 5, the LLaMA-2 model fine-tuned with 4700 samples achieves strong performance across all three evaluation metrics. The Scheduling results of GPT-4o indicating that powerful LLMs can deliver competitive performance even without fine-tuning. In contrast, all naive methods exhibit substantially inferior performance, characterized by prolonged response times and reduced success rates. While DQN-based methods show significant improvements over naive methods and even surpass some LLM-based schedulers, their inherent limitations hinder effective

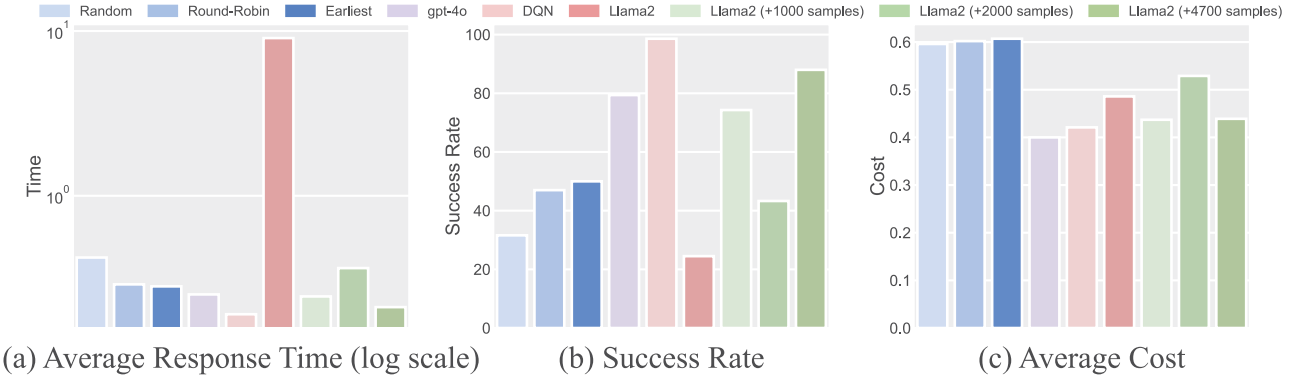

Fig. 5 Performance comparison of 9 scheduling strategies in the 11-VM environment in terms of: (**a**) average response time, (**b**) success rate, and (**c**) cost

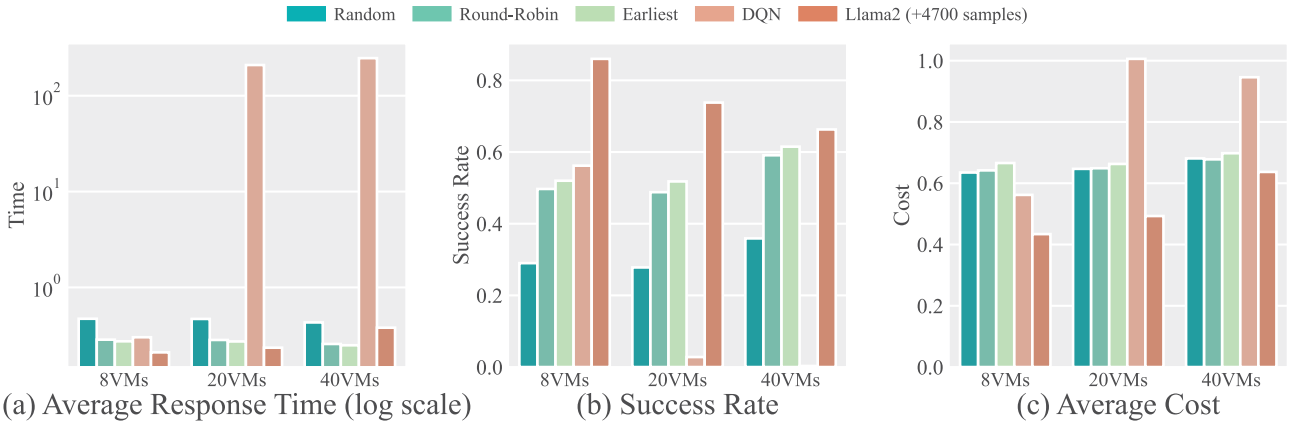

Fig. 6 Comparison of the generalization performance of the fine-tuned LLaMA-2 scheduler (trained on 4700 samples) and baseline methods (random, round-robin, earliest, and DQN) across three test environments with 8, 20, and 40 VMs: (**a**) average response time (log scale), (**b**) task success rate, and (**c**) average execution cost

adaptation to dynamic cloud environments. These results demonstrate that LLaMA-2, after appropriate fine-tuning, significantly improves scheduling success rate and efficiency, particularly as the sample size increases.

Evaluation of generalization performance

To comprehensively evaluate the generalization capability of our proposed scheduler, we deployed the fine-tuned LLaMA-2 model (+4700 samples) across three distinct test environments with varying scales (8 VMs, 20 VMs and 40 VMs). We compared its performance with four baseline scheduling methods: Random, Round-Robin, Earliest, and DQN. Performance was quantified through three metrics: average response time, task success rate, and average execution cost. It should be noted that for the DQN model, we trained it in an 11-VM environment and then applied it directly to the three environments without retraining.

As shown in Fig. 6(a), the scheduling LLM shows excellent performance in terms of average response time across environments. However, the DQN scheduler exhibits extremely poor performance in medium and large environments (20 and 40 VMs), which can be attributed to overfitting of its training strategy. The DQN scheduler develops a specialized strategy for its specific training scenario, resulting in significant response delays (over 200 seconds) when encountering larger-scale scenarios. In stark contrast, our scheduling LLM maintained consistently low response times, demonstrating robust generalization due to effective semantic knowledge transfer from the pre-trained language model.

The results in Fig. 6(b) further demonstrate the generalization strengths of the scheduling LLM. Our proposed method achieved the highest success rate **in three environments**, clearly surpassing all baseline methods. Although the success rate drops in the largest test environment (40 VMs), the scheduling LLM still maintains slightly better performance than the traditional baseline method. This reduction in success rate at the largest scale indicates potential challenges in scalability and adaptation to substantially more complex resource allocation contexts. However, even in this challenging scenario, our scheduler did not exhibit catastrophic performance degradation as observed with DQN, whose success rate

dramatically plummeted due to its inability to generalize beyond the narrow training regime.

Finally, in terms of average cost (Fig. 6(c)), our scheduler consistently delivered the lowest resource utilization costs in three environments, highlighting its capability to effectively optimize the cost. The scheduling LLM achieved a cost nearly 20% lower than the closest competing naive method in the 8 VMs environment and maintained competitive efficiency in the medium-scale environment. At the 20 and 40 VMs scale, although the cost of the scheduling model is narrowing compared to other methods, it still outperforms them.

In summary, these results affirm that the LLaMA-2 model, fine-tuned with a dataset of 4700 samples, possesses significant generalization advantages. It demonstrates exceptional efficiency and adaptability in low-scale and medium-scale environments, balancing low response times, high success rates, and cost efficiency. While facing scalability challenges at larger scales, the scheduling LLM continues to outperform naive methods and DQN approach, underscoring the robustness and practical utility of integrating LLM-based semantic reasoning into cloud scheduling strategies.

Conclusion

In this paper, we propose LarS, a DRL-enhanced scheduling LLM tailored for cloud task scheduling. LarS integrates the decision-making capability of deep reinforcement learning with the generalization and reasoning capabilities of large language models. By combining these complementary strengths, LarS enhances scheduling performance while overcoming the limited generalization typically observed in standalone DRL approaches. Experimental results show that LarS outperforms both naive baselines and standalone DRL methods across diverse cloud environments. Future work will focus on extending LarS to cloud–edge environments and improving its adaptability under few-shot learning conditions.

Authors' contributions

H.P.: Software, Validation, Writing original draft, G.Y.: Writing original draft, Writing-review & editing, S.Y.: Formal analysis, Validation, Q.W.: Conceptualization, Validation, C.L.: Supervision, X.C.: Software, Validation, L.C.: Conceptualization, Methodology, Writing-review & editing, Validation, Supervision.

Funding

This research was supported by the Fundamental Research Funds for the Central Universities (2025JC002).

Data availability

No datasets were generated or analysed during the current study.

Declartions

Competing interests

The authors declare no competing interests.

Received: 17 June 2025 / Accepted: 23 November 2025
Published online: 18 December 2025

References

1. Cheng L, He H, Gu Y, Liu Q, Zhao Z, Fang F (2024) Mars: multi-agent deep reinforcement learning for real-time workflow scheduling in hybrid clouds with privacy protection. In 2024 IEEE 30th International Conference on Parallel and Distributed Systems, pp 657–666
2. Devi N, Dalal S, Solanki K, Dalal S, Lilhore UK, Simaiya S, Nuristani N (2024) A systematic literature review for load balancing and task scheduling techniques in cloud computing. Artif Intel Rev 57(10):276
3. Gu Y, Liu Z, Dai S, Liu C, Wang Y, Wang S, Theodoropoulos G, Cheng L (2025) Deep reinforcement learning for job scheduling and resource management in cloud computing: an algorithm-level review. arXiv preprint arXiv:2501.01007
4. Zhou G, Tian W, Buyya R, Xue R, Song L (2024) Deep reinforcement learning-based methods for resource scheduling in cloud computing: a review and future directions. Artif Intel Rev 57(5):124
5. Gu Y, Cheng F, Yang L, Xu J, Chen X, Cheng L (2024) Cost-aware cloud workflow scheduling using drl and simulated annealing. Digit Commun Networks 10(6):1590–1599
6. Fan W, Zhao L, Liu X, Su Y, Li S, Wu F, Liu Y (2022) Collaborative service placement, task scheduling, and resource allocation for task offloading with edge-cloud cooperation. IEEE Trans Mob Comput 23(1):238–256
7. Ali A, Shah SAA, Al Shloul T, Assam M, Ghadi YY, Lim S, Zia A (2024) Multi-objective harris hawks optimization-based task scheduling in cloud-fog computing. IEEE Internet Things J 11(13):24334–24352
8. Lu J, Yang J, Li S, Li Y, Jiang W, Dai J, Hu J (2024) A2C-DRL: dynamic scheduling for stochastic edge-cloud environments using A2C and deep reinforcement learning. IEEE Internet Things J
9. Mangalampalli S, Karri GR, Ratnamani M, Mohanty SN, Jabr BA, Ali YA, Ali S, Abdullaeva BS (2024) Efficient deep reinforcement learning based task scheduler in multi cloud environment. Sci Rep 14(1):21850
10. Zhang Z, Zhang F, Xiong Z, Zhang K, Chen D (2024) LSIA3CS: deep reinforcement learning-based cloud-edge collaborative task scheduling in large-scale IIoT. In IEEE Internet of Things Journal
11. Xing Y (2024) Work scheduling in cloud network based on deep Q-LSTM models for efficient resource utilization. J Grid Comput 22(1):36
12. Raza M, Jahangir Z, Riaz MB, Saeed MJ, Sattar MA (2025) Industrial applications of large language models. Sci Rep 15(1):13755
13. Vasileiou SL, Yeoh W (2025) TRACE-CS: a synergistic approach to explainable course scheduling using LLMs and logic. Proc AAAI Conf Artif Intell, vol 39. pp 29706–29708
14. Krishnamurthy B, Shiva SG (2025) Large language model-guided SARSA algorithm for dynamic task scheduling in cloud computing. Mathematics 13(6):926
15. Tambwekar P, A (2024) Towards explainable task scheduling using large language models. Comput Operations Res 160:106323. https://doi.org/10.1016/j.cor.2023.106323
16. Ismayilov G, Topcuoglu HR (2020) Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in cloud computing. Future Gener Comput Syst 102:307–322
17. Shukri SE, Al-Sayyed R, Hudaib A, Mirjalili S (2021) Enhanced multi-verse optimizer for task scheduling in cloud computing environments. Expert Syst With Appl 168:114230
18. Nabi S, Ahmad M, Ibrahim M, Hamam H (2022) Adpso: adaptive pso-based task scheduling approach for cloud computing. Sensors 22(3):920
19. Mangalampalli S, Karri GR, Kose U (2023) Multi objective trust aware task scheduling algorithm in cloud computing using whale optimization. J King Saud Univ-Comput Inf Sci 35(2):791–809
20. Tong Z, Chen H, Deng X, Li K, Li K (2020) A scheduling scheme in the cloud computing environment using deep Q-learning. Inf Sci 512:1170–1191
21. Siddesha K, Jayaramaiah G, Singh C (2022) A novel deep reinforcement learning scheme for task scheduling in cloud computing. Cluster Comput 25(6):4171–4188
22. Islam MT, Karunasekera S, Buyya R (2021) Performance and cost-efficient spark job scheduling based on deep reinforcement learning in cloud computing environments. IEEE Trans Parallel And Distrib Syst 33(7):1695–1710
23. Xiu X, Li J, Long Y, Wu W (2023) Mrlcc: an adaptive cloud task scheduling method based on meta reinforcement learning. J Cloud Comput 12(1):75

24. Tang X, Liu F, Xu D, Jiang J, Tang Q, Wang B, Wu Q, Chen CP (2025) Llm-assisted reinforcement learning: leveraging lightweight large language model capabilities for efficient task scheduling in multi-cloud environment. In IEEE Transactions on Consumer Electronics
25. Abgaryan H, Harutyunyan A, Cazenave T (2024) Llms can schedule. arXiv preprint arXiv:2408.06993
26. Pallagani V, Muppasani BC, Roy K, Fabiano F, Loreggia A, Murugesan K, Srivastava B, Rossi F, Horesh L, Sheth A (2024) On the prospects of incorporating large language models (LLMs) in automated planning and scheduling (APS). In Proceedings of the International Conference on Automated Planning and Scheduling, vol 34. pp 432–444
27. Cheng F, Huang Y, Tanpure B, Sawalani P, Cheng L, Liu C (2022) Cost-aware job scheduling for cloud instances using deep reinforcement learning. Cluster Comput 1–13
28. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G et al. (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533
29. Li T, Ying S, Zhao Y, Shang J (2023) Batch jobs load balancing scheduling in cloud computing using distributional reinforcement learning. IEEE Trans On Parallel And Distrib Syst 35(1):169–185
30. He H, Gu Y, Hu Y, Fang F, Ning X, Chen X, Cheng L (2025) Real-time workflow scheduling in hybrid clouds with privacy and security constraints: a deep reinforcement learning approach. Expert Syst Appl 278:127376
31. Schaul T, Quan J, Antonoglou I, Silver D (2015) Prioritized experience replay. arXiv preprint arXiv:1511.05952
32. Hu EJ, Shen Y, Wallis P, Allen-Zhu Z, Li Y, Wang S, Wang L, Chen W et al. (2022) Lora: low-rank adaptation of large language models. ICLR 1(2):3

Publisher's Note