Deadline-Constrained Dynamic Workflow Scheduling Can be Cast as a Representation Learning Problem

Ya Shen*
Victoria University of Wellington
ya.shen@ecs.vuw.ac.nz

Gang Chen
Victoria University of Wellington
aaron.chen@ecs.vuw.ac.nz

Hui Ma
Victoria University of Wellington
hui.ma@ecs.vuw.ac.nz

Mengjie Zhang
Victoria University of Wellington
mengjie.zhang@ecs.vuw.ac.nz

Abstract

Deadline-Constrained **C**ost-**A**ware **D**ynamic **W**orkflow **S**cheduling (D-CADWS) aims to minimize virtual machine (VM) rental cost while maintaining high workflow deadline satisfaction in dynamic cloud environments. The problem is challenging because the deadline impact of assigning a workflow task to a VM is not directly observable at decision time. It depends on long-horizon effects, including downstream task dependencies, VM queueing, and interactions among dynamically arriving workflows. Existing methods usually fold deadline violations into a single reward or penalty term, which obscures the distinction between action safety and action cost. In this paper, we cast D-CADWS as a representation learning problem. The core idea is to learn a deadline-aware representation that can predict, before execution, whether a candidate action is risky with respect to the current task deadline, and then use it to guide cost-efficient scheduling. Based on this idea, we propose a representation-centered deep reinforcement learning (RCDRL) method. RCDRL constructs task-level deadline supervision, trains a predictive deadline model to learn deadline-aware representations, and learns a VM-cost critic on top of this representation. The final policy follows a safe-first rule that rejects risky actions before minimizing VM cost. We further propose a two-phase training strategy to keep the learned representation aligned with the evolving policy-induced data distribution. Experiments on dynamic workflow scheduling benchmarks show that RCDRL can reduce VM rental cost while maintaining strong deadline success.

1 Introduction

Cloud computing offers elastic computational resources for large-scale applications [Buyya et al., 2011a]. Many of these applications are naturally modeled as workflows with precedence constraints, typically represented as directed acyclic graphs (DAGs). In real-world cloud environments, workflows arrive dynamically, vary in structure and scale, and must be scheduled onto heterogeneous virtual machines (VMs) under pay-as-you-go pricing [Ibrahim et al., 2011]. These challenges make Cost-Aware Dynamic Workflow Scheduling (CADWS) a central problem in cloud resource management.

Deep reinforcement learning (DRL) has become a promising paradigm for CADWS because it can learn effective scheduling policies automatically from interaction with dynamic cloud environments [Shen et al., 2026a]. Handling workflow deadlines is critical in CADWS. However, most existing DRL-based methods focus on unconstrained or softly constrained formulations. In these

*Corresponding author

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

approaches, deadline violations are folded into a single scalar objective by combining deadline-related penalty terms with VM cost. Such a formulation is acceptable when deadline violations are treated as a cost that can be traded against VM rental cost. However, it is misaligned with the real-world setting, where workflow deadline misses should be explicitly controlled while VM rental costs are minimized [Buyya et al., 2009, Zhang et al., 2014].

In this paper, we study *deadline-constrained CADWS* (D-CADWS). The objective is to minimize VM rental cost while maintaining a high workflow deadline success rate. D-CADWS differs from standard CADWS because reducing VM cost is not sufficient: a low-cost policy that misses many workflow deadlines is not acceptable. Therefore, the policy should first avoid scheduling actions that are likely to harm deadline satisfaction, and then minimize VM cost.

However, the **key challenge** is that the deadline consequence of a scheduling action is not directly observable at decision time [Shen et al., 2025]. When the policy assigns a task to a candidate VM, the effect of this decision may only become clear much later through downstream task dependencies, VM queueing, and final workflow completion [Buyya et al., 2011a]. From this perspective, *D-CADWS can be cast as a representation learning problem*: the policy needs a good deadline-aware representation that makes the future deadline implication of each candidate action predictable before execution.

Under this view, decision-making in D-CADWS requires the policy to evaluate each candidate action by answering two key questions: *(i)* whether assigning the current workflow task to a given VM is risky with respect to its task deadline, and *(ii)* what future VM cost is likely to be incurred by that action. When both can be reliably determined from a learned deadline-aware representation, scheduling becomes both simpler and more aligned with the objective: the policy first filters out unsafe actions, and then selects the minimum-cost option from the remaining safe set.

Inspired by this idea, we propose a Representation-Centered DRL (RCDRL) method for D-CADWS. **First**, RCDRL constructs task-level deadline supervision to turn delayed workflow-level deadline feedback into dense action-level learning signals. **Second**, it trains a predictive deadline model, whose encoder learns a deadline-aware representation and whose output estimates the deadline risk of each candidate VM action. **Third**, a VM-cost critic is learned on top of this representation to estimate future VM rental cost. **After that**, the final scheduling policy can be derived from a safe-first rule: it rejects actions with high predicted deadline risk before selecting the lowest-cost action among the remaining candidates.

Finally, to ensure stable and effective learning, we propose a *two-phase alternating training strategy*. **In the first phase**, the predictive deadline model is updated by supervised learning to keep the representation focused on deadline-relevant state-action features. **In the second phase**, a VM-cost critic is trained by temporal-difference (TD) learning [Sutton, 1988] on top of the learned representation. This design separates supervised deadline-aware representation learning from critic learning while keeping the representation aligned with the continually evolving policy-induced data distribution. The resulting RCDRL fundamentally differs from existing DRL methods for D-CADWS by treating constraint handling as deadline-aware representation learning rather than as reward scalarization. Our main contributions are summarized as follows:

- **D-CADWS as a representation learning problem.** We present a new view of D-CADWS: its core challenge is to learn a *deadline-aware representation* that makes the deadline risk of each candidate action predictable before execution. Scheduling decisions are then guided by this learned representation. To the best of our knowledge, this is the first DRL method to cast D-CADWS as a representation learning problem rather than reward scalarization.
- **Task-level deadline supervision for representation learning.** We design a task-level deadline target that transforms delayed workflow-level deadline feedback into dense action-level supervision. This supervision trains a predictive deadline model to learn deadline-aware representations and estimate the deadline risk of each candidate VM action.
- **Safe-first policy with separate deadline and cost estimation.** On top of the learned deadline-aware representation, we train a VM-cost critic to estimate future VM rental cost. The final policy first filters out actions with high predicted deadline risk and then selects the lowest-cost action from the remaining safe set, matching the scheduling preference of D-CADWS.
- **Two-phase alternating training strategy.** We introduce a two-phase alternating training strategy that separates supervised deadline-aware representation learning from VM-cost

critic learning. This design keeps the representation aligned with the data distribution induced by the continually evolving policy while preserving its focus on deadline-relevant features.

2 Related Work

DRL for dynamic workflow scheduling. Deep reinforcement learning has been widely used for learning effective scheduling policies for CADWS [Dong et al., 2021, Gu et al., 2024, Pan and Wei, 2024, Shen et al., 2024, 2025, 2026a, Yang et al., 2025], Vehicle Routing [Meng et al., 2025, Pan et al., 2025, Goh et al., 2025], Job Shop Scheduling [Zhang et al., 2020, Wang et al., 2026, Xiao et al., 2026], and resource allocation in cloud computing [Fang et al., 2026a,b]. Existing DRL-based CADWS methods mainly improve scheduling performance through better state embedding learning, expressive policy architectures, or more effective training strategies [Shen et al., 2024, 2025, 2026a,b, Yang et al., 2025]. However, these methods are mostly developed for unconstrained or softly constrained CADWS, where VM cost and deadline-related penalty are combined into a single scalar objective. Under such formulations, the policy is trained to optimize a merged reward, rather than to explicitly distinguish whether a candidate action is deadline-safe and how costly it will be if executed.

Constraint handling in workflow scheduling. Another line of work designs heuristic algorithms for deadline-constrained workflow scheduling. OHDS combines task prioritization, sub-deadline allocation, and VM migration to improve resource efficiency [Fan et al., 2022]. SMWDSA uses sub-deadline assignment and scheduling feedback to reduce cost [Ye et al., 2022], while ET2FA improves scheduling efficiency through task-aware scheduling and VM management [Sun et al., 2022]. These methods handle deadline constraints explicitly, but their constraint handling is mainly based on manually designed rules, which may limit their adaptability to changing deadline requirements and dynamic scheduling conditions.

Our work is motivated by a different perspective: D-CADWS can be cast as a deadline-aware representation learning problem. Existing heuristic methods explicitly use deadline constraints but rely on handcrafted scheduling rules. Existing DRL methods learn adaptive policies but usually handle deadlines implicitly through scalarized rewards. RCDRL differs from both groups by learning a deadline-aware representation that makes the deadline risk of each candidate VM action predictable before execution. It then learns a VM-cost critic on top of this representation and derives a safe-first policy that filters high-risk actions before minimizing VM cost. This design directly matches the constrained objective of D-CADWS: control deadline misses first, and reduce VM rental cost within the safer action region. To the best of our knowledge, this is the first DRL method to explicitly formulate D-CADWS from a representation-learning perspective.

3 Preliminaries of Deadline-Constrained CADWS (D-CADWS)

Problem setting. A D-CADWS instance consists of a set of workflows $\mathcal{W} = \{W_1, \ldots, W_N\}$ that arrive over time and must be executed on virtual machines (VMs) leased from the cloud. Each workflow W_i is represented as a directed acyclic graph (DAG), $W_i = (\mathcal{O}_i, \mathcal{E}_i)$, where $\mathcal{O}_i$ is the set of tasks and $\mathcal{E}_i$ is the set of precedence edges. Each workflow is associated with an arrival time a_i and a workflow deadline d_i. In DRL settings, we formulate the D-CADWS problem as a Markov Decision Process (MDP). At each decision step, the scheduler considers the current ready task, namely a task whose predecessors have all completed, and assigns it to a feasible VM.

State. At decision step t, the scheduler observes a state $s_t \in \mathcal{S}$, where s_t summarizes the current scheduling context, including active workflows, ready tasks, DAG dependency information, deadline-related features, and the execution status of all available VMs.

Action. Let $\mathcal{M}_t = \{m_1, \ldots, m_{|\mathcal{M}_t|}\}$ denote the set of feasible VMs available at step t, which may include both previously leased active VMs and newly provisioned VMs. Let O_t^* denote the current ready task. An action $a_t \in \mathcal{A}(s_t)$ assigns O_t^* to one feasible VM in $\mathcal{M}_t$, where $\mathcal{A}(s_t) = \mathcal{M}_t$.

Transition. After executing a_t, the environment transitions to the next state $s_{t+1} \sim P(\cdot \mid s_t, a_t)$ and incurs an immediate VM rental cost $c_t^m \geq 0$. This immediate cost captures the VM cost directly induced by the current scheduling decision.

Trajectory. An episode terminates when all workflows in the current CADWS instance have been fully scheduled. The resulting trajectory is denoted by $\tau = (s_0, a_0, c_0^m, s_1, a_1, c_1^m, \ldots, s_T)$.

Workflow-level deadline satisfaction. Requiring every workflow in a long dynamic trajectory to meet its deadline is overly strict and often unrealistic, especially when workflows have different deadline tightness levels. We therefore formulate the deadline requirement through workflow-level miss and success rates. For each workflow W_i, let its deadline-miss indicator under trajectory τ be:

$$B_i(\tau) = \begin{cases} 1, & \text{if } CT(W_i) > d_i, \\ 0, & \text{otherwise}, \end{cases} \tag{1}$$

where $CT(W_i)$ denotes the completion time of workflow W_i. The workflow deadline miss rate $M(\tau)$ and success rate $S(\tau)$ of trajectory τ are then:

$$M(\tau) = \frac{1}{N(\tau)} \sum_{i=1}^{N(\tau)} B_i(\tau), \qquad S(\tau) = 1 - M(\tau), \tag{2}$$

where $N(\tau)$ is the number of workflows completed in trajectory τ.

Objective. Let the total VM rental cost of trajectory τ be $C^m(\tau) = \sum_{t=0}^{T-1} c_t^m$. The goal of D-CADWS is to minimize expected VM rental cost while keeping the expected workflow deadline miss rate below an acceptable tolerance ϵ_{miss}:

$$\min_{\pi} \ \mathbb{E}_{\tau\sim\pi}[C^m(\tau)] \qquad \text{s.t.} \quad \mathbb{E}_{\tau\sim\pi}[M(\tau)] \le \epsilon_{\text{miss}}. \tag{3}$$

Appendix A gives the full formulation, and Appendix C gives the details of the miss-rate constraint.

4 Methodology

We now present the proposed RCDRL for D-CADWS. RCDRL is motivated by an interesting insight: D-CADWS can be cast as a representation learning problem. Before executing a scheduling action, the policy should know: **(i)** whether the action is likely to harm deadline feasibility, and **(ii)** how much future VM cost it is likely to incur. These two quantities cannot be directly read from the raw scheduling state. RCDRL therefore learns a deadline-aware representation for each state-action pair (s, a), so that deadline risk and future VM cost can be estimated before execution. Figure 1 gives an overview of the decision and training procedure of RCDRL.

RCDRL consists of four key stages. **Stage 1**: It constructs task-level deadline supervision from workflow-level deadline requirements, turning delayed deadline feedback into dense action-level learning signals. **Stage 2**: It trains a predictive deadline model, which contains an action-conditioned encoder for producing deadline-aware representations and a risk head for estimating task-deadline risk. **Stage 3**: It trains a VM-cost critic on top of the learned representation by temporal-difference (TD) learning. Finally, in **Stage 4**, it derives a safe-first scheduling policy that first filters out actions with high predicted deadline risk and then selects the lowest-cost action among the remaining candidates.

4.1 Task-Level Deadline Supervision (Stage 1)

Workflow-level deadline success is observed only after a workflow completes, making it too delayed and sparse to directly supervise each scheduling decision. To obtain dense action-level supervision, RCDRL uses task deadlines decomposed from the workflow deadline, with the upward-rank construction detailed in Appendix A.

For a collected transition (s_t, a_t), let O_t^* be the current ready task, $\widehat{FT}(O_t^*, a_t)$ be its estimated finish time under action a_t, and d_t^{task} be its assigned task deadline. We define the normalized task-deadline lateness as:

$$\ell_t^{\text{task}} = \frac{\max\{0, \widehat{FT}(O_t^*, a_t) - d_t^{\text{task}}\}}{\max\{d_t^{\text{task}} - T_t, \epsilon_{\text{den}}\}}, \tag{4}$$

where T_t denotes the current scheduling time at decision step t, and $\epsilon_{\text{den}} > 0$ avoids division by zero. The supervision target is its soft-clipped form:

$$y_t^{\text{task}} = \frac{\ell_t^{\text{task}}}{1 + \ell_t^{\text{task}}}. \tag{5}$$

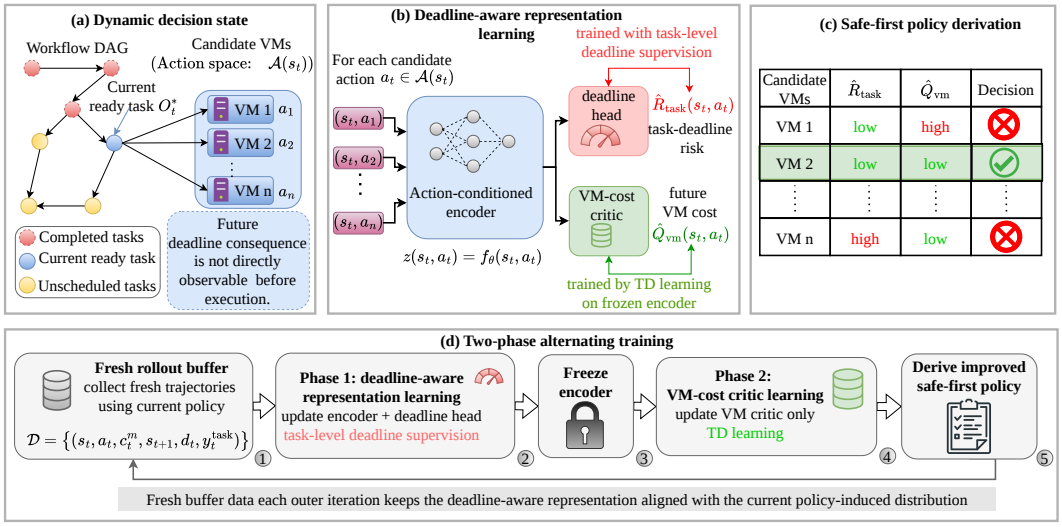

Figure 1: **Overview of RCDRL**. (a) At each decision step, the scheduler assigns the current ready task O_t^* to one feasible VM, while the deadline consequence of each candidate action is unknown before execution. (b) RCDRL learns a deadline-aware representation $z(s_t, a) = f_\theta(s_t, a)$, from which a predictive deadline model estimates task-deadline risk $\widehat{R}_{\text{task}}(s_t, a_t)$ and a VM-cost critic estimates future VM cost $\widehat{Q}_{\text{vm}}(s_t, a_t)$. (c) The safe-first policy filters high-risk actions and selects the lowest-cost action among the remaining safer candidates; if the safe set is empty, it falls back to the top-ρ lowest-risk actions. (d) Training alternates between task-level deadline supervision and VM-cost critic learning on a frozen encoder, using fresh rollout data at each outer iteration.

The target $y_t^{\text{task}} \in [0, 1)$ is zero when the action is expected to finish the task within its task deadline, and increases smoothly as the estimated lateness becomes larger. It therefore provides a bounded, action-level deadline-risk signal for learning the deadline-aware representation.

4.2 Predictive Deadline Model for Deadline-Aware Representation Learning (Stage 2)

The goal of Stage 2 is to learn a *deadline-aware representation*. This representation should make the deadline implication of each candidate scheduling action predictable before execution. To achieve this, RCDRL trains a predictive deadline model with the task-level supervision defined in Eq. (5). The model consists of an *action-conditioned encoder* and a *task-deadline risk head*. The encoder produces the representation, while the risk head forces this representation to capture deadline-relevant features by predicting whether assigning the current ready task to a candidate VM is likely to meet its task deadline.

Action-conditioned encoder. The encoder maps each candidate state-action pair to a latent representation:

$$z(s_t, a_t) = f_\theta(s_t, a_t) \in \mathbb{R}^d, \tag{6}$$

where f_θ is the encoder network with parameters θ, and $z(s_t, a_t)$ is the latent representation of the candidate decision (s_t, a_t). We encode state-action pairs rather than states alone because, in D-CADWS, the quality of a scheduling decision depends not only on the current system state, but also on the VM chosen for the current ready task.

Task-deadline risk head. To make the learned representation deadline-aware, the deadline-risk head g_ϕ^{dea} maps $z(s_t, a_t)$ to a task-deadline risk score:

$$\hat{r}_t^{\text{task}} = \sigma\big(g_\phi^{\text{dea}}\big(z(s_t, a_t)\big)\big)\,, \tag{7}$$

where $\sigma(\cdot)$ is the sigmoid function. The predicted score $\hat{r}_t^{\text{task}} \in (0, 1)$ estimates the task-deadline risk of taking action a_t at state s_t. A smaller value indicates that the candidate VM assignment is predicted to keep the current task within its deadline. The complete predictive deadline model is therefore:

$$\hat{R}_{\text{task}}(s_t, a_t) = \hat{r}_t^{\text{task}} = \sigma\big(g_\phi^{\text{dea}}(f_\theta(s_t, a_t))\big)\,. \tag{8}$$

Deadline-model objective. The encoder f_θ and the deadline-risk head g_ϕ^{dea} are trained jointly by minimizing a binary cross-entropy loss between the predicted task-deadline risk $\hat{r}_t^{\mathrm{task}}$ and the supervision target y_t^{task} in Eq. (5):

$$\mathcal{L}_{\mathrm{dea}} = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[-y_t^{\mathrm{task}}\log\hat{r}_t^{\mathrm{task}} - (1-y_t^{\mathrm{task}})\log(1-\hat{r}_t^{\mathrm{task}})\right]. \tag{9}$$

This objective gives the encoder a clear representation-learning role. Rather than learning generic scheduling features, f_θ is optimized to produce representations from which task-deadline risk can be accurately predicted. Therefore, the latent vector $z(s_t, a_t)$ becomes a *deadline-aware representation*, which is later reused for VM-cost critic learning and safe-first policy derivation.

4.3 VM-Cost Critic Learning (Stage 3)

After the predictive deadline model is updated, we freeze its encoder f_θ so that VM-cost critic learning does not perturb the deadline-aware representation learned in f_θ. The VM-cost critic is then learned on top of this fixed f_θ.

VM-cost critic. The VM-cost critic estimates the future VM rental cost-to-go induced by taking action a_t at state s_t and then following the current safe-first policy:

$$Q_{\mathrm{vm}}^{\pi}(s_t,a_t) = \mathbb{E}_\pi\left[\sum_{k=t}^{T-1}\gamma_c^{k-t}c_k^m \,\middle|\, s_t,a_t\right], \tag{10}$$

where $\gamma_c \in (0,1]$ is the discount factor. We parameterize the VM-cost critic as:

$$\hat{Q}_{\mathrm{vm}}(s_t,a_t) = \mathrm{softplus}\big(q_\beta^{\mathrm{vm}}(f_\theta(s_t,a_t))\big), \tag{11}$$

where q_β^{vm} is the VM-cost critic head. During Stage 3, the encoder f_θ is frozen, and only the VM-cost critic head q_β^{vm} is updated. The VM-cost critic is learned by TD learning. Let $\bar{Q}_{\mathrm{vm}}$ denote the target VM-cost critic. For a sampled transition $(s_t, a_t, c_t^m, s_{t+1}, d_t)$, the TD target is:

$$y_t^{\mathrm{vm,TD}} = c_t^m + \gamma_c(1-d_t)\bar{Q}_{\mathrm{vm}}(s_{t+1},a_{t+1}^*), \tag{12}$$

where $d_t = 1$ indicates a terminal transition, and a_{t+1}^* is selected by the current safe-first policy defined in Section 4.4. The VM-cost critic is trained by minimizing:

$$\mathcal{L}_{\mathrm{vm}} = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[\mathrm{Huber}\left(\hat{Q}_{\mathrm{vm}}(s_t,a_t) - y_t^{\mathrm{vm,TD}}\right)\right]. \tag{13}$$

Thus, the deadline model is used to decide which actions are deadline-safe, and the VM-cost critic is used to select the lowest-cost action among them. This directly follows the constrained objective of D-CADWS: satisfy deadlines first, then reduce VM cost, instead of mixing them into a scalarized reward.

4.4 Safe-First Policy Derivation (Stage 4)

Once the deadline model and VM critic are available, the scheduling policy can be derived explicitly. We do not use a separate actor network, because all feasible actions in $\mathcal{A}(s)$ can be scored directly. Given a risk threshold $\delta \in [0,1]$, RCDRL first forms the deadline-safe action set:

$$\mathcal{A}_{\mathrm{safe}}(s) = \{a \in \mathcal{A}(s) \mid \hat{R}_{\mathrm{task}}(s,a) \le \delta\}. \tag{14}$$

This set keeps only candidate actions whose predicted task-deadline risk is sufficiently small. If the safe set is nonempty, RCDRL selects the safe action with the lowest predicted VM cost:

$$\pi(s) = \arg\min_{a\in\mathcal{A}_{\mathrm{safe}}(s)} \hat{Q}_{\mathrm{vm}}(s,a). \tag{15}$$

This decision rule matches the constrained nature of D-CADWS: reduce deadline risk first, and optimize VM cost only within the safe action region.

If $\mathcal{A}_{\mathrm{safe}}(s)$ is empty, the scheduler must still select a feasible VM. RCDRL therefore uses a *top-ratio-then-cost* fallback rule: it first keeps the relatively safest ρ fraction of feasible actions:

$$\mathcal{A}_\rho(s) = \mathrm{TopRisk}_\rho\big(\mathcal{A}(s), \hat{R}_{\mathrm{task}}\big), \tag{16}$$

where $\mathrm{TopRisk}_\rho$ returns the actions with the lowest predicted task-deadline risks. RCDRL then selects the lowest-cost action from this relaxed safe set:

$$\pi_\rho(s) = \arg \min_{a \in \mathcal{A}_\rho(s)} \hat{Q}_{\mathrm{vm}}(s, a). \tag{17}$$

This fallback still prioritizes deadline safety, but avoids choosing purely by the smallest risk score when all actions slightly exceed the threshold. Considering the workflow characteristics and computing resource configurations of our experimental environment, we set $\delta = \rho = 0.2$ across all scales.

4.5 Two-Phase Alternating Training Procedure

RCDRL is trained by alternating between fresh rollout collection, deadline model learning, and VM-cost critic learning. At each outer iteration, a rollout buffer $\mathcal{D}$ is cleared and refilled with trajectories collected by the current behavior policy:

$$\mathcal{D} = \{(s_t, a_t, c_t^m, s_{t+1}, d_t, y_t^{\mathrm{task}})\}, \tag{18}$$

where $d_t \in \{0, 1\}$ is the terminal indicator. This fresh-buffer design keeps the training data aligned with the current policy-induced state-action distribution.

Phase 1: deadline-model learning. For each collected transition data, we compute the task-level target y_t^{task} in Eq. (5). We then update the encoder f_θ and the task-deadline risk head g_ϕ^{dea} by minimizing Eq. (9). This phase learns a deadline-aware representation for predicting deadline risk.

Phase 2: VM-cost critic learning. After Phase 1, we freeze the encoder to get a pre-trained deadline-aware representation. Keeping this representation fixed, we update only the VM-cost critic with the TD loss in Eq. (13). This phase improves long-horizon VM-cost estimation without disturbing the deadline-aware representation learned in Phase 1.

After the two phases, we derive an improved safe-first policy using Eqs. (14)–(17). The behavior policy for the next outer iteration is formed by ϵ-greedy exploration around this policy:

$$\mu_{k+1}(a \mid s) = \begin{cases} 1 - \epsilon, & \text{if } a = \pi(s), \\ \text{uniform over explored feasible alternatives}, & \text{otherwise}, \end{cases} \tag{19}$$

where $\epsilon \in [0, 1]$ is the exploration rate. Algorithm 1 in Appendix B summarizes the training process.

5 Experiments

This section evaluates whether RCDRL can save VM rental cost while keeping workflow deadline success close to an expensive safety reference, and which design choices produce this constrained trade-off. The code for RCDRL is available at: `https://github.com/yashenCS/RCDRL`

5.1 Experimental Setup

Environment and protocol. We evaluate all methods on the dynamic cloud workflow scheduling simulator used in recent CADWS studies [Shen et al., 2025, 2026a,b]. Each episode contains dynamically arriving workflows represented as DAGs, and the scheduler assigns each current ready task to a feasible VM while respecting dependencies and VM availability. We test three workflow-size settings, denoted by **S**, **M**, and **L**. Deadline tightness is assigned at the workflow level, so workflows with different deadline pressures can appear in the same episode. Learning-based methods use the same training instances and are evaluated on the same test set. Detailed VM configurations and workflow patterns are provided in Appendix C.

Compared methods. We compare RCDRL with three groups of methods. FASTEST-NEW-VM provisions a new fastest VM for every scheduled task and is used only as a safety-oriented upper reference for deadline success; its assumptions are given in Appendix C. We also compare with deadline-aware heuristic methods, including OHDS [Fan et al., 2022], SMWDSA [Ye et al., 2022], and ET2FA [Sun et al., 2022], and with recent DRL baselines for CADWS, including SPN-CWS [Shen et al., 2024], GATES [Shen et al., 2025], and DEFT [Shen et al., 2026a]. Unlike these DRL baselines, RCDRL

explicitly learns a deadline-aware representation to filter high-risk actions before minimizing VM cost.

Metrics. We report workflow deadline success rate and VM rental cost. We also report Deadline Gap and Cost Saving relative to FASTEST-NEW-VM. Since cost saving is useful only when deadline loss is acceptable, we further report deadline-constrained cost saving (DCCS), which counts Cost Saving only when Deadline Gap is no larger than a tolerance. The main table reports DCCS at 10%, 50%, and 100%, corresponding to strict, relaxed, and fully relaxed deadline-loss regimes. Definitions and the scale-dependent DCCS@10% threshold are given in Appendix C.

5.2 Main Results

Table 1 reports the main results under workflow-level dynamic deadline tightness. The table should be read in a constraint-first way: reducing VM cost is useful only if deadline success remains close to the safety reference.

Table 1: Main results under workflow-level dynamic deadline tightness. S, M, and L denote small, medium, and large workflow-size settings. Deadline Gap and Cost Saving are computed relative to FASTEST-NEW-VM. DCCS@τ reports cost saving only when Deadline Gap is no larger than τ; "–" indicates that the constraint is not satisfied.

Method	Setting	Success ↑	Cost ↓	D-Gap ↓	C-Saving ↑	DCCS@10% ↑	DCCS@50% ↑	DCCS@100% ↑
FASTEST-NEW-VM	S	0.960	652.339	0.000	0.000	0.000	0.000	0.000
	M	0.967	1195.622	0.000	0.000	0.000	0.000	0.000
	L	0.947	2287.642	0.000	0.000	0.000	0.000	0.000
OHDS	S	0.523	32.733	0.455	0.950	–	0.950	0.950
	M	0.507	70.096	0.476	0.941	–	0.941	0.941
	L	0.523	138.125	0.447	0.940	–	0.940	0.940
SMWDSA	S	0.537	30.656	0.441	0.953	–	0.953	0.953
	M	0.530	66.672	0.452	0.944	–	0.944	0.944
	L	0.547	130.656	0.423	0.943	–	0.943	0.943
ET2FA	S	0.623	32.947	0.351	0.949	–	0.949	0.949
	M	0.553	68.813	0.428	0.942	–	0.942	0.942
	L	0.560	134.554	0.408	0.941	–	0.941	0.941
SPN-CWS	S	0.115	13.269	0.880	0.980	–	–	0.980
	M	0.097	24.044	0.899	0.980	–	–	0.980
	L	0.047	35.302	0.951	0.985	–	–	0.985
GATES	S	0.132	14.543	0.863	0.978	–	–	0.978
	M	0.091	25.019	0.906	0.979	–	–	0.979
	L	0.047	42.709	0.951	0.981	–	–	0.981
DEFT	S	0.125	14.353	0.869	0.978	–	–	0.978
	M	0.085	24.606	0.912	0.979	–	–	0.979
	L	0.041	39.892	0.956	0.983	–	–	0.983
RCDRL (ours)	S	**0.870**	582.803	**0.094**	0.107	**0.107**	0.107	0.107
	M	**0.890**	1023.949	**0.079**	0.144	**0.144**	0.144	0.144
	L	**0.860**	2184.208	**0.092**	0.045	**0.045**	0.045	0.045

The key comparison is the trade-off between Deadline Gap and Cost Saving. The heuristic and soft-constrained DRL baselines achieve large raw cost savings, but their Deadline Gaps fall far outside the strict tolerance, so they do not meet the DCCS@10% criterion. In contrast, RCDRL explicitly estimates task-deadline risk, filters high-risk actions, and minimizes VM cost only among safer choices. RCDRL is therefore the only method in Table 1 that meets the DCCS@10% criterion while achieving positive cost savings across all three scales. Appendix D further compares RCDRL with constrained RL baselines to evaluate its effectiveness in reducing VM cost while satisfying deadline constraints.

The heuristic and soft-constrained DRL baselines save about 94%–98% VM cost, but their Deadline Gaps (0.351–0.956) violate the strict tolerance. RCDRL instead saves 0.045–0.144 while keeping Deadline Gap at 0.079–0.094 (Figure 2); absolute results and variability appear in Appendix D. Lower saving on L reflects less safe slack for cost reduction.

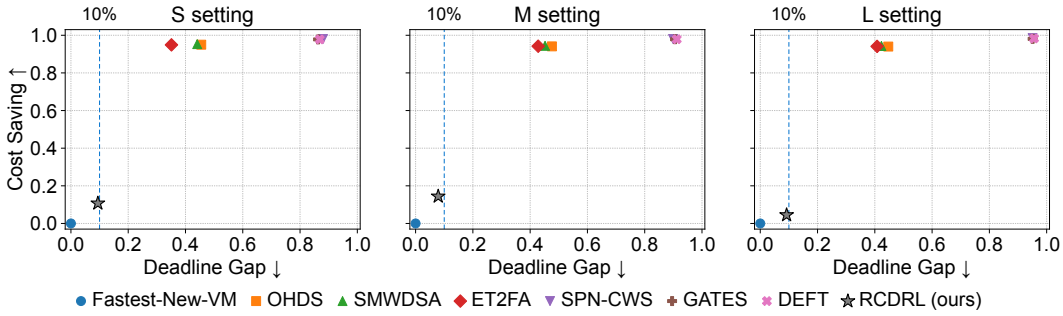

Figure 2: Deadline Gap–Cost Saving trade-off under workflow-level dynamic deadline tightness. The dashed vertical line marks the strict 10% Deadline Gap tolerance used by DCCS@10%.

5.3 Ablation Study

Table 2 reports ablation results relative to full RCDRL. The variants test the supervision signal, training strategy, and decision rule. Positive ΔD-Gap means worse deadline satisfaction than full RCDRL, while positive ΔC-Saving means larger VM-cost reduction.

Table 2: Ablation results relative to full RCDRL under S, M, and L workflow-size settings.

Variant	**S**		**M**		**L**	
	ΔD-Gap ↓	ΔC-Saving ↑	ΔD-Gap ↓	ΔC-Saving ↑	ΔD-Gap ↓	ΔC-Saving ↑
Full RCDRL	0.000	0.000	0.000	0.000	0.000	0.000
workflow-level soft-clipped target	0.047	0.021	0.061	0.026	0.073	0.018
binary workflow violation target	0.086	0.044	0.111	0.056	0.128	0.049
joint end-to-end training	0.032	0.013	0.044	0.017	0.058	0.011
Huber deadline loss	0.010	0.003	0.014	0.006	0.017	0.004
w/o deadline filtering	0.286	0.238	0.342	0.251	0.389	0.276
cost-only fallback	0.025	0.013	0.033	0.020	0.046	0.018
risk-first fallback	-0.006	-0.027	-0.009	-0.034	-0.007	-0.024

Coarser workflow-level and binary supervision worsen ΔD-Gap by 0.047–0.073 and 0.086–0.128, respectively; joint training also worsens deadline satisfaction, while Huber loss has a smaller effect. Removing deadline filtering gives the largest increase in ΔD-Gap (0.286/0.342/0.389 on S/M/L), confirming that cost-only scheduling accepts more deadline misses. Cost-only fallback is more aggressive, while risk-first fallback sacrifices savings for safety. Further discussion and additional comparisons appear in Appendices C and D.

6 Conclusions and Limitations

This paper presents RCDRL for deadline-constrained, cost-aware dynamic workflow scheduling. RCDRL learns reusable deadline-aware representations through dense task-level supervision and uses them to support long-horizon VM-cost optimization. Its safe-first policy prioritizes actions with low predicted deadline risk before minimizing VM rental cost. Experiments demonstrate that RCDRL achieves meaningful cost savings while satisfying the adopted DCCS@10% criterion across the evaluated workflow settings.

RCDRL also has limitations. First, task-level deadline supervision may not directly capture long-horizon workflow deadline outcomes, and selecting actions with low predicted risk does not guarantee workflow-level deadline satisfaction. Second, DCCS@10% reflects a specific trade-off between deadline success and cost savings, which may not be suitable for all applications. Stricter requirements may limit cost-saving opportunities, while more relaxed ones may permit unacceptable deadline violations. Future work will investigate more adaptive deadline-risk control and evaluate RCDRL under different constraint levels, broader workload distributions, and practical scheduling scenarios.

Acknowledgments and Disclosure of Funding

We thank Victoria University of Wellington for providing computational resources, our collaborators for their valuable discussions and suggestions, and the reviewers for their constructive feedback. This research received no dedicated project funding. The authors declare no competing interests.

References

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning*, pages 22–31, 2017.

Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.

Rajkumar Buyya, James Broberg, and Andrzej Goscinski. *Cloud computing*. John Wiley & Sons, Inc., Hoboken, New Jersey, Canada, 2011a.

Rajkumar Buyya, Saurabh Kumar Garg, and Rodrigo N Calheiros. Sla-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions. In *2011 international conference on cloud and service computing*, pages 1–10. IEEE, 2011b.

Dongsheng Ding, Kaiqing Zhang, Tamer Basar, and Mihailo R. Jovanovic. Natural policy gradient primal-dual method for constrained markov decision processes. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Tingting Dong, Fei Xue, Chuangbai Xiao, and Jiangjiang Zhang. Workflow scheduling based on deep reinforcement learning in the cloud environment. *Journal of Ambient Intelligence and Humanized Computing*, 12(12):10823–10835, 2021.

Guisheng Fan, Xingpeng Chen, Zengpeng Li, Huiqun Yu, and Yingxue Zhang. An energy-efficient dynamic scheduling method of deadline-constrained workflows in a cloud environment. *IEEE Transactions on Network and Service Management*, 20(3):3089–3103, 2022.

Zhengxin Fang, Hui Ma, Gang Chen, and Rajkumar Buyya. Hgraphscale: Hierarchical graph learning for autoscaling microservice applications in container-based cloud computing. *IEEE Transactions on Services Computing*, 2026a.

Zhengxin Fang, Hui Ma, Gang Chen, and Shiping Chen. Star: Spatial-temporal autoscaling for cloud applications with deep reinforcement learning. *Expert Systems with Applications*, 2026b.

Yong Liang Goh, Zhiguang Cao, Yining Ma, Jianan Zhou, Mohammed Haroon Dupty, and Wee Sun Lee. Shield: Multi-task multi-distribution vehicle routing solver with sparsity and hierarchy. *arXiv preprint arXiv:2506.08424*, 2025.

Yan Gu, Feng Cheng, Lijie Yang, Junhui Xu, Xiaomin Chen, and Long Cheng. Cost-aware cloud workflow scheduling using drl and simulated annealing. *Digital Communications and Networks*, 10(6):1590–1599, 2024.

Victoria Huang, Chen Wang, Hui Ma, Gang Chen, and Kameron Christopher. Cost-aware dynamic multi-workflow scheduling in cloud data center using evolutionary reinforcement learning. In *International Conference on Service-Oriented Computing*, pages 449–464. Springer, 2022.

Shadi Ibrahim, Bingsheng He, and Hai Jin. Towards pay-as-you-consume cloud computing. In *2011 IEEE International Conference on Services Computing*, pages 370–377. IEEE, 2011.

Dian Meng, Zhiguang Cao, Jie Gao, Yaoxin Wu, and Yaqing Hou. Uniteformer: Unifying node and edge modalities in transformers for vehicle routing problems. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Jiahui Pan and Yi Wei. A deep reinforcement learning-based scheduling framework for real-time workflows in the cloud environment. *Expert Systems with Applications*, 255:124845, 2024.

Yuxin Pan, Zhiguang Cao, Chengyang Gu, Liu Liu, Peilin Zhao, Yize Chen, and Fangzhen Lin. Multi-task vehicle routing solver via mixture of specialized experts under state-decomposable mdp. *arXiv preprint arXiv:2510.21453*, 2025.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Ya Shen, Gang Chen, Hui Ma, and Mengjie Zhang. Cost-aware dynamic cloud workflow scheduling using self-attention and evolutionary reinforcement learning. In *International Conference on Service-Oriented Computing*, pages 3–18. Springer, 2024.

Ya Shen, Gang Chen, Hui Ma, and Mengjie Zhang. Gates: Cost-aware dynamic workflow scheduling via graph attention networks and evolution strategy. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pages 8635–8643, 2025.

Ya Shen, Gang Chen, Hui Ma, and Mengjie Zhang. Deft scheduling of dynamic cloud workflows with varying deadlines via mixture-of-experts. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=yVFOdLjd7V.

Ya Shen, Gang Chen, Hui Ma, and Mengjie Zhang. Glare: Graph learning assisted deep reinforcement learning for cost-aware dynamic multi-workflow scheduling. *IEEE Transactions on Services Computing*, 2026b. doi: 10.1109/TSC.2026.3749653.

Zaixing Sun, Boyu Zhang, Chonglin Gu, Ruitao Xie, Bin Qian, and Hejiao Huang. Et2fa: A hybrid heuristic algorithm for deadline-constrained workflow scheduling in cloud. *IEEE Transactions on Services Computing*, 16(3):1807–1821, 2022.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3 (1):9–44, 1988.

Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. Reward constrained policy optimization. In *International Conference on Learning Representations*, 2019.

Jiaqi Wang, Zhiguang Cao, Peng Zhao, Rui Cao, Yubin Xiao, Yuan Jiang, and You Zhou. Learning memory-enhanced improvement heuristics for flexible job shop scheduling. *arXiv preprint arXiv:2603.02846*, 2026.

Xiangjie Xiao, Cong Zhang, Wen Song, and Zhiguang Cao. Resched: Rethinking flexible job shop scheduling from a transformer-based architecture with simplified states. *arXiv preprint arXiv:2603.07020*, 2026.

Yifan Yang, Gang Chen, Hui Ma, Cong Zhang, Zhiguang Cao, and Mengjie Zhang. Graph assisted offline-online deep reinforcement learning for dynamic workflow scheduling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4PlbIfmX9o.

Lingjuan Ye, Yuanqing Xia, Liwen Yang, and Yufeng Zhan. Dynamic scheduling stochastic multiworkflows with deadline constraints in clouds. *IEEE Transactions on Automation Science and Engineering*, 20(4):2594–2606, 2022.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Advances in neural information processing systems*, 33:1621–1632, 2020.

Linquan Zhang, Zongpeng Li, and Chuan Wu. Dynamic resource provisioning in cloud computing: A randomized auction approach. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 433–441. IEEE, 2014.

Appendix Contents

A Deadline-Constrained Cost-Aware Dynamic Workflow Scheduling (D-CADWS) Formulation **12**

A.1 Dynamic Cloud Workflows 12

A.2 Task Deadlines as Local Sufficient Conditions 13

A.3 Cloud Environment and VM Rental Cost 14

A.4 Workflow-Level Deadline Requirement 14

A.5 Optimization Objective of D-CADWS 15

B The full training process of RCDRL **15**

C Additional Experimental Details **16**

C.1 Environment Configuration 16

C.2 Assumptions for FASTEST-NEW-VM as the Deadline-Success Upper Reference . . 17

C.3 Metric Definitions 17

C.4 Additional Ablation Discussion 18

D Further Investigations and Analysis **19**

D.1 Performance Across Multiple Runs 19

D.2 Comparison with Constrained Reinforcement Learning Methods 19

D.3 Controlled Ablations: Direct Rules and Learned Components 20

A Deadline-Constrained Cost-Aware Dynamic Workflow Scheduling (D-CADWS) Formulation

This appendix provides a more complete formal definition of D-CADWS, complementing the concise description in Section 3. It also explains the relationship between workflow deadlines and task deadlines, which is used by RCDRL to obtain action-level supervision.

A.1 Dynamic Cloud Workflows

A D-CADWS instance consists of a set of workflows $\mathcal{W} = \{W_1, \ldots, W_N\}$ that arrive dynamically over time. Each workflow $W_i \in \mathcal{W}$ is modeled as a directed acyclic graph (DAG):

$$W_i = (\mathcal{O}_i, \mathcal{E}_i),$$

where $\mathcal{O}_i = \{O_{i1}, \ldots, O_{i|\mathcal{O}_i|}\}$ is the set of tasks and $\mathcal{E}_i$ is the set of precedence edges. An edge $(O_u, O_v) \in \mathcal{E}_i$ means that task O_v cannot start until task O_u has completed.

Each task $O_{ij} \in \mathcal{O}_i$ has a computational demand $cd_{ij} \in \mathbb{R}_+$. A task is *ready* when all predecessor tasks have completed. Following the no-delay scheduling principle commonly adopted in CADWS [Huang et al., 2022, Shen et al., 2024, 2025, 2026a], once a task becomes ready, the broker immediately schedules the current ready task. Therefore, at each decision step t, the scheduler only needs to determine which feasible VM should execute the current ready task O_t^*.

Each workflow W_i arrives at time a_i and is associated with a workflow deadline d_i under a Service Level Agreement (SLA) [Buyya et al., 2011b]. Following prior CADWS studies [Shen et al., 2024, 2025, 2026a], the deadline is defined as

$$d_i = a_i + \gamma_i \cdot \mathrm{minMakespan}(W_i), \tag{20}$$

where $\gamma_i \geq 1$ is the deadline relaxation coefficient of workflow W_i, and $\mathrm{minMakespan}(W_i)$ denotes the minimum execution time of W_i when all of its tasks are executed on the fastest available VM type without waiting. A smaller γ_i produces a tighter deadline. In our dynamic-deadline setting, this coefficient is assigned at the workflow level, not at the instance level; hence workflows with different deadline tightness can coexist in the same scheduling trajectory.

If task O_{ij} is assigned to VM m with processing speed $v(m)$, then its execution time is

$$ET(O_{ij}, m) = \frac{cd_{ij}}{v(m)}. \tag{21}$$

Its start time depends on both precedence feasibility and VM availability:

$$ST(O_{ij}) = \max\Big(a_i,\ \max_{O_{ik} \in \mathrm{Pred}(O_{ij})} CT(O_{ik}),\ AT(m)\Big), \tag{22}$$

where $\mathrm{Pred}(O_{ij})$ is the set of predecessor tasks of O_{ij}, and $AT(m)$ denotes the earliest time at which VM m becomes available. The task completion time is

$$CT(O_{ij}) = ST(O_{ij}) + ET(O_{ij}, m). \tag{23}$$

The workflow completion time is the finish time of its last completed task:

$$CT(W_i) = \max_{O_{ij} \in \mathcal{O}_i} CT(O_{ij}). \tag{24}$$

A.2 Task Deadlines as Local Sufficient Conditions

The original deadline requirement is specified at the workflow level through d_i. However, workflow-level success or failure is only observed after the whole workflow completes, which gives sparse and delayed feedback for individual scheduling decisions. To obtain action-level supervision, RCDRL uses task deadlines decomposed from the workflow deadline. We denote the task deadline of task O_{ij} by d_{ij}^{task}.

Upward-rank task-deadline allocation. For workflow W_i, let $D_i = d_i - a_i$ be its available deadline budget, let $e_{ij} = ET_{\mathrm{fast}}(O_{ij})$ be the execution time of task O_{ij} on the fastest VM type, and let $r_u(O_{ij})$ be the length of the longest remaining fastest-VM path starting at O_{ij} (including that task). With $R_i = \max_j r_u(O_{ij})$, the intermediate task deadline is constructed as

$$d_{ij}^{\mathrm{task}} = a_i + D_i \frac{R_i - r_u(O_{ij}) + e_{ij}}{R_i}. \tag{25}$$

The fraction allocates the deadline budget by each task's completion progress along the critical path. For an exit task, $r_u(O_{ij}) = e_{ij}$, hence $d_{ij}^{\mathrm{task}} = d_i$. As $r_u(O_{ij}) \geq e_{ij}$, intermediate task deadlines do not exceed d_i. For a ready task at scheduling time T_t, with candidate VM m available at $AT(m)$, the direct estimate of task completion is

$$\widehat{FT}(O_{ij}, m) = \max\{T_t, AT(m)\} + ET(O_{ij}, m). \tag{26}$$

The normalized lateness and bounded supervision target are then given by Eqs. (4)–(5). The direct estimate serves as a local training label, not as an estimate of eventual workflow completion or the long-horizon VM rental cost.

The connection between task deadline and workflow deadline. Task deadlines should be understood as local deadline budgets assigned within a workflow. They are not additional deadline requirements from the user; they are derived from the workflow deadline to make deadline reasoning available at the task level. A consistent task-deadline assignment satisfies the following sufficient-condition property: if all tasks are completed no later than their assigned task deadlines, and the task deadlines of exit tasks are no later than the workflow deadline, then the workflow deadline is satisfied. Formally, if:

$$CT(O_{ij}) \leq d_{ij}^{\mathrm{task}}, \quad \forall O_{ij} \in \mathcal{O}_i, \tag{27}$$

and for each exit task $O_{ij} \in \mathrm{Exit}(W_i)$,

$$d_{ij}^{\mathrm{task}} \leq d_i, \tag{28}$$

then

$$CT(W_i) = \max_{O_{ij} \in \mathrm{Exit}(W_i)} CT(O_{ij}) \le d_i. \tag{29}$$

Thus, satisfying all task deadlines is a sufficient local condition for satisfying the workflow deadline.

The converse is not required and generally does not hold. A task may exceed its assigned task deadline while the workflow still finishes before d_i, for example if later tasks are completed faster than expected. Therefore, task-deadline satisfaction is not treated as equivalent to workflow-deadline satisfaction. Instead, it is used as a dense local proxy that helps the scheduler avoid actions likely to accumulate delay. This is why RCDRL trains the deadline model with the task-deadline target in Eq. (5), while final performance is evaluated using workflow deadline success rate.

A.3 Cloud Environment and VM Rental Cost

At decision step t, the scheduler observes a set of feasible VM instances $\mathcal{M}_t$. These VM instances may include both previously leased active VMs and newly provisioned VMs. The size of $\mathcal{M}_t$ is therefore dynamic. Each VM instance $m \in \mathcal{M}_t$ is characterized by its processing speed $v(m)$ and rental price $c(m)$.

Under the pay-as-you-go cloud model [Ibrahim et al., 2011], new VM instances can be provisioned on demand. Assigning the current ready task O_t^* to a VM instance may therefore either reuse an active VM or trigger the leasing of a new VM. Although the decision is local to the current task, it may influence later queueing, resource contention, and the completion times of other workflows.

For a completed scheduling trajectory τ, let $\mathrm{Set}(\tau)$ denote the set of all VM instances leased during the episode. For each leased VM instance $m \in \mathrm{Set}(\tau)$, let $t_s(m,\tau)$ and $t_e(m,\tau)$ denote its lease start time and release time. Following the standard CADWS pay-as-you-go charging model [Shen et al., 2024], the total VM rental cost is

$$C^m(\tau) = \sum_{m \in \mathrm{Set}(\tau)} c(m) \cdot \left\lceil \frac{t_e(m,\tau) - t_s(m,\tau)}{3600} \right\rceil. \tag{30}$$

In the MDP formulation, this trajectory-level VM cost is represented step by step through immediate VM rental costs $c_t^m \ge 0$:

$$C^m(\tau) = \sum_{t=0}^{T-1} c_t^m. \tag{31}$$

This VM rental cost is the primary cost objective of D-CADWS.

A.4 Workflow-Level Deadline Requirement

For each workflow W_i, define its deadline-miss indicator under trajectory τ as

$$B_i(\tau) = \begin{cases} 1, & \text{if } CT(W_i) > d_i, \\ 0, & \text{otherwise.} \end{cases} \tag{32}$$

Instead of requiring a long dynamic trajectory to have zero missed workflows, we evaluate deadline satisfaction at the workflow level. The workflow deadline miss rate and success rate are

$$M(\tau) = \frac{1}{N(\tau)} \sum_{i=1}^{N(\tau)} B_i(\tau), \qquad S(\tau) = 1 - M(\tau), \tag{33}$$

where $N(\tau)$ is the number of completed workflows in τ. This definition is more suitable for dynamic CADWS than a trajectory-level all-or-nothing feasibility indicator, because a single missed workflow should not make the whole trajectory uninformative.

When violation severity is needed for diagnosis, we also use normalized workflow lateness:

$$\ell_{W_i}(\tau) = \frac{\max\{0, CT(W_i) - d_i\}}{\max\{d_i - a_i, \epsilon_{\mathrm{den}}\}}, \tag{34}$$

and its soft-clipped version

$$r_{W_i}(\tau) = \frac{\ell_{W_i}(\tau)}{1 + \ell_{W_i}(\tau)}. \tag{35}$$

These quantities are useful diagnostic metrics for how severe missed deadlines are. They are not used as the Stage-1 target of RCDRL, which instead uses the task-deadline target in Eq. (5).

A.5 Optimization Objective of D-CADWS

Under the above definitions, D-CADWS is a constrained cost-minimization problem. The policy should minimize expected VM rental cost while keeping the expected workflow deadline miss rate below a tolerance ϵ_{miss}:

$$\min_{\pi} \ \mathbb{E}_{\tau\sim\pi}[C^m(\tau)] \qquad \text{s.t.} \qquad \mathbb{E}_{\tau\sim\pi}[M(\tau)] \le \epsilon_{\text{miss}}. \tag{36}$$

Equivalently, the policy should achieve an expected workflow deadline success rate of at least $1-\epsilon_{\text{miss}}$ while minimizing VM rental cost. This formulation differs from previous CADWS definitions that merge VM cost and deadline penalty into a single unconstrained objective [Huang et al., 2022, Shen et al., 2024, 2025, 2026a]. RCDRL follows the constrained view by modeling task-deadline risk and VM cost separately, then deriving a safe-first policy that prioritizes deadline-risk control before VM-cost minimization.

B The full training process of RCDRL

Algorithm 1 gives the complete training procedure of RCDRL. It complements the two-phase alternating training strategy described in Section 4.5. At each outer iteration, RCDRL first clears the rollout buffer $\mathcal{D}$ and collects new trajectories using the current behavior policy μ_k. This fresh-buffer design avoids training on outdated transitions from old policies, keeping the deadline-aware representation and VM-cost critic aligned with the current policy-induced state-action distribution.

After data collection, RCDRL computes task-deadline targets and updates the predictive deadline model by supervised learning. It then freezes the encoder f_θ and trains only the VM-cost critic head q_β^{vm} by TD learning, so cost estimation can improve without disturbing the deadline-aware representation. Finally, the updated deadline model and VM-cost critic derive a safe-first policy, around which the next behavior policy μ_{k+1} is formed by ϵ-greedy exploration.

Algorithm 1 RCDRL with Two-Phase Alternating Training

Require: D-CADWS environment Env, rollout buffer $\mathcal{D}$, outer iterations K, deadline model updates U_{dea}, VM-cost critic updates U_{vm}, risk threshold δ, fallback ratio ρ, and exploration rate ϵ

Initialize predictive deadline model with encoder f_θ and risk head g_ϕ^{dea}, and initialize VM-cost critic q_β^{vm}
Construct an initial behavior policy μ_0
for $k = 0$ to $K-1$ **do**
    $\mathcal{D}$ is cleared before the new transitions collected
    Roll out μ_k in Env and store transitions in $\mathcal{D}$ (Eq. (18))
    Compute task-deadline targets y_t^{task} for transitions in $\mathcal{D}$ */* Stage 1 */* (Eq. (5))
    // Phase 1: predictive deadline-model learning
    for $u = 1$ to U_{dea} **do**
        Sample a minibatch from $\mathcal{D}$
        Update f_θ and g_ϕ^{dea} by minimizing $\mathcal{L}_{\text{dea}}$ */* Stage 2 */* (Eq. (9))
    end for
    Freeze f_θ and define $\hat{R}_{\text{task}}$ by Eq. (8)
    // Phase 2: VM-cost critic learning
    for $u = 1$ to U_{vm} **do**
        Sample a minibatch from $\mathcal{D}$
        Select a_{t+1}^* with the current safe-first policy (Eqs. (14)–(17))
        Form TD targets $y_t^{\text{vm,TD}}$ using Eq. (12)
        Update q_β^{vm} by minimizing $\mathcal{L}_{\text{vm}}$ */* Stage 3 */* (Eq. (13))
    end for
    Derive the safe-first policy π using $\hat{R}_{\text{task}}$ and $\hat{Q}_{\text{vm}}$ */* Stage 4 */* (Eqs. (14)–(17))
    Set μ_{k+1} by ϵ-greedy exploration around π (Eq. (19))
end for
return the learned scheduling policy π

C Additional Experimental Details

This appendix provides the experimental details omitted from the main text for space. It includes the environment configuration, the assumptions behind the safety-oriented reference policy, and the formal metric definitions.

C.1 Environment Configuration

Simulator and dynamic workload generation. We use the dynamic cloud workflow scheduling simulator adopted in recent CADWS studies [Shen et al., 2025, 2026a]. Each scheduling instance contains multiple workflows that arrive over time and share the same VM pool. Workflow arrivals follow a Poisson process with rate $\lambda = 0.01$. Each instance contains 10 workflows. When a workflow arrives, its tasks are released according to the precedence constraints of its DAG, and the scheduler repeatedly assigns the current ready task to one feasible VM. The simulator follows the no-delay scheduling protocol described in Appendix A: once a task becomes ready, the scheduler immediately makes a VM-assignment decision.

Workflow patterns and size settings. We use four standard scientific workflow patterns: CyberShake, Montage, Inspiral, and SIPHT. To evaluate scalability, we construct three workflow-size settings, denoted by S, M, and L. The number of tasks used for each workflow pattern and size setting is given in Table 5, and the corresponding DAG structures are shown in Figure 3. All methods are evaluated under the same workflow patterns and size settings.

Workflow-level dynamic deadlines. Following the dynamic-deadline setting in DEFT [Shen et al., 2026a], the deadline relaxation coefficient is sampled at the workflow level rather than fixed for the whole instance. Specifically, for each workflow W_i, we sample γ_i from

$$\Gamma = \{1.0, 1.25, 1.5, 1.75, 2.0, 2.25, 3.0\},$$

and set its deadline by Eq. (20). Therefore, a single scheduling instance may contain workflows with different deadline tightness levels. This setting is more challenging than assigning one common deadline coefficient to the whole instance, because the scheduler must adapt its VM-selection decisions to heterogeneous deadline pressure within the same trajectory.

Training and testing protocol. For learning-based methods, training instances are generated from the same mixed-deadline workload distribution described above. All learning-based methods are trained on the same training-instance distribution and evaluated on the same fixed test set. The heuristic baselines do not require training and are evaluated directly on the same test instances. The test set contains 30 fixed instances for each workflow-size setting S, M, and L, and each test instance contains 10 workflows whose deadline coefficients are independently sampled from Γ. Testing is conducted over 10 independent runs. The complete protocol is summarized in Table 3. This design ensures that performance differences come from the scheduling policy rather than from different workload distributions or test instances.

Table 3: Training and evaluation protocol used in the experiments.

Item	**Configuration**
Workflow arrival process	Poisson process with arrival rate $\lambda = 0.01$
Workflows per instance	10 workflows
Workflow patterns	CyberShake, Montage, Inspiral, and SIPHT
Workflow-size settings	S, M, and L
Deadline coefficient set	$\Gamma = \{1.0, 1.25, 1.5, 1.75, 2.0, 2.25, 3.0\}$
Deadline assignment	γ_i is sampled independently for each workflow
Training instances	Same mixed-deadline distribution for all learning-based methods
Test instances	30 fixed instances per workflow-size setting
Independent runs	10 runs
Evaluation rule	All methods are evaluated on the same fixed test instances

Cloud resource configuration. The cloud environment provides heterogeneous VM types following a pay-as-you-go pricing model. A scheduling action may assign the ready task to an active VM or provision a new VM of an available type. The VM configuration is shown in Table 4. These VM types differ in processing capacity and hourly rental price, creating the central deadline–cost trade-off: faster VMs reduce waiting and execution time but increase VM rental cost, while cheaper VMs may save cost but increase deadline risk.

Table 4: Configuration of VM instances used in the cloud environment.

VM Type	vCPU/Memory (GB)	Cost ($/hour)
m5.large	2/8	0.096
m5.xlarge	4/16	0.192
m5.2xlarge	8/32	0.384
m5.4xlarge	16/64	0.768
m5.8xlarge	32/128	1.536
m5.12xlarge	48/192	2.304

Table 5: Workflow patterns and the number of tasks under each workflow-size setting.

Scale	CyberShake	Montage	Inspiral/SIPHT
Small	30	25	30
Medium	50	50	50/60
Large	100	100	100

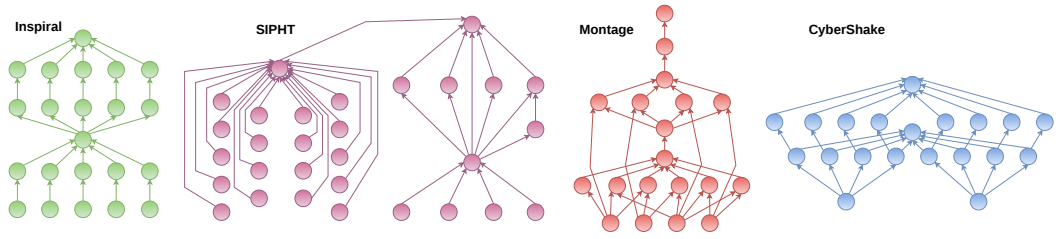

Figure 3: Workflow DAG patterns used in the experiments.

C.2 Assumptions for FASTEST-NEW-VM as the Deadline-Success Upper Reference

We use FASTEST-NEW-VM as an upper reference for workflow deadline success, not as a cost-efficient scheduling competitor. This reference policy is defined under the following assumptions: (i) the cloud has unlimited VM supply; (ii) whenever a task is scheduled, a new fastest VM can be provisioned immediately; and (iii) the newly provisioned VM has no existing task queue, so it introduces no additional queueing delay. The policy still respects workflow precedence constraints, task execution times, communication delays, and workflow arrivals.

Under these assumptions, assigning every ready task to a new fastest VM removes avoidable waiting caused by VM reuse and avoids choosing slower VM types. Therefore, within the VM-selection action space considered in this paper, FASTEST-NEW-VM gives an aggressive upper reference for workflow deadline success. Its purpose is to show how much deadline success can be obtained by heavy over-provisioning. It is not expected to have low VM rental cost, and a useful D-CADWS scheduler should stay close to its deadline success while saving much more VM cost.

C.3 Metric Definitions

For method A under workflow-size setting $z \in \{S, M, L\}$, let S_A^z denote the workflow deadline success rate and C_A^z denote the VM rental cost. Let S_{fast}^z and C_{fast}^z denote the corresponding values

of FASTEST-NEW-VM. As explained in Appendix C.2, FASTEST-NEW-VM is used as a deadline-success upper reference because it aggressively provisions a new fastest VM for each scheduled task.

The normalized Deadline Gap and Cost Saving are defined as

$$\mathrm{DeadlineGap}_z(A) = \frac{S^z_{\mathrm{fast}} - S^z_A}{S^z_{\mathrm{fast}}}, \tag{37}$$

$$\mathrm{CostSaving}_z(A) = 1 - \frac{C^z_A}{C^z_{\mathrm{fast}}}. \tag{38}$$

A smaller Deadline Gap means that method A stays closer to the safety reference in workflow deadline success. A larger Cost Saving means that method A reduces more VM rental cost relative to the safety reference.

Because deadline satisfaction is the primary requirement in D-CADWS, cost saving is counted only when the deadline loss is within a tolerance τ. We define deadline-constrained cost saving as

$$\mathrm{DCCS}_{z,\tau}(A) = \begin{cases} \mathrm{CostSaving}_z(A), & \text{if } \mathrm{DeadlineGap}_z(A) \le \tau, \\ \text{infeasible}, & \text{otherwise.} \end{cases} \tag{39}$$

The main results report DCCS at $\tau \in \{10\%, 50\%, 100\%\}$.

DCCS@10% and the workflow miss-rate constraint. For the strict criterion, $\mathrm{DeadlineGap}_z(A) \le 0.10$ is equivalent to $S^z_A \ge 0.90 S^z_{\mathrm{fast}}$. Because $M^z_A = 1 - S^z_A$, Eq. (3) can be evaluated at the scale-dependent miss-rate tolerance $\epsilon^z_{\mathrm{miss}} = 1 - 0.90 S^z_{\mathrm{fast}}$.

For the ablation analysis in Section 5.3, we compute the differences in Deadline Gap and Cost Saving between each ablation variant A and full RCDRL for each workflow-size setting $z \in \{S, M, L\}$:

$$\Delta\text{D-Gap}_z(A) = \mathrm{DeadlineGap}_z(A) - \mathrm{DeadlineGap}_z(\mathrm{RCDRL}), \tag{40}$$

$$\Delta\text{C-Saving}_z(A) = \mathrm{CostSaving}_z(A) - \mathrm{CostSaving}_z(\mathrm{RCDRL}). \tag{41}$$

A positive ΔD-Gap indicates worse deadline satisfaction, while a positive ΔC-Saving indicates greater VM-cost savings compared with full RCDRL.

C.4 Additional Ablation Discussion

This section provides additional interpretation of the ablation results in Table 2. The ablations are designed to isolate the effect of three components of RCDRL: the deadline-supervision target, the two-phase training strategy, and the safe-first decision rule.

First, the supervision target has a clear effect on deadline satisfaction. Replacing the task-level soft-clipped target with a workflow-level soft-clipped target consistently increases Deadline Gap across all three settings. This indicates that terminal workflow-level feedback is too coarse to identify which VM assignment is risky at a particular decision step. The binary workflow violation target performs even worse, because it removes lateness severity information and only tells the model whether the workflow eventually misses its deadline. These results support the use of task-level soft-clipped lateness as a denser action-level supervision signal for learning deadline-aware representations.

Second, the training strategy matters. Joint end-to-end training increases Deadline Gap in all workflow-size settings, showing that updating the encoder together with the VM-cost critic can weaken the deadline-aware features learned from supervised deadline signals. This supports the two-phase design of RCDRL: the encoder is first trained to represent deadline risk and is then frozen when the VM-cost critic is learned by TD learning. In contrast, replacing the default BCE-style deadline objective with Huber loss causes only a small degradation. This suggests that the main benefit comes from the granularity of the task-level target and the separation between deadline representation learning and cost critic learning, rather than from a minor loss-function choice.

Third, the decision rule is essential for the constrained objective. Removing deadline filtering gives the largest increase in Deadline Gap, while also increasing Cost Saving. This confirms the central failure mode of cost-driven scheduling: a policy can reduce VM cost by choosing cheaper actions that create many deadline misses. The fallback variants show a more fine-grained trade-off. The cost-only

fallback is more aggressive because it ignores deadline risk when no action is predicted to be safe. The risk-first fallback slightly improves Deadline Gap but reduces Cost Saving, indicating that it is more conservative than necessary. The top-ratio-then-cost fallback used by RCDRL provides a balanced compromise: it first restricts the decision to relatively safer actions and then still minimizes VM cost within that relaxed set.

D Further Investigations and Analysis

We report the additional experiments to analyze the performance of RCDRL. Unless stated otherwise, S, M, and L refer to the same workflow-size settings as in Section 5, and feasibility means DCCS@10% as defined in Appendix C.3. The main paper retains its original comparative evaluation; the following experiments provide complementary evidence.

D.1 Performance Across Multiple Runs

Table 6 reports the mean and standard deviation of RCDRL over 10 independent runs. Across all three workflow scales, RCDRL maintains a deadline gap below 10% while reducing VM rental cost. The small standard deviations also indicate consistent performance across runs.

Table 6: Absolute RCDRL performance (mean $\pm$ standard deviation over 10 independent runs). The miss rate is one minus workflow deadline success. D-Gap and C-Saving are relative to FASTEST-NEW-VM.

Scale	Workflow success	Miss rate	VM rental cost	D-Gap	C-Saving
S	0.870 ± 0.009	0.130 ± 0.009	582.8 ± 8.7	0.094	0.107
M	0.890 ± 0.008	0.110 ± 0.008	1023.9 ± 14.3	0.079	0.144
L	0.860 ± 0.010	0.140 ± 0.010	2184.2 ± 23.8	0.092	0.045

D.2 Comparison with Constrained Reinforcement Learning Methods

To compare with methods explicitly designed to handle constraints, we adapt four approaches to the D-CADWS VM-selection action space: *Penalty-tuned PPO*, which uses a validation-tuned fixed deadline penalty [Schulman et al., 2017]; *Adaptive Lagrangian*, an RCPO-style PPO variant with an adaptive multiplier [Tessler et al., 2019]; *Constrained Policy Optimization* (CPO), retaining its constrained KL trust-region update [Achiam et al., 2017]; and *Primal-dual*, an NPG-PD adaptation with natural-gradient policy updates and projected dual updates [Ding et al., 2020]. All methods are evaluated using the same simulator, training and test instances, interaction budget, seeds, validation procedure, and DCCS@10% threshold. The constrained mechanisms differ; the underlying scheduling task is the same.

Table 7: Comparison with constraint-aware RL baselines. Each cell gives workflow success / VM rental cost. Feasible scales counts which of S, M and L satisfy DCCS@10%.

Method	S: success / cost	M: success / cost	L: success / cost	Feasible scales
Penalty-tuned PPO	0.861 / 548.6	0.868 / 972.4	0.848 / 2085.3	0/3
Adaptive Lagrangian	0.872 / 612.1	0.883 / 1090.4	0.862 / 2256.4	3/3
CPO	0.884 / 628.8	0.899 / 1125.6	0.875 / 2278.9	3/3
Primal-dual	0.869 / 603.5	0.878 / 1078.2	0.856 / 2237.1	3/3
RCDRL (ours)	**0.870 / 582.8**	**0.890 / 1023.9**	**0.860 / 2184.2**	**3/3**

As Table 7 shows, penalty-tuned PPO obtains a lower rental cost but is infeasible on all three scales. Among baselines feasible on every scale, the primal-dual method achieves the lowest cost; RCDRL reduces this cost by 3.4%, 5.0%, and 2.4% on S, M, and L, respectively, while also obtaining slightly higher workflow success. This comparison distinguishes feasible cost reduction from cheap but constraint-violating solutions.

D.3 Controlled Ablations: Direct Rules and Learned Components

To further evaluate the benefits of deadline-aware representation learning, we compare RCDRL with four alternative designs that examine the roles of deadline information, representation learning, and long-horizon cost estimation. *Direct target + immediate price* directly filters VM choices using the analytical task-deadline target and selects the lowest-cost option based on immediate price. *Direct target + generic critic* uses the same deadline filter but estimates future VM cost without deadline-related features. *Direct target + TD-only encoder* also uses direct deadline filtering but includes deadline-related raw features and trains its encoder solely through VM-cost temporal-difference (TD) learning. *Learned risk + immediate price* retains the learned deadline-risk filter but selects VMs based on immediate price rather than future cost estimates.

Table 8: Controlled component ablations. Each cell gives workflow success / VM cost. All listed variants satisfy DCCS@10% on S, M, and L.

Variant	S: success / cost	M: success / cost	L: success / cost
Direct target + immediate price	0.882 / 628.5	0.902 / 1140.2	0.873 / 2274.8
Direct target + generic critic	0.874 / 611.6	0.889 / 1088.7	0.861 / 2248.9
Direct target + TD-only encoder	0.871 / 604.9	0.885 / 1075.8	0.857 / 2239.4
Learned risk + immediate price	0.886 / 619.8	0.904 / 1110.6	0.876 / 2260.2
Full RCDRL (ours)	**0.870 / 582.8**	**0.890 / 1023.9**	**0.860 / 2184.2**

As shown in Table 8, the full RCDRL has the lowest cost of the five feasible variants on every scale. Relative to the direct filter with immediate price, RCDRL reduces cost by 7.3%, 10.2%, and 4.0% on S, M, and L, with a 0.012–0.013 absolute reduction in workflow success. The direct target gives useful local information, but it neither learns reusable action-conditioned deadline features nor evaluates downstream VM cost. The controlled comparisons support keeping both the supervised deadline-aware representation and the long-horizon VM cost critic.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims made in the abstract and introduction are empirically verified in Section 5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We mention our limitations in Section 6.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: The paper does not present formal theoretical results such as theorems, lemmas, or convergence guarantees. It provides problem definitions, optimization objectives, model equations, and an algorithmic procedure rather than theorem-proof results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We disclose essential information needed for reproduction in the paper.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We will provide data, code, and instructions for the final paper.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: We provide experimental details in the Appendix.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We provide our results in Section 5 and the Appendix.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)

- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide the experimental configuration in the Appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have read and understood the code of ethics and have made every effort to adhere to it.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This work does not need to discuss both positive and negative societal impacts. It does not impact society at large.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release high-risk models or datasets such as pretrained language models, image generators, or scraped Internet datasets. The proposed method is a scheduling algorithm evaluated in a simulated cloud workflow environment.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [No]

    Justification: We use publicly available datasets. We adhere to the licenses and terms of use as specified by their original creators.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The main contribution is an algorithmic method and its evaluation on simulated dynamic workflow scheduling benchmarks. We will release our codebase along with detailed README files.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This work does not use crowdsourcing or human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This work does not use crowdsourcing or human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: We used an LLM solely for editing assistance.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.