SWAGES: SYSTEM-WIDE ATTENTION AND GUIDED EVOLUTION FOR DYNAMIC WORKFLOW SCHEDULING

Anonymous authors
Paper under double-blind review

ABSTRACT

Cost-Aware Dynamic Workflow Scheduling (CADWS) repeatedly selects task–VM assignments for workflow tasks and heterogeneous virtual machines (VMs), while minimizing VM rental cost and SLA penalties. Existing learning-based schedulers face two key limitations. First, their policies often focus on individual tasks or workflows without explicitly considering other active workflows or alternative task–VM assignments, limiting system-wide task prioritization and resource allocation. Second, capturing these system-wide interactions requires larger policy models that are more difficult to train efficiently. Gradient-based deep reinforcement learning (DRL) can be unstable, whereas evolution strategies (ES) are more robust but become inefficient in high-dimensional policy spaces because of costly population rollouts and underused transition data. We propose **SWAGES** to address both limitations. Its **System-Wide Graph Attention Network (SWAN)** jointly models active workflows, ready tasks, VMs, and task–VM compatibility, enabling all possible task–VM assignments to be scored under a shared global context. Its **Gradient-Guided Evolution Strategies (GUIDES)** reuse rollout transitions to train twin critics and derive surrogate policy-gradient directions that span a low-dimensional subspace for fitness-based ES search. Extensive experiments show that SWAGES consistently outperforms representative heuristics and recent learning-based CADWS methods, including gradient-based DRL and ES-trained schedulers. Ablation studies further validate the effectiveness of both SWAN and GUIDES.

1 INTRODUCTION

Cloud computing has transformed large-scale scientific and engineering computing by providing flexible access to heterogeneous computational resources (Buyya et al., 2011). Many applications are organized as workflows, in which task dependencies are represented by *directed acyclic graphs* (DAGs). In dynamic cloud environments, workflows arrive continuously, while virtual machines (VMs) differ in processing capacity and rental cost, making *Cost-Aware Dynamic Workflow Scheduling* (CADWS) a challenging sequential decision problem (Shen et al., 2025). At each decision step, the scheduler must determine which ready task to execute on which VM while balancing rental costs against *Service Level Agreement* (SLA) penalties. These decisions are strongly coupled across workflows because every task–VM assignment affects resource availability, workflow progress, and the readiness of future tasks. Consequently, a locally favorable assignment may lead to poor system-wide performance in terms of cost or deadline satisfaction.

Traditional heuristic schedulers rely on manually designed scheduling rules (Arabnejad et al., 2018; Wu et al., 2017), while deep reinforcement learning (DRL) can learn adaptive scheduling policies through interaction with the cloud environment (Pan & Wei, 2024). However, many DRL-learned CADWS methods still make decisions from only a partial system view, often centering the policy on one ready task and its workflow (Yang et al., 2025; Shen et al., 2026). Other ready tasks and active workflows may be overlooked when evaluating a task–VM assignment, even though they share the same VM pool and influence future scheduling dynamics. A locally attractive assignment may therefore be suboptimal from the system perspective.

This creates a *system-wide scheduling challenge*. The scheduler must track the progress and unresolved dependencies of all active workflows, account for competition over shared VM resources,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

and compare possible task–VM assignments under a common context. Task prioritization and VM allocation must therefore be optimized jointly rather than treated as isolated local decisions.

Additionally, *training such a policy efficiently remains challenging*. Several recent CADWS methods (Shen et al., 2024; 2025; 2026) adopt Evolution Strategies (ES) to optimize scheduling policies directly from episodic fitness, avoiding policy-gradient estimation (Salimans et al., 2017). However, conventional ES compresses each costly rollout into a scalar fitness value and continues searching in the full policy parameter space, leaving rich transition information unused for guided policy updates.

To address these challenges, we propose **SWAGES**. Its **System-Wide Graph Attention Network (SWAN)** represents all active workflows, ready tasks, and available VMs under a shared system context, enabling system-wide scoring of all possible task–VM assignments. By comparing pairs across workflows within the same representation, SWAN jointly determines task priority and VM allocation rather than scheduling each task in isolation.

To improve policy-training efficiency, we propose **Gradient-Guided Evolution Strategies (GUIDES)**. Instead of perturbing the full policy parameter space blindly, GUIDES reuses rollout transitions to estimate surrogate policy-gradient directions that are more likely to improve the scheduling policy. Subsequent populations are sampled within the low-dimensional subspace spanned by these directions, while the policy itself remains updated solely from rollout fitness. Consequently, the robustness of fitness-based evolutionary optimization is retained, while gradient guidance improves sample efficiency and accelerates policy learning.

The main contributions of this work are threefold. **First**, we develop **SWAN**, a system-wide scheduling policy that represents all active workflows and directly compares all possible task–VM assignments under a shared system context, enabling joint task prioritization and VM allocation. **Second**, we propose **GUIDES**, which reuses population-rollout transitions to estimate surrogate policy-gradient directions for low-dimensional ES search, while retaining fitness-based policy updates for robust evolutionary optimization. **Third**, extensive experiments provide strong empirical support for SWAGES, which achieves the lowest mean scheduling cost across all evaluated workflow scales and deadline settings. Experiments and ablation studies validate the effectiveness of both SWAN and GUIDES.

2 Related Work

This section reviews two lines of research relevant to our study: learning-based CADWS and evolution strategies for policy optimization. A more comprehensive review of related studies is presented in Appendix A.

Learning-based CADWS. DRL has been increasingly applied to workflow scheduling because it can improve scheduling policies through direct interaction with the environment (Pan & Wei, 2024). Early CADWS methods used feed-forward networks (Huang et al., 2022). Recent studies further explored self-attention (Shen et al., 2024; Luo et al., 2025), graph neural networks (GNNs) (Wang et al., 2025; Shen et al., 2025), and Mixture-of-Experts (MoE) models (Shen et al., 2026). Despite increasingly sophisticated representations, most existing methods remain centered on a single ready task and its workflow, with limited awareness of other active workflows and alternative task–VM assignments. In contrast, **SWAN** represents all active workflows, ready tasks, and VM choices within a shared global context, enabling system-wide comparison of all possible task–VM assignments.

ES for policy search. Evolution strategies (ES) optimize neural policies through population-based perturbation and fitness evaluation (Salimans et al., 2017; Ye et al., 2025). Prior work improves ES through structured perturbations (Choromanski et al., 2018), surrogate objectives for sample reuse (Liu et al., 2019), and low-dimensional policy-parameter search (Choromanski et al., 2019). Guided ES (Maheswaranathan et al., 2019) further exploits surrogate gradients to steer policy-parameter search. However, it assumes that suitable surrogate gradients are already available and does not specify how to estimate them from the transition data generated by population rollouts. **GUIDES** seamlessly integrates twin-critic learning with evolutionary search. Rollout transitions are explicitly utilized to determine surrogate policy-gradient directions that jointly span a low-dimensional ES search subspace, while policy updates remain solely fitness-based.

3 PRELIMINARIES

3.1 COST-AWARE DYNAMIC WORKFLOW SCHEDULING (CADWS)

In CADWS, workflows arrive dynamically and are executed by a dynamic pool of heterogeneous virtual machines (VMs). At time t, let $\mathcal{W}_t$ denote the set of *active workflows*, where a workflow is considered active after its arrival and before the completion of all its tasks. For each active workflow $W_i \in \mathcal{W}_t$, we represent its task dependencies by a directed acyclic graph (DAG) $G_i = (O_i, E_i)$, where $O_i = \{o_{ij}\}_{j=1}^{n_i}$ is the set of n_i tasks and $E_i \subseteq O_i \times O_i$ is the set of precedence edges. An edge $(o_{ij}, o_{ik}) \in E_i$ means that task o_{ik} can execute only after task o_{ij} completes. A task is *ready* once all its predecessors have completed. At decision step t, let $\mathcal{W}_t$ denote the set of active workflows, $\mathcal{Q}_t$ the set of their ready tasks, and $\mathcal{M}_t$ the set of eligible VMs. The available task–VM assignments form the action set:

$$\mathcal{A}_t = \mathcal{Q}_t \times \mathcal{M}_t = \{(o, m) \mid o \in \mathcal{Q}_t,\ m \in \mathcal{M}_t\}, \tag{1}$$

from which the scheduler selects

$$a_t = (o_t, m_t) \in \mathcal{A}_t. \tag{2}$$

Executing tasks on VMs incurs VM rental cost. An SLA penalty also applies upon completing a workflow after its deadline. Let ΔC_t^{vm} and $\Delta C_t^{\mathrm{sla}}$ denote the corresponding incremental costs generated between decision steps t and $t+1$. CADWS seeks a scheduling policy that minimizes the expected total cost over an episode of T decision steps:

$$\pi^* = \arg\min_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{T-1} \left(\Delta C_t^{\mathrm{vm}} + \Delta C_t^{\mathrm{sla}}\right)\right]. \tag{3}$$

Detailed formulations for workflow execution, billing, deadlines, VM rental costs, and SLA penalties are provided in Appendix B.

3.2 MDP FORMULATION

We formulate CADWS as an episodic MDP $(\mathcal{S}, \mathcal{A}, P, r)$. At decision step t, the state $s_t \in \mathcal{S}$ contains global system, workflow-DAG, ready-task, VM, and task–VM compatibility information. The complete raw state features are provided in Appendix B.5. Given the selected action $a_t = (o_t, m_t)$, task o_t is assigned to VM m_t, and the environment transitions to s_{t+1} according to $P(s_{t+1} \mid s_t, a_t)$. Figure 1 summarizes the agent–environment interaction. The immediate reward is defined as the negative incremental cost:

$$r_t = -\left(\Delta C_t^{\mathrm{vm}} + \Delta C_t^{\mathrm{sla}}\right). \tag{4}$$

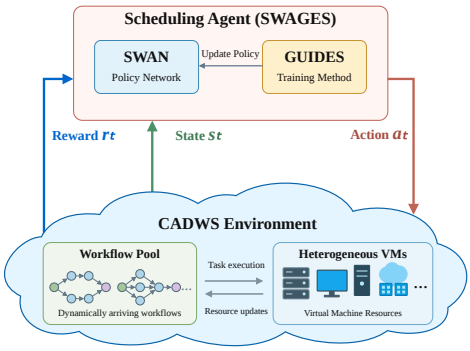

Figure 1: Agent–environment interaction in CADWS.

The total return is therefore

$$J(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} r_t\right] = -\mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} \left(\Delta C_t^{\mathrm{vm}} + \Delta C_t^{\mathrm{sla}}\right)\right]. \tag{5}$$

Maximizing $J(\pi_\theta)$ is thus equivalent to minimizing the CADWS objective in equation 3. The detailed MDP formulation is provided in Appendix B.4.

4 METHODOLOGY

SWAGES combines the **System-Wide Graph Attention Network (SWAN)** (Section 4.1) with **Gradient-Guided Evolution Strategies (GUIDES)** (Section 4.2). SWAN scores candidate task–VM assignments under a shared global context (Figure 2), while GUIDES uses rollout transitions for critic-guided policy perturbation and rollout fitness for policy updates.

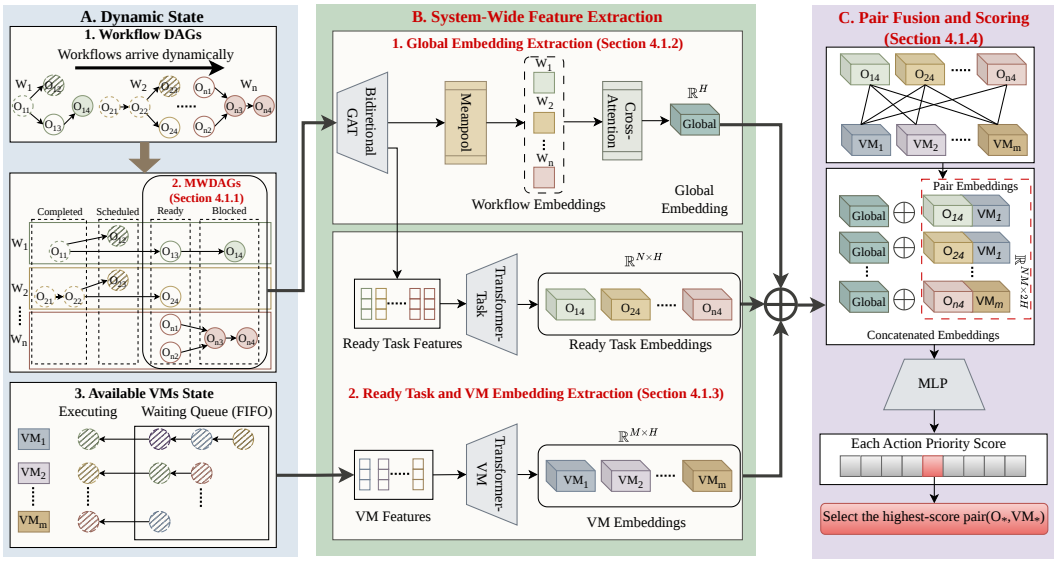

Figure 2: Overview of SWAN. **Block A** represents the dynamic system state containing MWDAGs of all active workflows and available VMs. **Block B** uses bidirectional graph attention and cross-attention to extract the global workflow representation, together with separate Transformers for ready tasks and VMs. **Block C** fuses these representations with task–VM compatibility features to score all possible assignments and selects the highest-scoring task–VM assignment. Blocks B and C are newly proposed components.

4.1 System-Wide Graph Attention Network

SWAN evaluates task–VM assignments through four stages, in line with Figure 2. First, each active workflow is reduced to a compact DAG containing only tasks that remain relevant to future scheduling decisions (Section 4.1.1). Second, bidirectional graph attention encodes the remaining task dependencies, while global cross-attention aggregates all active workflows into a system-wide representation (Section 4.1.2). Third, separate Transformers contextualize the ready-task and VM sets (Section 4.1.3). Finally, the global, task, VM, and task–VM compatibility representations are fused and scored for every possible assignment, enabling joint task prioritization and VM allocation (Section 4.1.4). Detailed information is provided in Appendix C.

4.1.1 Minimal Workflow DAG

At any decision step t, the tasks in each active workflow W_i are partitioned by scheduling status into four sets. $O_{i,t}^{\text{comp}}$ contains completed tasks, while $O_{i,t}^{\text{sched}}$ contains tasks already assigned to a VM, including those currently executing and those waiting to execute. $O_{i,t}^{\text{ready}}$ contains tasks eligible for scheduling, namely tasks whose predecessors have all completed. The remaining unfinished tasks form $O_{i,t}^{\text{blocked}}$, where at least one predecessor remains incomplete. Since completed and scheduled tasks no longer require scheduling decisions, we retain only the ready and blocked tasks when constructing the graph representation used by SWAN. We refer to this reduced representation of workflow W_i at decision step t as the **Minimal Workflow DAG (MWDAG)**. Its remaining task set is defined as $O_{i,t}^{\text{rem}} = O_{i,t}^{\text{ready}} \cup O_{i,t}^{\text{blocked}}$. The corresponding edge set retains only precedence dependencies between remaining tasks and is defined as $E_{i,t}^{\text{rem}} = (u, v) \in E_i \mid u, v \in O_{i,t}^{\text{rem}}$. Accordingly, the MWDAG of workflow W_i at decision step t is represented as $G_{i,t}^{\text{MW}} = (O_{i,t}^{\text{rem}}, E_{i,t}^{\text{rem}})$.

For multiple active workflows, SWAN therefore maintains a set of MWDAGs, with one MWDAG constructed for each active workflow. The MWDAG is updated after each scheduling decision, retaining only tasks and dependencies relevant to future scheduling (Xiao et al., 2026). Information from already assigned tasks is represented separately through readiness, VM availability, and task–VM timing features. As shown in Table 11 in Appendix H, removing resolved workflow history does not degrade scheduling performance, while reducing the representation and decision overhead.

4.1.2 Global Embedding Extraction via Bidirectional Graph Attention Network and Cross-Attention

Block B in Figure 2 encodes the MWDAGs of all active workflows using bidirectional graph attention and cross-attention, producing graph-aware task representations and a shared global context.

Let $X_{i,t}^{\mathrm{MW}}$ denote the node-feature matrix of $G_{i,t}^{\mathrm{MW}}$, and let $H_{i,t}^{(\ell)} \in \mathbb{R}^{|O_{i,t}^{\mathrm{rem}}| \times d_\ell}$ denote the node-representation matrix after layer ℓ, where d_ℓ is the representation dimension. We initialize $H_{i,t}^{(0)} = X_{i,t}^{\mathrm{MW}}$. A shared bidirectional Graph Attention Network (GAT) (Veličković et al., 2017) then encodes each active workflow using two directional message-passing branches. The forward branch propagates information from predecessors to successors along the original precedence edges, while the backward branch propagates information from successors to predecessors along the reversed edges. At layer ℓ, the two branch outputs are combined with a linear projection of each task's own representation:

$$H_{i,t}^{(\ell)} = \mathrm{ELU}\left(\frac{\mathrm{GAT}_{\mathrm{pred}}^{(\ell)}\left(H_{i,t}^{(\ell-1)}, E_{i,t}^{\mathrm{rem}}\right) + \mathrm{GAT}_{\mathrm{succ}}^{(\ell)}\left(H_{i,t}^{(\ell-1)}, (E_{i,t}^{\mathrm{rem}})^\top\right) + H_{i,t}^{(\ell-1)} W_{\mathrm{self}}^{(\ell)}}{3} \right). \tag{6}$$

This bidirectional design captures upstream requirements and downstream dependencies, while the third term preserves each task's own information. Let L denote the number of GAT layers. The final matrix $H_{i,t}^{(L)}$ contains the graph-aware task embeddings. Embeddings of ready tasks are passed to the ready-task Transformer, while mean pooling over all remaining tasks produces the workflow embedding:

$$h_{i,t}^{\mathrm{wf}} = \mathrm{MeanPool}\left(H_{i,t}^{(L)}\right). \tag{7}$$

Individual workflow embeddings capture intra-workflow dependencies but not their relative importance under the current workload, deadline pressure, and resource conditions. We therefore aggregate them using global-context-conditioned cross-attention. Let $q_t = f_{\mathrm{global}}(x_t^{\mathrm{global}}) \in \mathbb{R}^{d_h}$ denote the projected global features, and let $H_t^{\mathrm{wf}} = [h_{1,t}^{\mathrm{wf}}, \ldots, h_{|\mathcal{W}_t|,t}^{\mathrm{wf}}]^\top$ stack the active-workflow embeddings. Using q_t as the query and H_t^{wf} as both keys and values, we compute

$$u_t = \mathrm{LayerNorm}\left(q_t + \mathrm{MHA}(q_t, H_t^{\mathrm{wf}}, H_t^{\mathrm{wf}})\right), \tag{8}$$

$$h_t^{\mathrm{global}} = \mathrm{LayerNorm}(u_t + \mathrm{FFN}_{\mathrm{global}}(u_t)). \tag{9}$$

Because q_t reflects the current global conditions, cross-attention weights workflows by their scheduling relevance and produces a fixed-dimensional, order-independent global representation.

4.1.3 Ready Task and VM Embedding Extraction via Dual Transformers

A ready task should be evaluated relative to other tasks competing for execution, while a VM should be assessed relative to the alternative VMs available for assignment. SWAN therefore contextualizes the entire ready-task set and VM set using two Transformer encoders with the same architecture but independent parameters. Let X_t^{task} and X_t^{vm} denote their respective feature matrices, and let f_{task} and f_{vm} denote learnable input projections. The contextualized representations are

$$H_t^{\mathrm{task}} = \mathrm{Transformer}_{\mathrm{task}}\left(f_{\mathrm{task}}(X_t^{\mathrm{task}})\right), \tag{10}$$

$$H_t^{\mathrm{vm}} = \mathrm{Transformer}_{\mathrm{vm}}(f_{\mathrm{vm}}(X_t^{\mathrm{vm}})). \tag{11}$$

No positional encoding is used, preserving permutation equivariance. We denote the ith row of H_t^{task} by $h_{i,t}^{\mathrm{task}}$ and the jth row of H_t^{vm} by $h_{j,t}^{\mathrm{vm}}$. They represent ready task o_i and VM m_j, respectively, in the context of the other elements in their corresponding sets.

4.1.4 Task–VM Fusion and Scoring

For each candidate task–VM assignment $(o_i, m_j) \in \mathcal{A}_t$ at decision step t, its compatibility features $x_{i,j,t}^{\mathrm{comp}}$ characterize execution time, waiting time, completion slack, and incremental rental cost (see Appendix B.5 for detailed explanation). All these are encoded as $h_{i,j,t}^{\mathrm{comp}} = f_{\mathrm{comp}}(x_{i,j,t}^{\mathrm{comp}})$.

Let $w(i)$ denote the workflow containing o_i. SWAN represents each assignment by

$$z_{i,j,t} = h_t^{\text{global}} \| h_{i,t}^{\text{task}} \| h_{j,t}^{\text{vm}} \| h_{i,j,t}^{\text{comp}}, \tag{12}$$

where $\|$ stands for concatenation. The components in equation 12 capture the global system context, competition among ready tasks, alternative VM choices, and task–VM compatibility, respectively. A shared scoring network produces $\ell_{i,j,t} = f_{\text{score}}(z_{i,j,t})$, defining the stochastic policy:

$$\pi_\theta((o_i, m_j) \mid s_t) = \frac{\exp(\ell_{i,j,t})}{\sum_{(o_p,m_q)\in\mathcal{A}_t} \exp(\ell_{p,q,t})}. \tag{13}$$

Actions are sampled from this distribution during training, while at evaluation time, the scheduler selects $a_t = \arg\max_{(o_i,m_j)\in\mathcal{A}_t} \ell_{i,j,t}$. By scoring all possible task–VM assignments under the same global context, SWAN can jointly determine system-wide task priorities and VM allocations.

4.2 Gradient-Guided Evolution Strategies

GUIDES learns policy search guidance directly from population rollouts. Their transitions are stored in a replay buffer and reused to train twin critics, which produce multiple surrogate policy-gradient directions. These directions form a low-dimensional search subspace for subsequent ES perturbations, while the policy update remains determined solely by rollout fitness. The complete training procedure is given in Algorithm 1 in Appendix D.

Let $\theta_g \in \mathbb{R}^d$ denote the center-policy parameters at generation g. The first generation uses full-dimensional perturbations to collect initial experience. For $g > 1$, the critics and replay buffer construct a K-dimensional guidance basis $U_g \in \mathbb{R}^{d\times K}$, where $K \ll d$. Population policies are sampled around θ_g within this subspace. Their transitions are exploited to further train the critics, while their rollout fitness determines the ES update.

4.2.1 Replay-Based Twin-Critic Learning

Each population rollout stores transitions $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ in the replay buffer $\mathcal{D}$, where a_{t+1} is selected by the same rollout policy at s_{t+1}. We maintain two critics Q_{ϕ_1}, Q_{ϕ_2} and their target critics $Q_{\bar{\phi}_1}, Q_{\bar{\phi}_2}$. A minibatch of B transitions is sampled from $\mathcal{D}$, with the bth sample denoted by $(s_b, a_b, r_b, s'_b, a'_b, \delta_b)$. Its TD target is

$$y_b = r_b + \gamma_Q \min_{j\in\{1,2\}} Q_{\bar{\phi}_j}(s'_b, a'_b), \tag{14}$$

where γ_Q is used only for auxiliary critic learning. The critics minimize

$$\mathcal{L}_Q = \frac{1}{B}\sum_{b=1}^{B}\sum_{j=1}^{2} \text{Huber}\big(Q_{\phi_j}(s_b, a_b) - y_b\big). \tag{15}$$

Critic gradients are clipped, and target parameters are softly updated as $\bar{\phi}_j \leftarrow \tau\phi_j + (1-\tau)\bar{\phi}_j$. Taking the minimum of the two critics in equation 14 reduces optimistic value errors, while target networks and Huber loss further stabilize learning from replayed population experience.

4.2.2 Critic-Derived Guidance Subspace

For generation $g > 1$, GUIDES constructs K search directions around the center policy π_{θ_g} using independently sampled guidance batches $\{\mathcal{B}_k\}_{k=1}^K$, each containing B_G states from the replay buffer. For each state $s_b \in \mathcal{B}_k$, the twin critics evaluate every possible assignment $a \in \mathcal{A}(s_b)$ using $\widehat{Q}(s_b, a) = \min_{j\in\{1,2\}} Q_{\phi_j}(s_b, a)$. These critic values jointly define the following surrogate objective:

$$\mathcal{J}_k(\theta) = \frac{1}{B_G}\sum_{s_b\in\mathcal{B}_k}\sum_{a\in\mathcal{A}(s_b)} \pi_\theta(a \mid s_b)\,\widehat{Q}(s_b, a). \tag{16}$$

Maximizing $\mathcal{J}_k$ shifts policy probability toward assignments with larger critic-estimated returns $\widehat{Q}(s_b, a)$. We therefore use its normalized gradient at the center parameters θ_g as the kth search direction:

$$g_k = \frac{\nabla_\theta \mathcal{J}_k(\theta)|_{\theta=\theta_g}}{\|\nabla_\theta \mathcal{J}_k(\theta)|_{\theta=\theta_g}\|_2 + \varepsilon_{\text{norm}}}.$$

The K directions are then orthogonalized through QR decomposition:

$$[U_g, R_g] = \mathrm{QR}([g_1, \ldots, g_K]), \qquad U_g^\top U_g = I_K, \tag{17}$$

forming a K-dimensional critic-guided policy search subspace for updating θ_g.

4.2.3 Guided Population Sampling and Fitness-Based Update

For population member i at generation g, GUIDES samples

$$\epsilon_{g,i} = \begin{cases} \xi_{g,i}, & g = 1, \quad \xi_{g,i} \sim \mathcal{N}(0, I_d), \\ U_g z_{g,i}, & g > 1, \quad z_{g,i} \sim \mathcal{N}(0, I_K), \end{cases} \tag{18}$$

and forms $\theta_{g,i} = \theta_g + \sigma\epsilon_{g,i}$, where σ is the perturbation scale. The first generation uses full-space perturbations to bootstrap critic learning; subsequent generations search only within the critic-derived K-dimensional subspace.

Each sampled policy is evaluated over a complete episode using the negative undiscounted CADWS cost: $F_{g,i} = -\sum_{t=0}^{T_{g,i}-1}(\Delta C^{\mathrm{vm}}_{g,i,t} + \Delta C^{\mathrm{sla}}_{g,i,t})$. The fitness values are converted to centered ranks $\rho_{g,i} \in [-0.5, 0.5]$, reducing sensitivity to fitness scale (Shen et al., 2025). The center policy is then updated along the direction below using Adam:

$$\widehat{g}_{\mathrm{ES},g} = \frac{1}{N\sigma} \sum_{i=1}^{N} \rho_{g,i}\epsilon_{g,i}. \tag{19}$$

Applicability beyond CADWS. GUIDES is task-agnostic and requires only a differentiable stochastic policy, replay transitions, action-value critics, and rollout fitness. It can therefore be applied beyond CADWS, as demonstrated in Appendix E.

5 Experiments

We evaluate SWAGES through total-cost comparisons, generalization to larger workflows and tighter deadlines, and separate ablation studies of SWAN and GUIDES.

5.1 CADWS Experimental Setup

Environment. We use a CADWS simulator (Shen et al., 2025; Yang et al., 2025) with dynamically arriving workflows and heterogeneous VMs. Four scientific workflow patterns, CyberShake, Montage, Inspiral, and SIPHT, are evaluated at small, medium, and large scales (Deelman et al., 2015). VM types differ in processing capacity and rental cost. The deadline coefficient γ_{dl} controls deadline tightness, with smaller values imposing stricter deadlines. Detailed settings are provided in Appendix G.

Baselines. We include five baselines covering cost-aware heuristics and learning-based scheduling. *ProLiS* (Wu et al., 2017) minimizes cost under deadline constraints, while *GRP-HEFT* (Faragardi et al., 2019) jointly considers scheduling and resource provisioning under cloud cost constraints. *DRL-G* (Song et al., 2022) models operation–machine interactions with a graph neural network trained by proximal policy optimization (PPO) (Schulman et al., 2017). *SPN-CWS* (Shen et al., 2024) uses self-attention to model VM interactions. *GATES* (Shen et al., 2025) combines graph-based workflow representations with ES, making it the closest baseline to SWAGES. These learning-based comparisons cover both gradient-based and ES-based policy training.

SWAGES Parameters. We use a population size of $N = 40$, an initial ES learning rate of 0.001, and perturbation standard deviation $\sigma = 0.05$. The twin critics use a learning rate of 3×10^{-4}, auxiliary discount $\gamma_Q = 0.99$, target-update rate 0.005, batch size 256, replay capacity 500,000, and 500 updates per generation. We set $K = 16$ to construct the guidance basis U_g from 16 surrogate policy-gradient directions in equation 17. Hyperparameter sensitivity results are provided in Appendix G.

Training and Testing. SWAGES is trained for 1,000 generations on small-scale instances containing 30 workflows with $\gamma_{\mathrm{dl}} = 5$. The policy with the best training performance is then evaluated on small, medium, and large instances with $\gamma_{\mathrm{dl}} \in \{1, 1.5, 2, 2.5, 3\}$. These tighter deadlines create a harder test regime with less scheduling slack, allowing us to assess whether a policy trained under relaxed SLA conditions can meet stricter scheduling requirements. Results are averaged over 30 independent tests with matched random seeds.

5.2 Main Results

Total scheduling cost. Table 1 shows that SWAGES achieves the lowest mean cost in all 15 scale–deadline settings. Compared with ProLiS and GRP-HEFT, it reduces cost by 94.5% and 96.1% on average, respectively. Compared with the learning-based baselines GATES, SPN-CWS, and DRL-G, the corresponding average reductions are 56.0%, 59.2%, and 70.9%. All reported improvements over the compared methods are statistically significant.

Table 1: Total cost across workflow scales and deadline coefficients. Values are reported as *mean total cost* (*standard deviation*); standard deviations are omitted for the deterministic heuristics ProLiS and GRP-HEFT. Bolded results in each row are statistically significant under the Wilcoxon rank-sum test ($p < 0.05$). $(-)$ indicates that the corresponding method performs significantly worse than SWAGES.

Scale / γ_{dl}	ProLiS	GRP-HEFT	DRL-G	SPN-CWS	GATES	SWAGES
S / 1	795.31 (−)	1497.60 (−)	173.73(44.49) (−)	140.30(5.27) (−)	139.05(8.01) (−)	**70.44(3.07)**
S / 1.5	888.39 (−)	1366.12 (−)	167.08(40.19) (−)	138.09(7.45) (−)	129.14(12.21) (−)	**59.76(5.34)**
S / 2	783.11 (−)	1273.57 (−)	161.86(39.92) (−)	132.54(7.50) (−)	121.83(10.52) (−)	**55.98(4.22)**
S / 2.5	782.96 (−)	1207.53 (−)	157.53(39.64) (−)	128.48(6.61) (−)	115.69(7.83) (−)	**53.48(3.69)**
S / 3	752.78 (−)	1158.45 (−)	153.96(40.09) (−)	126.84(6.04) (−)	114.13(8.95) (−)	**51.27(3.09)**
M / 1	1725.90 (−)	2644.56 (−)	323.63(114.43) (−)	243.58(43.92) (−)	231.53(46.43) (−)	**106.82(4.51)**
M / 1.5	1916.63 (−)	2395.44 (−)	318.29(114.42) (−)	240.28(45.54) (−)	218.88(57.04) (−)	**95.90(5.61)**
M / 2	1806.13 (−)	2242.49 (−)	313.33(114.46) (−)	230.76(51.30) (−)	208.00(54.54) (−)	**91.87(5.73)**
M / 2.5	1791.93 (−)	2133.20 (−)	308.54(114.50) (−)	233.15(52.49) (−)	198.78(45.42) (−)	**88.90(5.67)**
M / 3	1834.11 (−)	2057.88 (−)	303.93(114.44) (−)	233.02(55.24) (−)	189.96(42.08) (−)	**86.30(6.25)**
L / 1	3434.03 (−)	5147.24 (−)	678.74(444.78) (−)	370.30(96.43) (−)	370.50(137.58) (−)	**159.83(10.26)**
L / 1.5	3896.60 (−)	4700.78 (−)	674.07(444.73) (−)	373.39(113.41) (−)	368.86(166.79) (−)	**148.86(10.18)**
L / 2	3687.11 (−)	4413.83 (−)	669.46(444.74) (−)	375.29(120.50) (−)	367.75(155.17) (−)	**143.98(10.90)**
L / 2.5	3576.39 (−)	4202.98 (−)	664.87(444.78) (−)	375.00(128.47) (−)	355.02(138.78) (−)	**141.59(11.52)**
L / 3	3565.48 (−)	4038.89 (−)	660.28(444.82) (−)	379.39(127.82) (−)	355.95(141.78) (−)	**138.51(11.84)**

Generalization across problem scales and deadlines. Although trained only on small-scale problem instances with $\gamma_{\mathrm{dl}} = 5$, SWAGES generalizes to larger workflows and tighter deadlines. Relative to GATES, its average cost reduction increases from 53.2% on S-scale instances to 55.1% on M-scale and 59.7% on L-scale instances. Even at the tightest tested deadline, $\gamma_{\mathrm{dl}} = 1$, SWAGES reduces the mean cost from 231.53 to 106.82 on M-scale instances and from 370.50 to 159.83 on L-scale instances.

Convergence and stability. Figure 3 compares the convergence and stability of the learning-based methods on S-scale workflows with $\gamma_{\mathrm{dl}} = 2$. SWAGES reaches a low-cost region within fewer generations and converges earlier than the competing methods. Across independent runs, it also achieves the lowest and most concentrated final-cost distribution. Together, these results show that SWAGES improves both convergence speed and run-to-run consistency, providing further evidence for the effectiveness of critic-guided ES search.

Appendix E further evaluates GUIDES against representative evolutionary policy-search methods on four continuous-control benchmarks. GUIDES outperforms the baselines on most of the tasks, while maintaining stable learning behavior. These results provide further evidence that critic-guided evolutionary search generalizes beyond CADWS to broader reinforcement learning problems.

Ablation studies. We evaluate both the SWAN policy architecture and the GUIDES search strategy under matched experimental settings. Table 2 shows that replacing MWDAGs with full work-

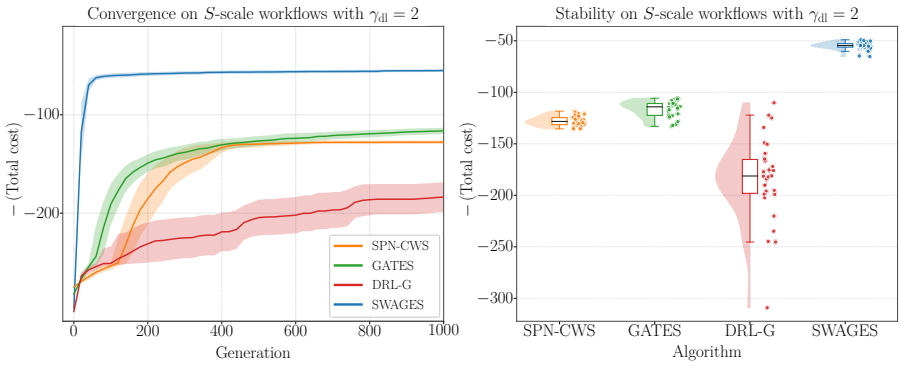

Figure 3: Convergence and run-to-run stability on S-scale workflows with $\gamma_{\mathrm{dl}} = 2$. Higher is better.

flow DAGs gives similar performance, while removing the Global module, Task–Transformer, assignment compatibility, or directional GAT consistently increases scheduling cost. Among these components, removing assignment compatibility causes the largest degradation at all three problem scales.

For GUIDES, α controls the contribution of full-space perturbations in mixed policy search, with $\alpha = 0$ corresponding to pure critic-guided subspace search. As α increases, both the mean total cost and variability consistently increase, with the performance gap widening as the workflow scale increases. These results support the proposed SWAN components and pure critic-guided policy subspace search. Detailed settings and analyses are provided in Appendices H.1 and H.2.

Table 2: Ablation results for the SWAN policy architecture and GUIDES policy search algorithm with $\gamma_{\mathrm{dl}} = 1$. Values are reported as *mean total cost* (*standard deviation*) over 10 tests.

Ablation	**Variant**	**S**	**M**	**L**
SWAGES	**SWAN + GUIDES (ours)**	**14.51 (0.25)**	20.32 (0.42)	**27.71 (0.73)**
Policy Architecture	Full Workflow DAGs	14.60 (0.12)	**20.31 (0.58)**	27.83 (1.01)
	w/o Global Module	14.75 (0.28)	21.28 (0.71)	28.76 (1.36)
	w/o Task–Transformer	14.78 (0.29)	21.30 (0.60)	28.82 (1.22)
	w/o Assignment Compatibility	14.85 (0.22)	21.39 (0.95)	29.30 (1.57)
	w/o Directional GAT	14.70 (0.12)	20.53 (0.54)	28.77 (0.84)
Policy Search	Mixed search ($\alpha = 0.2$)	15.38 (2.62)	22.64 (6.29)	31.88 (11.44)
	Mixed search ($\alpha = 0.5$)	15.52 (2.57)	22.73 (6.23)	32.04 (11.47)
	Mixed search ($\alpha = 0.8$)	16.18 (2.69)	23.78 (6.07)	34.13 (11.13)

6 Conclusion

This paper presented **SWAGES**, which combines system-wide attention with gradient-guided evolutionary learning for cost-aware dynamic workflow scheduling. SWAN jointly models active workflows, ready tasks, VMs, and task–VM compatibility within a shared global context, while GUIDES reuses population-rollout experience to construct low-dimensional search directions for efficient fitness-based evolutionary optimization. Extensive experiments demonstrate strong scheduling performance and generalization across workflow scales and deadline conditions. Results on continuous-control tasks further demonstrate the broader applicability of GUIDES beyond workflow scheduling.

Future work will extend SWAGES to multi-cloud environments with heterogeneous pricing models, capacity limits, and resource-usage constraints across distributed cloud providers. We will also evaluate GUIDES with larger policy architectures and diverse state–action representations, as well as on other large-scale reinforcement learning and combinatorial decision-making problems.

Ethics Statement

This work focuses on dynamic workflow scheduling in cloud computing environments. The work does not involve human subjects, personally identifiable information, or sensitive data. The experiments are conducted entirely using synthetic and publicly available benchmark datasets. The proposed methods are intended to improve resource utilization and reduce computational cost in cloud systems, which may contribute to more efficient use of computing resources. We do not identify significant ethical concerns arising from the current study.

Reproducibility Statement

To ensure reproducibility, we provide (1) a complete description of the problem formulation, algorithm design, and network architecture in Sections 3 and 4 and Appendices B, C, D, and F; (2) detailed experimental settings, including datasets, baselines, and hyperparameters, in Section 5 and Appendices E and G; and (3) complete pseudocode for the training algorithm in Appendix D. All CADWS experiments use the publicly available simulator at `https://github.com/yashenCS/GATES`, supporting fair and reproducible comparisons. The source code for SWAGES is provided in the supplementary material.

LLM Usage Statement

During the preparation of this manuscript, we used large language models solely to assist with language refinement, including polishing, grammar correction, and rephrasing. All LLM-assisted text was carefully reviewed and revised by the authors to ensure its accuracy and originality. The LLM was not used to produce experimental results, design algorithms or neural network architectures, or develop any of the core technical contributions of this work.

References

Vahid Arabnejad, Kris Bubendorfer, and Bryan Ng. Budget and deadline aware e-science workflow scheduling in clouds. *IEEE Transactions on Parallel and Distributed Systems*, 30(1):29–44, 2018.

Ignacio Bermudez, Stefano Traverso, Marco Mellia, and Maurizio Munafo. Exploring the cloud from passive measurements: The amazon aws case. In *2013 Proceedings IEEE INFOCOM*, pp. 230–234. IEEE, 2013.

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

Krzysztof Choromanski, Mark Rowland, Vikas Sindhwani, Richard Turner, and Adrian Weller. Structured evolution with compact architectures for scalable policy optimization. In *International Conference on Machine Learning*, pp. 970–978. PMLR, 2018.

Krzysztof M Choromanski, Aldo Pacchiano, Jack Parker-Holder, Yunhao Tang, and Vikas Sindhwani. From complexity to simplicity: Adaptive es-active subspaces for blackbox optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira Da Silva, Miron Livny, et al. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, 2015.

Hamid Reza Faragardi, Mohammad Reza Saleh Sedghpour, Saber Fazliahmadi, Thomas Fahringer, and Nayereh Rasouli. Grp-heft: A budget-constrained resource provisioning scheme for workflow scheduling in iaas clouds. *IEEE Transactions on Parallel and Distributed Systems*, 31(6):1239–1254, 2019.

Chong Huang, Gaojie Chen, Pei Xiao, Yue Xiao, Zhu Han, and Jonathon A Chambers. Joint offloading and resource allocation for hybrid cloud and edge computing in sagins: A decision assisted

hybrid action space deep reinforcement learning approach. *IEEE Journal on Selected Areas in Communications*, 42(5):1029–1043, 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*, pp. 449–464. Springer, 2022.

Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in Neural Information Processing Systems*, 12, 1999.

Chien-Liang Liu, Chun-Jan Tseng, and Po-Hao Weng. Dynamic job-shop scheduling via graph attention networks and deep reinforcement learning. *IEEE Transactions on Industrial Informatics*, 20(6):8662–8672, 2024.

Guoqing Liu, Li Zhao, Feidiao Yang, Jiang Bian, Tao Qin, Nenghai Yu, and Tie-Yan Liu. Trust region evolution strategies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4352–4359, 2019.

Fu Luo, Xi Lin, Yaoxin Wu, Zhenkun Wang, Tong Xialiang, Mingxuan Yuan, and Qingfu Zhang. Boosting neural combinatorial optimization for large-scale vehicle routing problems. In *International Conference on Learning Representations*, volume 2025, pp. 14754–14780, 2025.

Yining Ma, Zhiguang Cao, and Yeow Meng Chee. Learning to search feasible and infeasible regions of routing problems with flexible neural k-opt. *Advances in Neural Information Processing Systems*, 36:49555–49578, 2023.

Niru Maheswaranathan, Luke Metz, George Tucker, Dami Choi, and Jascha Sohl-Dickstein. Guided evolutionary strategies: Augmenting random search with surrogate gradients. In *International Conference on Machine Learning*, pp. 4264–4273. PMLR, 2019.

Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search of static linear policies is competitive for reinforcement learning. *Advances in Neural Information Processing Systems*, 31, 2018.

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.

Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 1(2):127–190, 1999.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

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

Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, 2010.

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*, pp. 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*, 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*, 2026.

Wen Song, Xinyang Chen, Qiqiang Li, and Zhiguang Cao. Flexible job-shop scheduling via graph neural network and deep reinforcement learning. *IEEE Transactions on Industrial Informatics*, 19(2):1600–1610, 2022.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 12, 1999.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Conghui Wang, Zhiguang Cao, Yaoxin Wu, Long Teng, and Guohua Wu. Deep reinforcement learning for solving vehicle routing problems with backhauls. *IEEE Transactions on Neural Networks and Learning Systems*, 36(3):4779–4793, 2024.

Tianfu Wang, Li Shen, Qilin Fan, Tong Xu, Tongliang Liu, and Hui Xiong. Joint admission control and resource allocation of virtual network embedding via hierarchical deep reinforcement learning. *IEEE Transactions on Services Computing*, 17(3):1001–1015, 2023.

Zhi Wang, Wenhan Zhan, Hancong Duan, Geyong Min, and Hualong Huang. Deep-reinforcement-learning-based continuous workflows scheduling in heterogeneous environments. *IEEE Internet of Things Journal*, 12(10):14036–14050, 2025.

Quanwang Wu, Fuyuki Ishikawa, Qingsheng Zhu, Yunni Xia, and Junhao Wen. Deadline-constrained cost optimization approaches for workflow scheduling in clouds. *IEEE Transactions on Parallel and Distributed Systems*, 28(12):3401–3412, 2017.

Xiangjie Xiao, Cong Zhang, Wen Song, and Zhiguang Cao. RESCHED: Rethinking flexible job shop scheduling from a transformer-based architecture with simplified states. In *The Fourteenth International Conference on Learning Representations*, 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 *International Conference on Learning Representations*, volume 2025, pp. 21561–21589, 2025.

Feiyang Ye, Yueming Lyu, Xuehao Wang, Masashi Sugiyama, Yu Zhang, and Ivor Tsang. Sharpness-aware black-box optimization. In *International Conference on Learning Representations*, volume 2025, pp. 60022–60052, 2025.

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.

APPENDIX CONTENTS

A Related Work **14**

A.1 Related Work on Learning-based Methods 14

A.2 Related Work on Evolution Strategies for Policy Optimization 15

B Detailed Cost-Aware Dynamic Workflow Scheduling Problem Setting and Formulation **15**

B.1 Environment Settings 15

B.2 Dynamic Cloud Workflows 16

B.3 VM Rental Cost and SLA Penalty 16

B.4 MDP Formulation 17

B.5 Raw State Features 17

C The details of System-Wide Graph Attention Network **19**

C.1 Minimal Workflow DAG Node Embedding Extraction 19

C.2 Global Embedding Extraction 20

C.3 Ready Task and VM Embedding Extraction 21

C.4 Task–VM Fusion and Scoring 21

D Detailed Gradient Guided ES Design **22**

E Detailed Experiments and Analysis on Robotics Benchmarks **23**

E.1 Robotics Experimental Setup 23

E.2 Hyperparameter Sensitivity Analysis 24

E.3 Main Results on Robotics Benchmarks 25

E.4 Performance and Stability 26

F Detailed Workflow and VM Configuration **27**

F.1 Workflow Directed Acyclic Graph 27

F.2 VM Configuration 28

G Hyperparameter Sensitivity Analysis on CADWS **28**

H Detailed Ablation Studies **29**

H.1 Network Ablation Studies 29

H.2 Algorithm Ablation Studies 30

A Related Work

A.1 Related Work on Learning-based Methods

Learning-based methods have been increasingly applied to combinatorial optimization problems, including Vehicle Routing Problems (VRP) (Ma et al., 2023; Wang et al., 2024), Resource Allocation Problems (RAP) (Huang et al., 2024; Wang et al., 2023), and Job Scheduling Problems (JSP) (Liu et al., 2024; Song et al., 2022; Zhang et al., 2020). Since these problems differ substantially in their state structures, decision spaces, and operational constraints, the policy architecture must be designed according to the characteristics of each problem.

For VRP, (Ma et al., 2023) proposed a learning-to-search policy that iteratively improves routing solutions by learning flexible k-opt exchanges through a recurrent dual-stream decoder to solve Vehicle Routing Problems. They further introduced feasibility-related features and reward shaping to guide the policy to explore both feasible and infeasible solution regions. (Wang et al., 2024) formulated route construction as a Markov decision process and developed an encoder–decoder policy that sequentially selects customers to construct vehicle routes. A two-stage attention encoder was employed to capture the relationships between the depot and different types of customers

For RAP, (Huang et al., 2024) considered joint task offloading and resource allocation in hybrid cloud and edge computing systems. They modeled user tasks as DAGs and developed a multi-agent DRL framework in which hybrid discrete and continuous actions are learned to determine offloading and resource-allocation decisions. (Wang et al., 2023) addressed virtual network embedding through hierarchical reinforcement learning, decomposing the decision process into an upper-level policy for admission control and a lower-level policy for physical resource allocation. A feature-aware graph neural network was used to represent virtual requests and physical network resources, while the hierarchical policies were optimized based on PPO.

For JSP, (Liu et al., 2024) represented dynamic job-shop scheduling states using disjunctive graphs and employed a graph attention network to learn operation embeddings under stochastic job arrivals and machine breakdowns. The learned embeddings were then provided to an actor–critic policy trained by PPO to determine the scheduling of job operations on machines. (Song et al., 2022) modeled flexible job-shop scheduling using a heterogeneous graph containing both operations and machines, and employed a heterogeneous graph neural network to capture their complex relationships. Instead of treating operation selection and machine selection separately, they combined them into a composite action and trained the DRL policy to directly select operation–machine pairs.

Motivated by the promising performance of learning-based methods in other combinatorial optimization problems, we further consider their application to Cost-Aware Dynamic Workflow Scheduling (CADWS). Compared with the problems discussed above, CADWS exhibits several distinctive characteristics that make policy learning particularly challenging. Workflows are represented as DAGs with complex task dependencies, workflows arrive dynamically during system operation, and each workflow may contain a large number of tasks.

Although existing learning-based methods have achieved promising results in CADWS, their policy representations are still largely based on local scheduling information. For example, early studies employed feed-forward networks (FFNs) to learn scheduling policies from task and resource features (Huang et al., 2022), while later approaches introduced self-attention mechanisms to enhance the representation of scheduling states (Shen et al., 2024). Graph neural networks (GNNs) were further adopted to capture structural dependencies within individual workflow DAGs (Shen et al., 2025; Yang et al., 2025), and Mixture-of-Experts (MoE) architectures were used to improve policy adaptability across different workflow patterns (Shen et al., 2026).

Building on the above discussion, our work extends the existing literature on CADWS by moving beyond local policy representations toward a more global view of the scheduling environment. Specifically, the proposed policy jointly captures information from all active workflows, models the competition among ready tasks, and explicitly represents task–VM compatibility, enabling scheduling decisions to be made from a broader system-level perspective.

A.2 Related Work on Evolution Strategies for Policy Optimization

Reinforcement learning (RL) methods have achieved great performance in policy optimization by learning policies directly from environment interactions. In particular, gradient-based methods such as policy-gradient and actor–critic algorithms can efficiently exploit local gradient information and have been successfully applied to a wide range of sequential decision-making problems (Sutton et al., 1999; Konda & Tsitsiklis, 1999). However, their performance often depends on the quality of gradient estimation and can be sensitive to noisy rewards. These limitations have motivated the exploration of gradient-free alternatives for policy optimization.

Evolution Strategies (ES) provide such an alternative by directly optimizing policy parameters according to their evaluated returns, without requiring explicit policy-gradient estimation through the environment. (Salimans et al., 2017) demonstrated that ES can achieve competitive performance on MuJoCo continuous-control tasks and Atari games, while offering a simple optimization procedure and strong parallel scalability. Related parameter-space search methods, such as Augmented Random Search (ARS) (Mania et al., 2018), further showed that direct perturbation of policy parameters can be effective for continuous-control tasks. Compared with conventional gradient-based RL, ES is particularly attractive when the optimization objective is noisy, delayed, or difficult to differentiate, since it only requires fitness evaluations of candidate policies.

Despite these advantages, standard ES typically relies on isotropic random perturbations in a high-dimensional policy parameter space, which can lead to inefficient exploration. Subsequent studies therefore focused on improving the quality of search directions. Structured Evolution Strategies (Choromanski et al., 2018) employ structured perturbations to reduce estimation variance, while Trust Region Evolution Strategies (Liu et al., 2019) improve sample utilization through surrogate optimization. ASEBO (Choromanski et al., 2019) adaptively identifies active subspaces and concentrates perturbations on promising directions. More closely related to our work, Guided ES (Maheswaranathan et al., 2019) incorporates surrogate gradients into the perturbation distribution so that ES searches preferentially within a low-dimensional subspace containing informative directions. However, Guided ES assumes that suitable surrogate gradients are already available and does not specify how such directions should be obtained during policy optimization. Our method addresses this limitation by learning guiding directions directly from accumulated population experience through replay-based twin critics and multi-batch policy-gradient estimation, and then performing fitness-based ES optimization within the resulting dynamically updated subspace.

B Detailed Cost-Aware Dynamic Workflow Scheduling Problem Setting and Formulation

B.1 Environment Settings

Cost-aware dynamic workflow scheduling (CADWS) schedules tasks from dynamically arriving workflows on heterogeneous virtual machines (VMs). At each decision step, the scheduler assigns one ready task to one eligible VM. The objective is to minimize the total VM rental cost and SLA violation penalties. According to the existing works (Huang et al., 2022; Shen et al., 2024; 2025), the following scheduling assumptions and constraints are widely considered:

1. The complete DAG structure and task information of a workflow become available when the workflow arrives.

2. Each ready task can be assigned to an eligible heterogeneous VM. The eligible choices can include an existing rented VM or a newly provisioned VM, and the task execution time depends on its workload and the processing speed of the selected VM.

3. Each VM executes at most one task at a time. Additional tasks assigned to the same VM wait in its queue and execute in first-in-first-out (FIFO) order.

4. A task becomes ready only after all of its predecessor tasks have completed. An entry task with no predecessors is ready when its workflow arrives.

B.2 Dynamic Cloud Workflows

A workflow W_i is represented by a directed acyclic graph (DAG)

$$G_i = (O_i, E_i), \tag{20}$$

where

$$O_i = \{o_{i1}, o_{i2}, \ldots, o_{in_i}\} \tag{21}$$

is the set of n_i tasks and E_i is the set of directed precedence edges. An edge $(o_{ij}, o_{ik}) \in E_i$ means that task o_{ij} must complete before o_{ik} can become ready.

Each task o_{ij} has workload $\mathrm{WL}_{ij} \in \mathbb{R}^+$. For a VM m with processing speed PS_m, the task execution time is

$$\mathrm{ET}(o_{ij}, m) = \frac{\mathrm{WL}_{ij}}{\mathrm{PS}_m}. \tag{22}$$

Workflows arrive dynamically during system operation. The arrival time of workflow W_i is denoted by $\mathrm{AT}(W_i)$, and its complete DAG becomes available at that time. In our experiments, workflow arrivals follow the Poisson process used in previous CADWS studies (Huang et al., 2022; Shen et al., 2025), with rate parameter $\lambda = 0.01$.

At decision step t, $\mathcal{W}_t$ denotes the workflows that have arrived but are not yet complete, $\mathcal{Q}_t$ denotes all ready tasks from these workflows, and $\mathcal{M}_t$ denotes the eligible VM choices. These sets change as workflows arrive, tasks complete, and VM states change.

B.3 VM Rental Cost and SLA Penalty

CADWS has two cost components: VM rental fees and SLA violation penalties. Let $\mathcal{H} = [\tau_s, \tau_e]$ denote the time horizon covered by one scheduling episode, and let $\mathrm{Set}(\pi, \mathcal{H})$ denote the VM instances rented by policy π during this horizon.

For each rented VM instance $m \in \mathrm{Set}(\pi, \mathcal{H})$, its rental interval is

$$\mathrm{UP}(m, \pi, \mathcal{H}) = [\tau_s(m, \pi, \mathcal{H}), \tau_e(m, \pi, \mathcal{H})], \tag{23}$$

where $\tau_s(m, \pi, \mathcal{H})$ and $\tau_e(m, \pi, \mathcal{H})$ are the times at which VM m starts and stops being rented.

VMs are billed in one-hour units. The total VM rental cost is therefore

$$\mathrm{VMFee}(\pi, \mathcal{H}) = \sum_{m \in \mathrm{Set}(\pi, \mathcal{H})} \mathrm{Price}(m) \left\lceil \frac{\tau_e(m, \pi, \mathcal{H}) - \tau_s(m, \pi, \mathcal{H})}{3600} \right\rceil, \tag{24}$$

where $\mathrm{Price}(m)$ is the hourly rental price of VM m. A partially used billing hour is charged as a full hour. Accordingly, the billing interval used by the state features is $\Delta_{\mathrm{pay}} = 3600$ seconds.

Each workflow W_i has an SLA deadline

$$\mathrm{DL}(W_i) = \mathrm{AT}(W_i) + \gamma_{\mathrm{dl}} \, \mathrm{MinMakespan}(W_i), \tag{25}$$

where $\mathrm{MinMakespan}(W_i)$ is its theoretical minimum execution time when the fastest VM resources are used without resource-induced waiting. The deadline coefficient γ_{dl} controls deadline tightness: a smaller value gives a tighter deadline. The subscript distinguishes γ_{dl} from the critic discount factor γ_Q used by GUIDES.

Let $\mathrm{FT}(W_i, \pi)$ denote the completion time of workflow W_i under policy π. If the workflow finishes after its deadline, its SLA penalty is

$$\mathrm{SLAPenalty}(W_i, \pi) = \beta_{\mathrm{sla}} \max\{0, \mathrm{FT}(W_i, \pi) - \mathrm{DL}(W_i)\}, \tag{26}$$

where $\beta_{\mathrm{sla}} > 0$ is the penalty rate per unit of deadline violation. In all CADWS experiments, we set $\beta_{\mathrm{sla}} = 0.24/hour$, following the configuration used in the GATES simulator (Shen et al., 2025).

The total scheduling cost is

$$\mathrm{TotalCost}(\pi, \mathcal{H}) = \mathrm{VMFee}(\pi, \mathcal{H}) + \sum_{W_i \in \mathcal{W}(\mathcal{H})} \mathrm{SLAPenalty}(W_i, \pi), \tag{27}$$

where $\mathcal{W}(\mathcal{H})$ is the set of workflows considered during the episode. The CADWS objective is

$$\pi^* = \arg\min_{\pi} \mathbb{E}_{\pi}[\text{TotalCost}(\pi, \mathcal{H})]\,. \tag{28}$$

Using faster or additional VMs can reduce SLA penalties but increase rental cost, while cheaper VM choices can reduce rental cost but increase the risk of deadline violations. The scheduling policy must therefore balance these two costs.

For the stepwise MDP formulation, let C_t^{vm} and C_t^{sla} denote the cumulative VM rental cost and SLA penalty after decision step t. The incremental costs generated by the transition from t to $t+1$ are

$$\Delta C_t^{\text{vm}} = C_{t+1}^{\text{vm}} - C_t^{\text{vm}}, \qquad \Delta C_t^{\text{sla}} = C_{t+1}^{\text{sla}} - C_t^{\text{sla}}. \tag{29}$$

The cumulative VM cost increases when a new billing unit is charged, while the cumulative SLA cost increases when a workflow completes after its deadline. For an episode with T decisions and zero initial accumulated cost,

$$\sum_{t=0}^{T-1} \left(\Delta C_t^{\text{vm}} + \Delta C_t^{\text{sla}}\right) = \text{TotalCost}(\pi, \mathcal{H}). \tag{30}$$

Thus, the stepwise costs used by the MDP exactly recover the CADWS objective.

B.4 MDP Formulation

We formulate CADWS as an episodic MDP $(\mathcal{S}, \mathcal{A}, P, r)$ with state, action, transition, and reward defined as follows.

State. At decision step t, state $s_t \in \mathcal{S}$ describes the current CADWS system. It contains five groups of information: global system information, workflow-DAG information, ready-task information, VM information, and task–VM compatibility information. The complete raw features are listed in Appendix B.5.

Action. The action is a possible task–VM assignment

$$a_t = (o_t, m_t), \qquad o_t \in \mathcal{Q}_t, \quad m_t \in \mathcal{M}_t. \tag{31}$$

The action set is state-dependent because the ready-task set $\mathcal{Q}_t$ and eligible VM choices $\mathcal{M}_t$ change during scheduling.

Transition. Executing a_t assigns task o_t to VM m_t. The environment then updates task completion, workflow progress, VM queues and availability, and newly arrived workflows, producing the next state s_{t+1} according to $P(s_{t+1} \mid s_t, a_t)$.

Reward. The immediate reward is the negative cost generated by the transition:

$$r_t = -\left(\Delta C_t^{\text{vm}} + \Delta C_t^{\text{sla}}\right). \tag{32}$$

Using equation 29, this is equivalently

$$r_t = -\left[\left(C_{t+1}^{\text{vm}} - C_t^{\text{vm}}\right) + \left(C_{t+1}^{\text{sla}} - C_t^{\text{sla}}\right)\right]. \tag{33}$$

Together with equation 30, the undiscounted episodic return is the negative total CADWS cost. Maximizing the return is therefore equivalent to minimizing equation 28.

B.5 Raw State Features

The state representation is constructed from five raw feature groups: workflow-DAG, ready-task, global, VM, and task–VM compatibility features. All these features follow the existing workflow simulator (Shen et al., 2024; 2025; Yang et al., 2025). Table 3 gives the complete feature definitions used by the scheduling policy.

Table 3: The detailed raw state features in CADWS.

Feature group	Feature	Description
Workflow-DAG	**Predecessor**	the number of predecessor tasks of each task in its corresponding MWDAG.
	Successor	the number of successor tasks of each task in its corresponding MWDAG.
	Processing time	the average duration required to execute a task on all types of VMs.
	Task size	the data size associated with the task.
	Task deadline	the task deadline d_i^{sub} is calculated using the method in (Wu et al., 2017).
	Task status	0 for a blocked task with at least one incomplete predecessor, and 1 for a ready task with no predecessors or with all predecessors completed.
Ready-task	**Direct successors**	reflects the number of tasks immediately affected by completing o_i and is defined as $\log(1+n_i^{\mathrm{child}})$, where n_i^{child} is the number of direct child tasks.
	Current slack	measures the remaining time before the task sub-deadline relative to the billing interval and is defined as $\mathrm{clip}\big((d_i^{\mathrm{sub}}-t)/\Delta_{\mathrm{pay}}, -10, 10\big)$, in which Δ_{pay} denotes the billing interval and t the current time.
Global	**System size**	the number of active workflows and the number of unfinished tasks.
	Task-state ratios	the proportions of unfinished tasks that are ready or blocked.
	Average completion ratio	the mean completion ratio across active workflows.
	Deadline pressure	the minimum and mean remaining time before the sub-deadlines of ready tasks.
	Workload	the total remaining computational workload and the proportion of that workload belonging to ready tasks.
VM	**CPU capacity**	the processing capacity of the VM.
	VM price	the rental price per unit time of the VM.
	New VM indicator	whether the candidate action reuses an existing VM or provisions a new one.
	Availability delay	the remaining time before the VM can start executing another task.
	Remaining paid time	the unused time in the current billing interval of an existing VM.
	Pending task count	the number of tasks waiting in the VM's queue.
	Queued processing time	the total processing time required by tasks waiting in the VM's queue.
Compatibility	**Execution time**	the estimated time required for the candidate VM to execute the task.
	Waiting time	the estimated time the task must wait before execution begins on the candidate VM.
	Post-completion slack	the difference between the task sub-deadline and its estimated completion time on the candidate VM. A nonnegative value indicates that the VM is expected to meet the task sub-deadline.
	Incremental VM cost	the additional VM rental fee incurred by using the candidate VM to execute the task.

C THE DETAILS OF SYSTEM-WIDE GRAPH ATTENTION NETWORK

C.1 MINIMAL WORKFLOW DAG NODE EMBEDDING EXTRACTION

At decision step t, the agent considers only workflows that have already arrived but have not yet been completed. We define the set of such active workflows as

$$\mathcal{W}_t = \{W_i \mid \mathrm{AT}(W_i) \le t,\ W_i \text{ is not completed at } t\}\,. \tag{34}$$

Once a workflow arrives, its complete DAG $G_i = (O_i, E_i)$ becomes available and the workflow is added to $\mathcal{W}_t$. It remains in this set until all of its tasks have been completed.

For each workflow $W_i \in \mathcal{W}_t$, let $O_{i,t}^{\mathrm{fin}}$ denote the set of tasks that have been completed by decision step t, and let $O_{i,t}^{\mathrm{sch}}$ denote the set of tasks that have already been scheduled but have not yet completed. Since neither completed nor scheduled tasks require further scheduling decisions, both are excluded from the graph representation. Accordingly, the minimal workflow DAG (MWDAG) is constructed as

$$O_{i,t}^{\min} = O_i \setminus \left(O_{i,t}^{\mathrm{fin}} \cup O_{i,t}^{\mathrm{sch}}\right), \tag{35}$$

with the remaining precedence edges defined as

$$E_{i,t}^{\min} = \left\{(u,v) \in E_i \mid u, v \in O_{i,t}^{\min}\right\}. \tag{36}$$

The resulting MWDAG is therefore

$$G_{i,t}^{\min} = \left(O_{i,t}^{\min}, E_{i,t}^{\min}\right). \tag{37}$$

For each retained task $o_{in} \in O_{i,t}^{\min}$, its input feature vector is denoted by $\mathbf{x}_{i,n} \in \mathbb{R}^{d_{\mathrm{dag}}}$, where d_{dag} denotes the unified node-feature dimension used for workflow DAG encoding. The MWDAGs of all workflows in $\mathcal{W}_t$ are encoded separately using a shared GAT encoder with the same learnable parameters.

To explicitly model the directional dependencies between tasks, each precedence edge is considered in two message-passing directions. Let $d \in \{\mathrm{f}, \mathrm{b}\}$ denote the direction, where f represents the forward branch and b represents the backward branch. For a task o_{in}, we define the corresponding predecessor and successor sets as

$$\mathcal{N}_{i,n}^{\mathrm{f}} = \left\{o_{ij} \mid (o_{ij}, o_{in}) \in E_{i,t}^{\min}\right\}, \tag{38}$$

and

$$\mathcal{N}_{i,n}^{\mathrm{b}} = \left\{o_{ij} \mid (o_{in}, o_{ij}) \in E_{i,t}^{\min}\right\}, \tag{39}$$

respectively. The parent-to-child direction allows a task to aggregate information from its predecessors, whereas the child-to-parent direction propagates information from its successors.

For either direction d and attention head q in GAT layer ℓ, the normalized attention coefficient from task $o_{ij} \in \mathcal{N}_{i,n}^{d}$ to task o_{in} is

$$\alpha_{jn}^{(\ell,d,q)} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}^{(\ell,d,q)\top}\left[\mathbf{W}^{(\ell,d,q)}\mathbf{h}_{i,n}^{(\ell-1)} \,\|\, \mathbf{W}^{(\ell,d,q)}\mathbf{h}_{i,j}^{(\ell-1)}\right]\right)\right)}{\sum_{o_{ik} \in \mathcal{N}_{i,n}^{d}} \exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}^{(\ell,d,q)\top}\left[\mathbf{W}^{(\ell,d,q)}\mathbf{h}_{i,n}^{(\ell-1)} \,\|\, \mathbf{W}^{(\ell,d,q)}\mathbf{h}_{i,k}^{(\ell-1)}\right]\right)\right)}. \tag{40}$$

The corresponding directional representation is obtained by concatenating the attention-weighted outputs of all heads:

$$\mathbf{h}_{i,n}^{(\ell,d)} = \Big\|_{q=1}^{H_\ell} \left(\sum_{o_{ij} \in \mathcal{N}_{i,n}^{d}} \alpha_{jn}^{(\ell,d,q)} \mathbf{W}^{(\ell,d,q)} \mathbf{h}_{i,j}^{(\ell-1)}\right). \tag{41}$$

Accordingly, $\mathcal{N}_{i,n}^{\mathrm{f}}$ contains the predecessors of the target task, whereas $\mathcal{N}_{i,n}^{\mathrm{b}}$ contains its successors. The two directions use independent learnable parameters.

The task's own information is independently transformed through a linear layer:

$$\mathbf{h}_{i,n}^{(\ell,\mathrm{self})} = \mathbf{W}_{\mathrm{self}}^{(\ell)} \mathbf{h}_{i,n}^{(\ell-1)} + \mathbf{b}_{\mathrm{self}}^{(\ell)}. \tag{42}$$

Accordingly, self-loops in the two GAT branches are disabled to avoid duplicating the task's own information.

The outputs of the three paths are combined using equal weights:

$$\mathbf{h}_{i,n}^{(\ell)} = \mathrm{ELU}\left(\frac{\mathbf{h}_{i,n}^{(\ell,\mathrm{f})} + \mathbf{h}_{i,n}^{(\ell,\mathrm{b})} + \mathbf{h}_{i,n}^{(\ell,\mathrm{self})}}{3}\right). \tag{43}$$

The encoder uses two such GAT layers. The first layer maps the node features to an intermediate representation, and the second produces the final graph-aware node embedding:

$$\mathbf{h}_{i,n}^{(1)} \in \mathbb{R}^{d_1}, \qquad \mathbf{h}_{i,n}^{(2)} \in \mathbb{R}^{d_h}. \tag{44}$$

The final node-representation matrix $\mathbf{H}_{i,t}^{(L)}$ contains the graph-aware embeddings of all remaining tasks. The workflow-level representation is obtained by mean pooling:

$$\mathbf{h}_{i,t}^{\mathrm{wf}} = \mathrm{MeanPool}\left(\mathbf{H}_{i,t}^{(L)}\right). \tag{45}$$

C.2 Global Embedding Extraction

Section C.1 defines the embedding $h_{i,t}^{\mathrm{wf}}$ for each active workflow. These embeddings capture the structural information of the remaining workflow DAGs, but they do not explicitly reflect the relative importance of each workflow under the current system workload, deadline pressure, and resource contention. To incorporate this system-level context, we employ a cross-attention mechanism to aggregate the workflow embeddings according to the current global state. Specifically, the global system representation is used as the query, while the workflow embeddings are used as the keys and values.

Let $x_t^{\mathrm{global}} \in \mathbb{R}^{d_{\mathrm{global}}}$ denote the global feature vector at decision step t, and let $f_{\mathrm{global}} : \mathbb{R}^{d_{\mathrm{global}}} \to \mathbb{R}^{d_h}$ denote a learnable multilayer perceptron. The global query vector q_t is defined as

$$q_t = f_{\mathrm{global}}(x_t^{\mathrm{global}}) \in \mathbb{R}^{d_h}. \tag{46}$$

Let

$$H_t^{\mathrm{wf}} = \left[h_{1,t}^{\mathrm{wf}}, \ldots, h_{|\mathcal{W}_t|,t}^{\mathrm{wf}}\right]^\top \tag{47}$$

stack the embeddings of all active workflows. For the cross-attention mechanism, the workflow matrix is used as both the key and the value:

$$K_t = V_t = H_t^{\mathrm{wf}}. \tag{48}$$

Let H_A denote the number of cross-attention heads, let $r \in \{1, \ldots, H_A\}$ index a head, and let d_k denote the dimension of each head, with $d_h = H_A d_k$. For head r, let W_r^Q, W_r^K, and W_r^V denote the learnable query, key, and value projections, respectively. The operator $\mathrm{softmax}$ normalizes the attention scores over the active workflows. The head output is

$$\mathrm{head}_r = \mathrm{softmax}\left(\frac{(q_t W_r^Q)(K_t W_r^K)^\top}{\sqrt{d_k}}\right) V_t W_r^V. \tag{49}$$

Let W^O denote the learnable output projection. The head outputs are concatenated and projected to obtain the workflow context vector c_t^{wf}:

$$c_t^{\mathrm{wf}} = \left(\Big\Vert_{r=1}^{H_A} \mathrm{head}_r\right) W^O. \tag{50}$$

Let $\mathrm{FFN}_{\mathrm{global}}$ denote the learnable position-wise feed-forward network. The attention output is processed by two residual blocks. The first combines the query with the workflow context:

$$g_t^{(1)} = \mathrm{LayerNorm}(q_t + c_t^{\mathrm{wf}}) \in \mathbb{R}^{d_h}. \tag{51}$$

The second applies the feed-forward network and produces the final global embedding:

$$h_t^{\mathrm{global}} = \mathrm{LayerNorm}\left(g_t^{(1)} + \mathrm{FFN}_{\mathrm{global}}(g_t^{(1)})\right) \in \mathbb{R}^{d_h}. \tag{52}$$

Because the attention weights depend on q_t, the contribution of each active workflow changes with the current workload, deadline pressure, and resource state.

C.3 Ready Task and VM Embedding Extraction

Since the ready task set and the VM candidate set change dynamically during scheduling, we denote the corresponding sets at decision step t by $\mathcal{Q}_t$ and $\mathcal{M}_t$, respectively. We encode these two sets using Transformer (Vaswani et al., 2017) encoders with the same network architecture but separate learnable parameters.

For a unified formulation, let $s \in \{\text{task}, \text{vm}\}$ index the two encoders. The number of elements and the input feature dimension of each encoder are defined as

$$N_t^s = \begin{cases} |\mathcal{Q}_t|, & s = \text{task}, \\ |\mathcal{M}_t|, & s = \text{vm}, \end{cases} \qquad d_s = \begin{cases} d_{\text{task}}, & s = \text{task}, \\ d_{\text{vm}}, & s = \text{vm}. \end{cases} \tag{53}$$

Let $X_t^s \in \mathbb{R}^{N_t^s \times d_s}$ denote the corresponding input feature matrix. Each feature vector is first mapped to a common d_h-dimensional latent space through a learnable projection $f_s : \mathbb{R}^{d_s} \to \mathbb{R}^{d_h}$:

$$H_t^{s,0} = f_s(X_t^s) \in \mathbb{R}^{N_t^s \times d_h}. \tag{54}$$

Let H_s denote the number of attention heads in encoder s, and let d_k^s denote the dimension of each head, such that $d_h = H_s d_k^s$. For the rth attention head, $r \in \{1, \ldots, H_s\}$, the learnable query, key, and value projection matrices are denoted by $W_{r,s}^Q$, $W_{r,s}^K$, and $W_{r,s}^V$, respectively. Given an embedding matrix H, the multi-head self-attention operation is defined as

$$\text{MHA}_s(H) = \left(\Big\Vert_{r=1}^{H_s} \left[\text{softmax}\left(\frac{(HW_{r,s}^Q)(HW_{r,s}^K)^\top}{\sqrt{d_k^s}} \right) HW_{r,s}^V \right] \right) W_s^O, \tag{55}$$

where W_s^O is the learnable output projection matrix and $\Vert$ denotes concatenation across attention heads.

The self-attention output is then followed by a feed-forward network FFN_s. Residual connections and layer normalization are applied after both operations:

$$\widetilde{H}_t^s = \text{LayerNorm}\Big(H_t^{s,0} + \text{MHA}_s(H_t^{s,0}) \Big), \tag{56}$$

$$H_t^s = \text{LayerNorm}\Big(\widetilde{H}_t^s + \text{FFN}_s(\widetilde{H}_t^s) \Big). \tag{57}$$

For $s = \text{task}$, the ith row $h_{i,t}^{\text{task}}$ of H_t^{task} represents ready task o_i relative to all other currently ready tasks. For $s = \text{vm}$, the jth row $h_{j,t}^{\text{vm}}$ of H_t^{vm} represents VM candidate m_j relative to other VMs.

C.4 Task–VM Fusion and Scoring

Let $\mathcal{A}_t$ denote the possible action set at decision step t:

$$\mathcal{A}_t = \mathcal{Q}_t \times \mathcal{M}_t = \{(o_i, m_j) \mid o_i \in \mathcal{Q}_t,\ m_j \in \mathcal{M}_t\}. \tag{58}$$

For each assignment $(o_i, m_j) \in \mathcal{A}_t$, let $x_{i,j,t}^{\text{comp}} \in \mathbb{R}^{d_{\text{comp}}}$ denote its raw compatibility-feature vector. Let $f_{\text{comp}} : \mathbb{R}^{d_{\text{comp}}} \to \mathbb{R}^{d_h}$ denote a learnable multilayer perceptron. The compatibility embedding is

$$h_{i,j,t}^{\text{comp}} = f_{\text{comp}}(x_{i,j,t}^{\text{comp}}) \in \mathbb{R}^{d_h}. \tag{59}$$

The representation of candidate assignment (o_i, m_j) concatenates the contextual task embedding, the global embedding, the VM embedding, and the pairwise compatibility embedding:

$$z_{i,j,t} = h_{i,t}^{\text{task}} \Vert h_t^{\text{global}} \Vert h_{j,t}^{\text{vm}} \Vert h_{i,j,t}^{\text{comp}} \in \mathbb{R}^{4d_h}. \tag{60}$$

Let $f_{\text{score}} : \mathbb{R}^{4d_h} \to \mathbb{R}$ denote the scoring multilayer perceptron (MLP). It assigns the score to every possible assignment.

$$s_{i,j,t} = f_{\text{score}}(z_{i,j,t}) \tag{61}$$

Let s_t denote the current scheduling state and let θ denote the policy parameters. Using (o_p, m_q) as the summation variable over $\mathcal{A}_t$, the scores define the policy distribution

$$\pi_\theta((o_i, m_j) \mid s_t) = \frac{\exp(s_{i,j,t})}{\sum_{(o_p, m_q) \in \mathcal{A}_t} \exp(s_{p,q,t})}. \tag{62}$$

Actions are sampled from this distribution during training, while at evaluation time, the scheduler selects the task–VM assignment with the highest score.

D Detailed Gradient Guided ES Design

This section details the training procedure of GUIDES, as summarized in Algorithm 1. GUIDES reuses transitions collected during population rollouts to train critics and construct a low-dimensional search subspace. The critics guide perturbation sampling, while population fitness determines how the sampled directions are combined to update the center policy.

Initialization and First-Generation Exploration. GUIDES maintains a center policy π_{θ_g}, two online critics Q_{ϕ_1} and Q_{ϕ_2}, their corresponding target critics, and a replay buffer $\mathcal{D}$. At initialization, the target critics copy the parameters of the online critics, and the replay buffer is empty. Since no rollout experience is available for critic learning, the first generation uses full-space Gaussian perturbations, as specified by the $g = 1$ case in equation 18. These initial rollouts provide both population fitness values and transition data for subsequent critic training.

Critic-Guided Search Subspace. From the second generation onward, GUIDES constructs a search subspace using the current center policy, the learned critics, and replayed experience. Specifically, it independently samples K guidance batches $\{\mathcal{B}_k\}_{k=1}^{K}$, each containing B_G transitions. For each transition $b \in \mathcal{B}_k$, let $q_{k,b} = \min_{j\in\{1,2\}} Q_{\phi_j}(s_b, a_b)$. The centered critic weight is

$$w_{k,b} = q_{k,b} - \frac{1}{B_G} \sum_{c\in\mathcal{B}_k} q_{k,c}. \tag{63}$$

These weights are used in the surrogate objective defined in equation 16. Differentiating this objective at θ_g and normalizing the resulting gradients produces K guidance directions. The directions are orthogonalized according to equation 17 to obtain $U_g \in \mathbb{R}^{d\times K}$. Population perturbations are then sampled using the $g > 1$ case in equation 18. Because the guidance is estimated from replayed transitions generated by multiple population policies, it provides surrogate search directions rather than exact on-policy gradients.

Population Rollout and Experience Collection. Given a sampled perturbation $\epsilon_{g,i}$, the parameters of the i-th population member are

$$\theta_{g,i} = \theta_g + \sigma\epsilon_{g,i}, \tag{64}$$

where σ controls the perturbation scale. Each candidate policy is evaluated over a complete episode. For CADWS, its fitness is the negative undiscounted total cost:

$$F_{g,i} = -\sum_{t=0}^{T_{g,i}-1} \left(\Delta C^{\mathrm{vm}}_{g,i,t} + \Delta C^{\mathrm{sla}}_{g,i,t}\right), \tag{65}$$

where $T_{g,i}$ denotes the episode length, and $\Delta C^{\mathrm{vm}}_{g,i,t}$ and $\Delta C^{\mathrm{sla}}_{g,i,t}$ are the incremental VM rental cost and SLA penalty at step t, respectively. The rollout transitions $(s_t, a_t, r_t, s_{t+1}, a_{t+1}, \delta_t)$ are stored in $\mathcal{D}$. Whenever the buffer exceeds its capacity C, the oldest transitions are discarded. Each rollout therefore provides a fitness value for ES optimization and transition data for critic learning.

Replay-Based Twin-Critic Learning. After population evaluation, GUIDES performs N_Q critic updates using mini-batches of B transitions sampled from $\mathcal{D}$. For each batch, TD targets are computed using the target critics according to equation 14. Both online critics are optimized using the loss in equation 15, with gradient clipping applied to limit large updates. After each critic update, the target parameters are softly updated as

$$\bar{\phi}_j \leftarrow (1-\tau)\bar{\phi}_j + \tau\phi_j, \qquad j \in \{1,2\}, \tag{66}$$

where τ controls the target-update rate. The discount factor γ_Q is used only for auxiliary TD learning and does not change the undiscounted fitness defined in equation 65.

Fitness-Based ES Policy Update. The population fitness values $\{F_{g,i}\}_{i=1}^{N}$ are converted into centered ranks $\{\rho_{g,i}\}_{i=1}^{N}$ to reduce sensitivity to the magnitude of returns. GUIDES then combines these ranks with the corresponding perturbations to estimate the ES search direction using equation 19. An Adam ascent step with learning rate η_{ES} updates the center policy from θ_g to θ_{g+1}. Thus, critic gradients determine the search subspace but are not directly applied as policy updates. The updated critics and replay buffer are subsequently used to construct the guidance basis for the next generation.

Algorithm 1 Training SWAGES with GUIDES. Steps marked with ▷ denote the newly proposed components of GUIDES.

Require: Number of generations G, population size N, perturbation scale σ, subspace dimension K
Require: Critic updates per generation N_Q, critic batch size B, guidance batch size B_G, replay capacity C
Require: Critic discount γ_Q, target-update rate τ, ES learning rate η_{ES}
Initialize center policy π_{θ_1}
Initialize online critics Q_{ϕ_1}, Q_{ϕ_2}
Initialize target critic parameters $\bar{\phi}_j \leftarrow \phi_j$, $j \in \{1, 2\}$
Initialize empty replay buffer $\mathcal{D}$
for $g = 1, 2, \ldots, G$ **do**
  if $g = 1$ **then**
    Sample full-space perturbations $\{\epsilon_{g,i}\}_{i=1}^{N}$ using the $g = 1$ case in equation 18
  else
    for $k = 1, \ldots, K$ **do**
      ▷Sample a guidance batch $\mathcal{B}_k$ of B_G transitions from $\mathcal{D}$
      ▷Compute centered critic weights using equation 63
      ▷Compute and normalize the gradient of equation 16 at θ_g to obtain g_k
    end for
    ▷Construct U_g using equation 17
    ▷Sample guided perturbations $\{\epsilon_{g,i}\}_{i=1}^{N}$ using the $g > 1$ case in equation 18
  end if
  for $i = 1, \ldots, N$ **do**
    Construct $\theta_{g,i}$ using equation 64
    Roll out $\pi_{\theta_{g,i}}$ and compute $F_{g,i}$ using equation 65
    ▷Insert rollout transitions into $\mathcal{D}$, discarding the oldest whenever $|\mathcal{D}| > C$
  end for
  for $u = 1, \ldots, N_Q$ **do**
    ▷Sample B transitions $(s_b, a_b, r_b, s'_b, a'_b, \delta_b)$ from $\mathcal{D}$
    ▷Compute TD targets using equation 14
    ▷Update both online critics using equation 15 with gradient clipping
    ▷Update target critics using equation 66
  end for
  Convert $\{F_{g,i}\}_{i=1}^{N}$ into centered ranks $\{\rho_{g,i}\}_{i=1}^{N}$
  Compute $\widehat{g}_{\mathrm{ES},g}$ using equation 19
  Apply an Adam ascent step with $\widehat{g}_{\mathrm{ES},g}$ and learning rate η_{ES} to obtain θ_{g+1}
end for
return center policy $\pi_{\theta_{G+1}}$

E Detailed Experiments and Analysis on Robotics Benchmarks

E.1 Robotics Experimental Setup

Environment. We consider four MuJoCo-based continuous-control benchmarks: Hopper-v4, HalfCheetah-v4, Ant-v4, and Walker2d-v4. Performance is measured by episodic return, with higher values indicating better performance.

Table 4: Selected GUIDES hyperparameters for the robotics benchmarks.

Environment	Subspace Dimension	Critic LR
Hopper-v4	16	$3e-4$
HalfCheetah-v4	16	$3e-4$
Ant-v4	8	$3e-4$
Walker2d-v4	16	$1e-4$

Baselines. We compare GUIDES with CEM (Rubinstein, 1999), PEPG (Sehnke et al., 2010), ARS (Mania et al., 2018), and OpenAI-ES (Salimans et al., 2017), matching the methods reported in Table 7. All methods are trained for 5M environment steps using 10 random seeds. The policy is represented by an MLP with two hidden layers of 256 units, Tanh activations in both hidden and output layers, and a maximum episode length of 1000 steps. Performance is evaluated over three episodes. The baseline hyperparameter settings are summarized in Table 5. This setup ensures a consistent comparison of GUIDES against alternative parameter-space optimization methods across the four robotics environments.

Table 5: Hyperparameter settings of the baseline methods for the robotics benchmarks.

Method	**Hyperparameter Setting**
OpenAI-ES	Population $= 40$, $\sigma = 0.02$, 20 antithetic perturbation pairs, centered-rank fitness, Adam ($\beta_1 = 0.9, \beta_2 = 0.999$)
CEM	Population $= 256$, elite ratio $= 0.10$ (26 elites), $\sigma_{\text{init}} = 0.10$, $\sigma_{\min} = 0.01$, smoothing $= 0.70$
PEPG	Population $= 256$ (128 antithetic perturbation pairs), $\sigma_{\text{init}} = 0.10$, $\sigma_{\text{lr}} = 0.20$, $\sigma_{\text{decay}} = 0.999$, $\sigma_{\min} = 0.01$, $\sigma_{\text{max_change}} = 0.20$, LR $= 0.01$, LR decay $= 0.9999$, LR min $= 0.01$, weight decay $= 0.01$
ARS	Directions $= 8$, top directions $= 8$, noise std $= 0.03$, step size $= 0.02$, 16 trajectories per generation

Parameters. The robotics hyperparameter analysis considers subspace dimensions $K \in \{8, 16, 32\}$ and critic learning rates $\eta_c \in \{1e-4, 3e-4, 1e-3\}$. The Base configuration uses subspace dimension $K = 16$ and a critic learning rate of $\eta_c = 3e-4$ in all four environments. The detailed sensitivity results are reported in Table 6. Based on the mean episodic return, we select the environment-specific configurations summarized in Table 4.

Train and Test. The training and evaluation protocol is specified separately for each robotics environment. Each method is trained for a fixed budget of 5 million environment steps. During training, the checkpoint with the highest evaluation episodic return is retained as the best-performing policy and is used for the final test. The final comparison reports the test episodic return of the selected policy, averaged over 10 independent tests.

E.2 Hyperparameter Sensitivity Analysis

We analyze the sensitivity of GUIDES to the subspace dimension K and critic learning rate η_c on Hopper-v4, HalfCheetah-v4, Ant-v4, and Walker2d-v4. Since these environments exhibit different dynamics and control characteristics, the analysis is conducted separately for each task using mean episodic return as the primary criterion.

As shown in Table 6, the Base configuration performs best on both Hopper-v4 and HalfCheetah-v4, where $K = 16$ and $\eta_c = 3 \times 10^{-4}$ provide a good balance between guided exploration and critic adaptation. In contrast, Ant-v4 benefits from a smaller search subspace, with $K = 8$ achieving the highest mean return, while a larger critic learning rate improves performance. For Walker2d-v4, a smaller critic learning rate of $\eta_c = 1 \times 10^{-4}$ produces the best mean return, while reducing the subspace dimension also improves performance over the Base configuration.

Overall, the results indicate that the effectiveness of GUIDES depends on the interaction between the search subspace and environment dynamics. Moderate settings are sufficient for Hopper-v4 and HalfCheetah-v4, whereas Ant-v4 and Walker2d-v4 benefit from more environment-specific configurations. These results motivate using task-specific hyperparameters in the main robotics experiments.

Table 6 provides a separate panel for each environment, with entries for the mean, standard deviation, and best episodic return.

Table 6: Hyperparameter sensitivity analysis on robotics benchmarks.

(a) Hopper-v4

Configuration	Mean	STD	BEST
Base	**1132.07**	**29.55**	1167.74
$K = 8$	1098.46	67.21	1227.23
$K = 32$	1119.11	72.88	**1228.94**
$\eta_c = 1e-4$	1083.64	43.37	1165.56
$\eta_c = 1e-3$	1086.68	50.09	1158.45

(b) HalfCheetah-v4

Configuration	Mean	STD	BEST
Base	**4566.79**	**563.88**	**5365.38**
$K = 8$	4469.72	810.48	5253.80
$K = 32$	3614.78	1056.21	4921.46
$\eta_c = 1e-4$	3812.59	840.08	4980.16
$\eta_c = 1e-3$	4387.17	729.78	5220.21

(c) Ant-v4

Configuration	Mean	STD	BEST
Base	1104.32	116.17	1335.52
$\mathbf{K = 8}$	**1321.71**	306.49	**1933.99**
$K = 32$	1008.56	**36.19**	1101.93
$\eta_c = 1e-4$	1095.90	181.09	1616.68
$\eta_c = 1e-3$	1134.07	137.57	1443.89

(d) Walker2d-v4

Configuration	Mean	STD	BEST
Base	1311.55	**272.20**	1816.85
$K = 8$	1439.76	400.59	2139.01
$K = 32$	1349.66	418.12	**2269.35**
$\boldsymbol{\eta_c = 1e-4}$	**1591.57**	275.63	1966.10
$\eta_c = 1e-3$	1497.20	385.74	2140.93

E.3 Main Results on Robotics Benchmarks

We evaluate GUIDES against CEM, PEPG, ARS, and OpenAI-ES on Hopper-v4, HalfCheetah-v4, Ant-v4, and Walker2d-v4. The baselines use the configurations recommended in their respective papers. For GUIDES, hyperparameters are selected separately for each environment, with the selection details provided in Appendix E. Table 7 reports the mean and standard deviation of episodic returns over 10 runs at 5 million environment steps.

Reward. GUIDES achieves the highest mean episodic return on all four robotics benchmarks. On Hopper, GUIDES reaches a mean return of 1110.35, improving over the strongest baseline, OpenAI-ES (1053.57), while maintaining relatively stable performance across runs. The improvement becomes more pronounced on HalfCheetah, where GUIDES achieves 4457.31 compared with 3661.36 for ARS, corresponding to a 21.74% increase. Its standard deviation of 307.13 is also substantially lower than those of ARS and OpenAI-ES, indicating that the improvement in mean performance is accompanied by more consistent optimization.

Table 7: Mean (standard deviation) episodic return of each algorithm over 10 runs at 5 million environment steps. Bold values indicate the highest mean return in each environment. $(+)$, $(-)$, and $(=)$ indicate that the corresponding method performs significantly better than, significantly worse than, or shows no significant difference from each preceding method, respectively, according to the Wilcoxon rank-sum test ($p < 0.05$).

Environment	CEM	PEPG	ARS	OpenAI-ES	GUIDES (ours)
Ant-v4	954.60(19.71)	939.50(45.30) $(=)$	905.17(154.45) $(=)(=)$	1051.99(66.91) $(+)(+)(+)$	**1321.61(306.57)** $(+)(+)(+)(+)$
HalfCheetah-v4	921.12(554.72)	471.14(249.83) $(=)$	3661.36(1186.90) $(+)(+)$	3571.66(637.34) $(+)(+)(=)$	**4457.31(307.13)** $(+)(+)(=)(+)$
Hopper-v4	1038.46(13.06)	984.56(43.91) $(-)$	1041.03(21.97) $(=)(+)$	1053.57(17.82) $(=)(+)(=)$	**1110.35(34.27)** $(+)(+)(+)(+)$
Walker2d-v4	1570.66(334.04)	582.65(84.47) $(-)$	559.64(506.93) $(-)(=)$	1353.57(316.76) $(=)(+)(+)$	**1618.14(498.69)** $(=)(+)(+)(+)$

The largest relative improvement is observed on Ant. GUIDES achieves a mean return of 1321.61, compared with 1051.99 for OpenAI-ES, yielding a 25.63% improvement. This result suggests that restricting evolutionary exploration to critic-informed directions can be particularly beneficial in a higher-dimensional control problem. However, GUIDES also exhibits a larger standard deviation on Ant, indicating that the higher return is obtained with increased run-to-run variability.

On Walker2d, GUIDES achieves the highest mean return of 1618.14, slightly surpassing CEM at 1570.66 and clearly outperforming OpenAI-ES, ARS, and PEPG. Its standard deviation is relatively large, showing that the advantage of GUIDES in this environment is mainly reflected in its higher average performance rather than improved stability. Overall, these results show that GUIDES consistently improves the mean return across environments, although the degree of improvement and training variability differ across tasks.

Cross-Environment Performance. GUIDES achieves the highest mean episodic return across all four benchmarks, demonstrating consistent effectiveness across different locomotion tasks. While the improvements on Hopper and Walker2d are relatively modest, larger gains are observed on HalfCheetah and Ant. These results suggest that the critic-guided subspace search can effectively exploit rollout information to provide more informative search directions, particularly in environments with more complex policy optimization landscapes.

E.4 Performance and Stability

Performance Analysis. Figure 4 compares the optimization performance of OpenAI-ES, CEM, PEPG, ARS, and GUIDES over five million environment steps. GUIDES consistently reaches the highest final return across all four environments, while the improvement patterns vary across tasks. On Ant, GUIDES continues to improve throughout training and establishes a clear performance gap over the other methods, whereas the competing approaches plateau at substantially lower returns. A similar advantage is observed on HalfCheetah, where GUIDES maintains sustained improvement after ARS begins to plateau and eventually reaches a considerably higher return. On Hopper, most methods converge rapidly to similar performance levels, but GUIDES still achieves the highest final return. Walker2d presents a more competitive case, with CEM approaching GUIDES during later training, although GUIDES ultimately attains the highest performance. These results indicate that the critic-guided subspace search is particularly effective on Ant and HalfCheetah, where continued progress beyond the plateaus of conventional evolutionary methods leads to substantially better solutions.

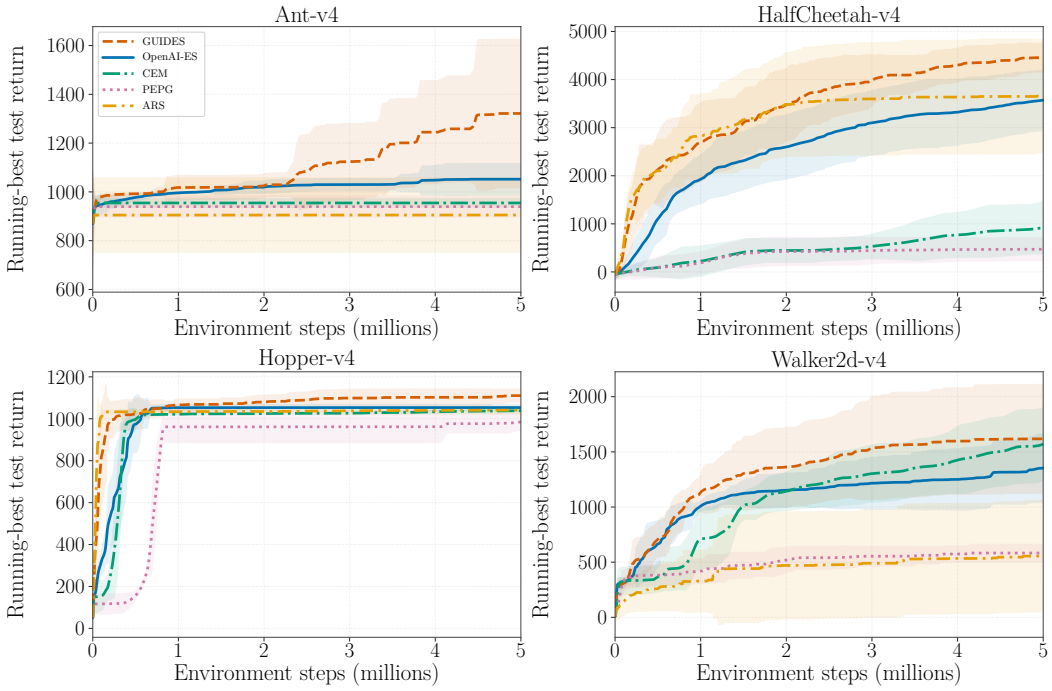

Figure 4: Learning curves on Mujoco-based continuous control benchmarks

Stability Analysis. Figure 5 further illustrates the distribution of test returns across independent runs. On Hopper, GUIDES maintains a relatively concentrated distribution while shifting the overall performance to a higher return region. On HalfCheetah, GUIDES also exhibits a compact high-return distribution, in contrast to the substantially broader distribution of ARS and the low-return distributions of CEM and PEPG. The behavior is different on Ant and Walker2d. In these two environments, GUIDES achieves the highest overall return level but shows greater run-to-run variability. In particular, the Ant results span a wider range than those of OpenAI-ES, CEM, and PEPG, while Walker2d also exhibits a relatively broad distribution despite its higher average performance. Therefore, the main advantage of GUIDES on these environments lies in its ability to reach substantially stronger policies rather than uniformly reducing performance variance. Overall, the results show that GUIDES consistently improves solution quality across the four benchmarks, while the stability of the resulting optimization remains dependent on the dynamics of each environment.

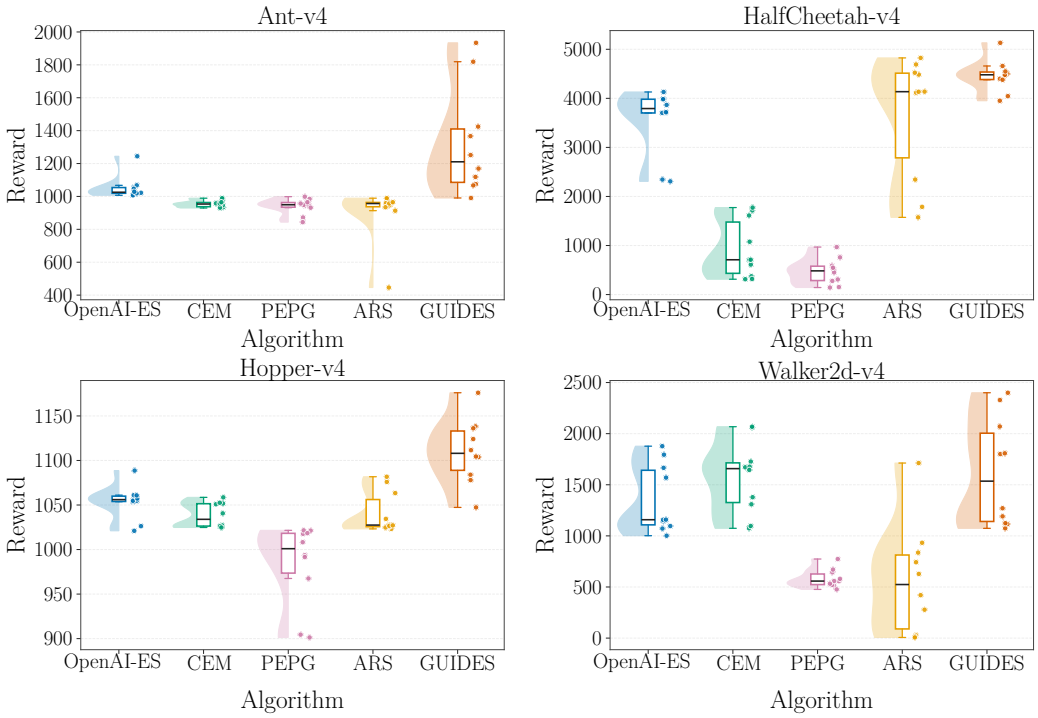

Figure 5: Stability Analysis on Mujoco-based continuous robotic benchmarks

F Detailed Workflow and VM Configuration

F.1 Workflow Directed Acyclic Graph

A workflow W_i is modeled as a directed acyclic graph (DAG) $G_i = (O_i, E_i)$. The node set $O_i = \{o_{i1}, o_{i2}, \ldots, o_{in_i}\}$ contains the computational tasks of W_i, while E_i records their directed dependencies. An edge $(o_{ij}, o_{ik}) \in E_i$ means that o_{ij} must finish before o_{ik} can begin.

Following commonly adopted experimental settings in cloud workflow scheduling (Huang et al., 2022; Shen et al., 2025), we consider four widely studied workflow patterns: Montage, CyberShake, Inspiral, and SIPHT (Deelman et al., 2015) . As illustrated in Figure 6, these patterns contain different combinations of sequential stages, parallel branches, and fork–join dependencies. Table 8 summarizes the three workflow scales used in this study: small, medium, and large.

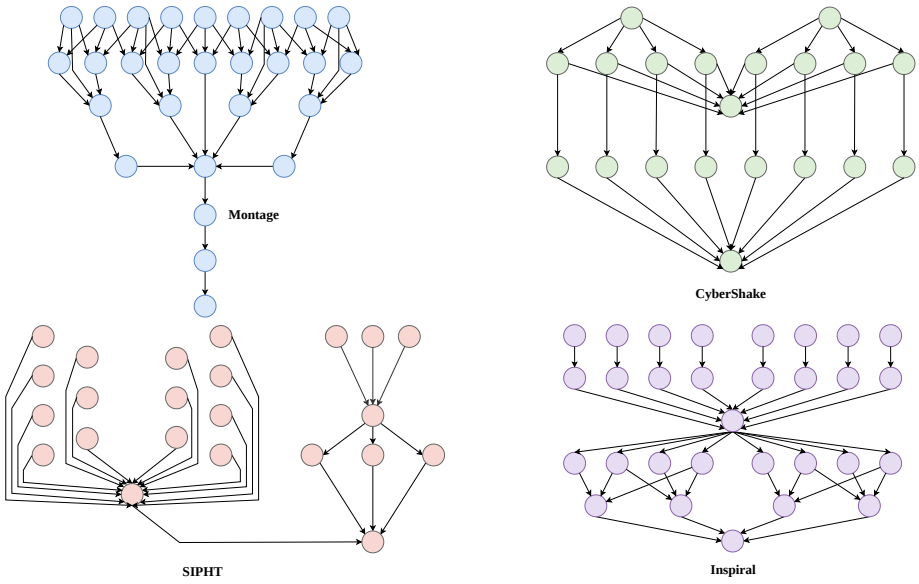

Figure 6: The workflows studied in this paper

Table 8: The three workflow sets used in this study.

Workflow set	**Workflow pattern / number of tasks**			
Small	CyberShake/30	Montage/25	Inspiral/30	SIPHT/30
Medium	CyberShake/50	Montage/50	Inspiral/50	SIPHT/60
Large	CyberShake/100	Montage/100	Inspiral/100	SIPHT/100

F.2 VM Configuration

To represent the heterogeneity of cloud resources, we consider six VM types with different computational capacities and rental costs. As shown in Table 9, the number of vCPUs ranges from 2 to 48, while the memory capacity ranges from 8 GB to 192 GB. The hourly rental cost increases accordingly from \$0.096 to \$2.304 (Shen et al., 2025; Bermudez et al., 2013). These VM types provide different trade-offs between computational capability and monetary cost, allowing the scheduler to select appropriate resources according to the current task requirements and system conditions. All VM types can be provisioned on demand during the scheduling process.

Table 9: Configuration of VM types.

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

G Hyperparameter Sensitivity Analysis on CADWS

We examine the sensitivity of GUIDES to two optimization hyperparameters: (i) the subspace dimension K, which specifies the number of basis directions in the critic-guided search subspace; and (ii) the critic learning rate η_c, which controls the step size of the critic parameter updates.

The Base configuration uses $K = 16$ and $\eta_c = 3e-4$. We vary one parameter at a time while keeping the other one at its Base value. All configurations use the same environment and experimental settings as the main CADWS experiments, with 10 independent random seeds per configuration. Table 10 reports the mean and standard deviation of total scheduling costs across these runs. Bold values indicate the lowest mean cost within each parameter group.

Table 10: Hyperparameter Analysis on CADWS.

(a) Subspace Dimension			**(b) Critic Learning Rate**		
Value	Mean	Std	Value	Mean	Std
12	73.8823	2.0302	$5e-5$	73.9356	6.5504
16	**72.6150**	1.8184	$1e-4$	73.6486	**0.9459**
24	73.7674	2.1092	$3e-4$	**72.6150**	1.8184
32	72.6243	**1.5104**	$5e-4$	73.2683	5.1978

Results and Analysis. Table 10 shows that subspace dimensions of 16 and 32 yield comparable mean cost, while $K = 32$ requires additional gradient computations to construct the larger search subspace. We therefore select $K = 16$ to balance performance and computational cost. For the critic learning rate, $\eta_c = 3e-4$ achieves the lowest mean cost among the tested settings and is adopted for the CADWS experiments.

H Detailed Ablation Studies

In this part, we conduct two sets of ablation studies, one focusing on our network and the other focusing on the proposed algorithm.

H.1 Network Ablation Studies

To examine the contribution of each component in SWAN, we compare the complete network with variants that remove the global module, the Task–Transformer, or explicit task–VM compatibility, as well as a variant that replaces MWDAGs with full workflow DAGs. These variants are evaluated across three workflow scales and five deadline coefficients. The variant without directional GAT is additionally evaluated at $\gamma_{\mathrm{dl}} = 1$ across all three scales. Table 11 reports the mean total scheduling cost and standard deviation, with lower costs indicating better performance. The results assess the contribution of individual modules and whether MWDAGs preserve scheduling performance while reducing the graph encoding workload.

- **SWAN** achieves the lowest mean total cost in most evaluated settings. Its advantage over the component-removal variants supports the effectiveness of combining workflow structure, global context, and task–VM assignment information within a unified policy network.
- **w/o Global Module** incurs higher mean costs than SWAN, with larger absolute performance gaps at the large scale. This suggests that global workflow context helps the policy account for competing resource demands across active workflows, particularly as the scheduling problem grows.
- **w/o Task–Transformer** performs worse than SWAN across all evaluated settings. This result supports the value of contextual representations that capture relationships among candidate elements, allowing the policy to assess their relative suitability beyond their individual features.
- **w/o Assignment Compatibility** consistently increases mean scheduling cost, with particularly clear degradation at the large scale. This indicates that explicit pair-specific information complements the separate task and VM representations, helping the policy evaluate the execution time, deadline slack, and incremental cost of candidate assignments.
- **Full Workflow DAGs** achieve scheduling performance similar to that of SWAN, with small differences in mean cost. However, retaining completed and scheduled tasks requires the

Table 11: SWAN ablations across small (S), medium (M), and large (L) workflow scales and deadline coefficients. Values are mean total cost ± standard deviation. Bold means indicate the lowest reported cost in each row, determined before rounding.

Scale / γ_{dl}	SWAN (ours)	Full Workflow DAGs	w/o Global Module	w/o Task-Transformer	w/o Assignment Compatibility	w/o Directional GAT
S / 1	$\mathbf{14.51 \pm 0.25}$	14.60 ± 0.12	14.75 ± 0.28	14.78 ± 0.29	14.85 ± 0.22	14.70 ± 0.12
S / 1.5	$\mathbf{13.35 \pm 0.24}$	13.37 ± 0.33	13.35 ± 0.35	13.59 ± 0.48	13.55 ± 0.39	13.69 ± 0.50
S / 2	$\mathbf{12.10 \pm 0.13}$	12.26 ± 0.28	12.34 ± 0.28	12.50 ± 0.54	12.36 ± 0.35	12.63 ± 0.44
S / 2.5	$\mathbf{11.69 \pm 0.36}$	11.79 ± 0.34	11.98 ± 0.33	12.05 ± 0.45	11.92 ± 0.35	11.84 ± 0.52
S / 3	$\mathbf{11.46 \pm 0.35}$	11.54 ± 0.34	11.75 ± 0.32	11.94 ± 0.64	11.79 ± 0.41	11.66 ± 0.53
M / 1	20.32 ± 0.42	$\mathbf{20.31 \pm 0.58}$	21.28 ± 0.71	21.30 ± 0.60	21.39 ± 0.95	20.53 ± 0.54
M / 1.5	$\mathbf{19.40 \pm 0.38}$	19.53 ± 0.57	19.84 ± 0.73	19.85 ± 0.46	20.06 ± 0.81	19.79 ± 0.81
M / 2	$\mathbf{17.80 \pm 0.48}$	18.05 ± 0.44	18.36 ± 0.85	18.60 ± 0.72	18.60 ± 1.05	18.55 ± 1.04
M / 2.5	17.16 ± 0.58	$\mathbf{17.12 \pm 0.47}$	17.61 ± 0.73	17.87 ± 0.60	17.76 ± 0.63	17.60 ± 1.15
M / 3	$\mathbf{16.91 \pm 0.72}$	16.92 ± 0.36	17.35 ± 0.73	17.45 ± 0.55	17.54 ± 0.51	17.12 ± 1.06
L / 1	$\mathbf{27.71 \pm 0.73}$	27.83 ± 1.01	28.76 ± 1.36	28.82 ± 1.22	29.30 ± 1.57	28.77 ± 0.84
L / 1.5	26.80 ± 1.01	$\mathbf{26.75 \pm 0.73}$	27.80 ± 1.50	27.46 ± 1.22	27.93 ± 0.95	27.48 ± 1.51
L / 2	$\mathbf{25.33 \pm 0.98}$	25.42 ± 0.84	26.57 ± 1.71	26.22 ± 1.29	26.64 ± 1.20	26.67 ± 1.90
L / 2.5	$\mathbf{24.05 \pm 1.18}$	24.12 ± 1.13	25.29 ± 1.86	25.13 ± 1.23	25.47 ± 1.11	24.93 ± 2.19
L / 3	$\mathbf{23.61 \pm 1.13}$	23.79 ± 1.01	24.83 ± 1.62	24.48 ± 1.38	24.99 ± 1.06	24.56 ± 1.73

graph encoder to process additional nodes and edges. MWDAGs remove these tasks and their incident edges while preserving the unresolved workflow structure, thereby reducing the graph encoding workload without an evident loss in scheduling performance.

- **w/o Directional GAT** produces higher mean costs than SWAN at all three scales, with the largest absolute gap at the large scale. This supports the contribution of directional graph encoding to representing task dependencies under the evaluated deadline setting.

Overall, the results support the complementary roles of the components in SWAN. Global context and Transformer-based representations help contextualize scheduling decisions, while explicit task–VM compatibility and directional graph encoding provide assignment-specific and structural information. Meanwhile, MWDAG preserves scheduling performance comparable to that of full workflow DAGs while requiring fewer nodes and edges to be processed. Together, these components support effective scheduling with a compact workflow representation.

H.2 Algorithm Ablation Studies

We conduct an ablation study to examine the search strategy of GUIDES. Specifically, we compare the critic-guided subspace search used by GUIDES with mixed-search strategies that additionally introduce full-space perturbations. All variants use the same SWAN policy architecture and experimental settings, with the complete GUIDES algorithm serving as the baseline. The results are reported as the mean total scheduling cost ± standard deviation over 10 test evaluations across the S, M, and L workflow scales and five deadline coefficients. Lower costs indicate better scheduling performance.

Search-strategy ablation. To examine whether introducing full-space exploration improves the critic-guided search, we construct mixed-search variants that combine full-space and guided-subspace perturbations. For generations $g > 1$, the perturbation is defined as

$$\epsilon_{g,i} = \sqrt{\alpha}\,\epsilon_{g,i}^{\mathrm{full}} + \sqrt{1-\alpha}\,\epsilon_{g,i}^{\mathrm{subspace}}, \tag{67}$$

where $\epsilon_{g,i}^{\mathrm{full}} = \xi_{g,i}$ with $\xi_{g,i} \sim \mathcal{N}(0, I_d)$, and $\epsilon_{g,i}^{\mathrm{subspace}} = U_g z_{g,i}$ with $z_{g,i} \sim \mathcal{N}(0, I_K)$. The two noise vectors are sampled independently, and U_g is the orthonormal critic-guided basis defined in Section D. The coefficient $\alpha \in [0, 1]$ controls the contribution of full-space exploration. When $\alpha = 0$, perturbations are restricted to the critic-guided subspace, corresponding to GUIDES. When

$\alpha = 1$, the guided component vanishes, recovering the full-space Gaussian perturbation used by OpenAI-ES. We compare GUIDES with mixed-search variants using $\alpha \in \{0.2, 0.5, 0.8\}$.

Table 12: Search-strategy ablation comparing guided-subspace search in GUIDES with mixed search across workflow scales and deadline coefficients. Entries report mean ± standard deviation of total scheduling costs. Bold entries indicate the lowest mean cost in each setting, with both the mean and standard deviation highlighted.

Scale / γ_{dl}	**GUIDES** $\alpha = 0$	**Mixed search** $\alpha = 0.2$	**Mixed search** $\alpha = 0.5$	**Mixed search** $\alpha = 0.8$
S / 1	$\mathbf{14.51 \pm 0.25}$	15.38 ± 2.62	15.52 ± 2.57	16.18 ± 2.69
S / 1.5	$\mathbf{13.35 \pm 0.24}$	14.22 ± 2.78	14.23 ± 2.75	15.17 ± 2.79
S / 2	$\mathbf{12.10 \pm 0.13}$	13.08 ± 2.86	13.21 ± 2.81	14.11 ± 2.98
S / 2.5	$\mathbf{11.69 \pm 0.36}$	12.55 ± 2.77	12.79 ± 2.67	13.39 ± 2.92
S / 3	$\mathbf{11.46 \pm 0.35}$	12.31 ± 2.48	12.51 ± 2.39	12.96 ± 2.58
M / 1	$\mathbf{20.32 \pm 0.42}$	22.64 ± 6.29	22.73 ± 6.23	23.78 ± 6.07
M / 1.5	$\mathbf{19.40 \pm 0.38}$	21.57 ± 6.36	21.55 ± 6.35	22.77 ± 6.17
M / 2	$\mathbf{17.80 \pm 0.48}$	20.06 ± 6.62	20.36 ± 6.48	21.51 ± 6.34
M / 2.5	$\mathbf{17.16 \pm 0.58}$	19.38 ± 6.57	19.29 ± 6.57	20.66 ± 6.34
M / 3	$\mathbf{16.91 \pm 0.72}$	19.01 ± 6.41	19.04 ± 6.37	20.22 ± 6.17
L / 1	$\mathbf{27.71 \pm 0.73}$	31.88 ± 11.44	32.04 ± 11.47	34.13 ± 11.13
L / 1.5	$\mathbf{26.80 \pm 1.01}$	30.63 ± 11.54	30.88 ± 11.56	33.10 ± 11.01
L / 2	$\mathbf{25.33 \pm 0.98}$	29.35 ± 11.74	29.63 ± 11.70	31.74 ± 11.15
L / 2.5	$\mathbf{24.05 \pm 1.18}$	28.40 ± 11.76	28.87 ± 11.72	30.60 ± 11.31
L / 3	$\mathbf{23.61 \pm 1.13}$	27.76 ± 11.67	28.21 ± 11.64	29.95 ± 11.23

As shown in Table 12, GUIDES ($\alpha = 0$) achieves the lowest mean total cost in all 15 evaluated settings. It also produces substantially smaller standard deviations than the mixed-search variants, and this difference becomes more pronounced at the M and L scales. These results indicate that restricting perturbations to the critic-guided subspace provides both better scheduling performance and more consistent results than combining guided and full-space exploration.

Among the mixed-search variants, $\alpha = 0.2$ achieves the lowest mean cost in 13 of the 15 settings. Although the relationship is not strictly monotonic in every setting, the overall cost generally increases as α becomes larger. The performance degradation also grows with the workflow scale, indicating that excessive full-space exploration becomes increasingly inefficient as the scheduling problem expands. These findings support the use of pure critic-guided subspace search in GUIDES.