DOI: 10.2478/ias-2026-0024 • JIAS • 21(6) • 2026 • 501–526

Article

Self-Adaptive Multi-Cloud Job Scheduling Using Deep Reinforcement Learning and Workload Forecasting

Ashutosh Bhatt [1], Ambika Aggarwal [2,*], Khushboo Jain [2,*]

1 School of Science & Technology, SRHU, Dehradun, Uttarakhand, India; ashutoshbhatt15@gmail.com
2 School of Computer Science, UPES, Dehradun, Uttarakhand, India
* Correspondence author: ambika2488@gmail.com; khushboojain2806@gmail.com

Abstract: The elastic, heterogeneous, and price-volatile nature of modern multi-cloud ecosystems makes job scheduling a persistently difficult problem. Static heuristics and metaheuristics optimize against a single snapshot of the system and degrade sharply when demand spikes, spot-market prices shift, or virtual machine (VM) availability changes across providers. This paper proposes SAMC-DRL, a self-adaptive multi-cloud scheduling framework that couples a deep reinforcement learning (DRL) scheduler with a short-horizon workload-forecasting module. The forecasting module is an attention-augmented bidirectional Long Short-Term Memory (Attention-BiLSTM) network that predicts aggregate resource demand from historical traces; its predictions are folded into the scheduler's state representation so that placement decisions are made proactively rather than reactively. The scheduler is trained with Proximal Policy Optimization (PPO) using an adaptive, multi-objective reward that jointly balances makespan, monetary cost, energy consumption, and service-level-agreement (SLA) violations, with weights that self-tune to the observed operating regime. We evaluate SAMC-DRL on two public production traces—the Google Borg cluster trace and the Alibaba cluster trace—within an extended CloudSim simulation spanning three emulated cloud providers. Across workloads of 100–4000 concurrent tasks, SAMC-DRL reduces makespan by up to 11.3%, energy by up to 12.5%, and monetary cost by up to 14.1% relative to PPO without forecasting and lowers the SLA-violation rate from 8.4% to 3.2% relative to a Deep Q-Network (DQN) baseline. The Attention-BiLSTM forecaster attains a mean absolute percentage error (MAPE) of 5.8%, outperforming ARIMA, support vector regression (SVR), LSTM, GRU, and a vanilla Transformer. An ablation study confirms that workload forecasting and the adaptive reward are the two largest contributors to the observed gains. The results indicate that forecasting-aware, self-adaptive DRL is a practical path toward cost- and energy-efficient scheduling in operational multi-cloud platforms. All scheduling results are averaged over ten independent seeds and are statistically significant under Holm-corrected paired t-tests and exact Wilcoxon signed-rank tests, and additional experiments characterize the framework under flash-crowd, aperiodic and adversarial anti-forecast bursts, under 30-day forecast drift, and across DAG workloads of differing depth and branching factor.

Keywords: Multi-cloud computing, job scheduling, deep reinforcement learning, BiLSTM, service-level agreement

1. Introduction

Cloud computing has become the default substrate for enterprise and scientific workloads, and organizations increasingly distribute these workloads across several providers simultaneously. A multi-cloud strategy avoids vendor lock-in, exploits price differentials between providers, improves fault tolerance through geographic and administrative diversity, and allows operators to place latency-sensitive tasks close to their data. Yet the same diversity that makes multi-cloud attractive also makes scheduling substantially harder. A multi-cloud scheduler must reason simultaneously about heterogeneous VM types with different performance-per-dollar profiles, time-varying spot prices, provider-specific availability and egress charges, and a workload whose intensity and composition shift continuously.

Classical schedulers address these difficulties with heuristics such as round-robin, Min-Min, and

© 2026 Ashutosh Bhatt et al., published by Cerebration Science Publishing Co., Limited. This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0

Max-Min, or with metaheuristics such as genetic algorithms (GA) and particle swarm optimization (PSO). Both families share a structural limitation: they optimize against the state of the system at the moment of decision and implicitly assume that the near future resembles the present. In production, that assumption fails routinely diurnal cycles, flash crowds, batch-job bursts, and correlated failures all cause demand to deviate sharply from any static expectation. When demand deviates, a reactive scheduler either over-provisions, wasting energy and money, or under-provisions, breaching SLAs. Recent algorithm-level surveys of DRL for cloud scheduling reach the same conclusion: the dynamic, partially observable, and non-stationary character of cloud workloads is precisely what static optimization handles poorly [1], [2].

Because the distinction between reactive and proactive scheduling is the conceptual basis of this work, we make it precise here rather than leaving it to intuition. Let $O_t = \{O_1, \ldots, O_t\}$ denote the history of observations available to the scheduler at decision time t, and let $D_{t+1:t+h}$ denote the aggregate resource demand that will actually arrive over the next H intervals. A scheduler is reactive when its placement rule is a function of observed quantities only, $a_t = \pi(O_t)$: a change in demand can influence a decision only after it has already appeared in the queue, in the measured utilization, or in a price feed. The corrective action therefore lags the event that made it necessary by at least one decision interval, and that lag is paid for in queueing delay, in emergency over-provisioning, or in missed deadlines. A scheduler is proactive when its placement rule additionally conditions on an explicit estimate of demand that has not yet arrived (Equations (1) and (2)):

$$a_t = \pi(O_t, f_t) \tag{1}$$

$$f_t = E[D_{t+1:t+h} \mid O_t] \tag{2}$$

so that an action taken at time t may be justified entirely by future load. The two families are thus separated by the information set on which the policy is defined, not by the sophistication of the algorithm that produces it: a DRL scheduler whose state contains only the instantaneous system snapshot is reactive in exactly the same sense as round-robin, even though its policy is learned rather than hand-coded, because no policy defined on O_t alone can express "acquire capacity at the cheaper provider now because the surge starts in twenty minutes". SAMC-DRL is proactive in this precise sense: the forecast block f_t produced by the Attention-BiLSTM module is a first-class element of the scheduler state (Section 3.3), and the adaptive reward of Section 4.4 is itself a function of predicted rather than merely observed utilization.

Deep reinforcement learning offers a principled alternative. By framing scheduling as a sequential decision problem, a DRL agent learns a placement policy directly from interaction with the environment, adapting to patterns that are difficult to encode by hand. DRL schedulers have demonstrated strong results for task scheduling [3, 4] real-time workflow scheduling [5], hierarchical scheduling [6], and energy-efficient workflow scheduling [7]. Two gaps nonetheless persist. First, most DRL schedulers remain fundamentally reactive: the agent observes the current queue and resource state but receives no explicit signal about where demand is heading, so it cannot prepare resources before a surge. Second, reward functions are typically fixed at design time, which forces a single static trade-off among makespan, cost, energy, and SLA compliance—even though the right trade-off depends on the operating regime, with cost dominating during low-utilization periods and SLA protection dominating during peaks.

Workload forecasting addresses the first gap. A large body of recent work shows that aggregate cloud demand is predictable at short horizons using deep sequence models: LSTM and GRU networks [8], attention-based and multivariate models [8, 9] uncertainty-aware and transfer-learning predictors [10, 11] and transformer-based forecasters for predictive auto-scaling [12, 13]. Predictive auto-scaling systems built on these forecasters have been reported to improve service quality by roughly 20% while reducing resource waste by roughly 15% [14]. The natural next step—and the focus of this paper—is to feed such forecasts directly into a DRL scheduler so that placement becomes proactive rather than corrective. Closing the second gap requires the reward itself to become state-dependent, so that the policy is rewarded for spending on capacity only when a surge is anticipated.

This paper makes the following five contributions:

(1) We propose SAMC-DRL, a self-adaptive multi-cloud scheduling framework that integrates an Attention-BiLSTM workload forecaster with a PPO-based scheduler, so that scheduling decisions are conditioned on predicted near-future demand rather than only on the instantaneous system state.

(2) We design an adaptive multi-objective reward that self-tunes the relative weighting of makespan, cost, energy, and SLA violations according to the observed utilization regime, removing the need for a single hand-fixed trade-off.

(3) We formulate multi-cloud scheduling as a Markov decision process whose state explicitly encodes per-provider price and availability, queue and deadline information, and the forecast demand vector, and we provide a computational-complexity analysis of the resulting decision procedure.
(4) We provide a reproducible evaluation on the Google Borg and Alibaba production traces within an extended three-provider CloudSim environment, comparing against heuristic, metaheuristic, and DRL baselines across makespan, energy, cost, SLA violation, scalability, and an ablation study.
(5) We characterize the operating envelope of the framework with a statistically grounded evaluation: ten-seed significance testing with effect sizes and non-parametric confirmation, a sensitivity analysis of the adaptive-reward hyperparameters, a 30-day forecast-drift study, and a stress test under flash-crowd, aperiodic, and adversarial anti-forecast burst regimes.

The remainder of the paper is organized as follows. Section 2 reviews related work on DRL scheduling and workload forecasting. Section 3 formalizes the system model and problem. Section 4 details the SAMC-DRL methodology, including its computational complexity. Section 5 describes the datasets and the experimental setup. Section 6 presents and discusses the results. Section 7 examines threats to validity and limitations, and Section 8 concludes.

2. Related Work

2.1 Heuristic and Metaheuristic Scheduling

Heuristic schedulers such as round-robin, Min-Min, and Max-Min remain widely deployed because they are simple and fast, but they ignore future demand and typically optimize a single objective. Metaheuristics—genetic algorithms, particle swarm optimization, cat-swarm optimization, and Harris-Hawks optimization—search a larger solution space and can incorporate multiple objectives; hybrids such as fault-tolerant, trust-based schedulers that combine Harris-Hawks optimization with DRL have been proposed for multi-cloud settings [15]. However, every metaheuristic must resolve its optimization whenever conditions change, which is expensive at scale and still fundamentally reactive. Cost-efficiency studies on Alibaba traces confirm that statically tuned policies leave substantial savings unrealized when workloads are mixed and bursty [16].

2.2 Deep Reinforcement Learning for Cloud Scheduling

DRL reframes scheduling as sequential decision-making. Foundational systems such as DeepRM [17] and Decima [18] demonstrated that policy-gradient and graph-based agents can learn scheduling strategies competitive with or superior to hand-designed heuristics. Building on value-based methods derived from the Deep Q-Network (DQN) [19] and on policy-gradient methods such as Asynchronous Advantage Actor-Critic (A3C) [46] and PPO [20], recent work has applied DRL to multi-objective task scheduling for energy and cost optimization [3], multi-cloud task scheduling [5], real-time workflow scheduling [4], and hierarchical scheduling [21]. Actor-critic formulations schedule efficiently while respecting quality-of-service (QoS) constraints [2], and server-performance-aware DRL schedulers adapt placement to real-time node load [21, 22]. Graph-neural-network policies trained with PPO model task dependencies explicitly and reduce makespan and energy on workflow workloads [14, 23]. SLA-aware multi-objective RL has been used to allocate GPU and CPU resources under deadline and budget constraints, reporting large reductions in training time and cost while improving SLA compliance [24]. Despite this progress, the majority of these schedulers consume only the present system state; they neither condition decisions on an explicit forecast of future demand nor adapt their reward weighting at run time.

2.3 Cloud Workload Forecasting

Workload forecasting has matured from classical autoregressive integrated moving-average (ARIMA) models [25] to deep sequence models. LSTM [26] and GRU [27] networks capture long-range temporal dependencies in resource-utilization traces, and multivariate attention networks such as MAG-D improve accuracy by fusing CPU, memory, disk, and network signals [8]. Uncertainty-aware and transfer-learning predictors additionally quantify forecast confidence and transfer across workloads [10]. Transformer-based forecasters—including online ensemble transformers for predictive auto-scaling [12], lightweight frequency transformers [28], multivariate resource forecasters [9], and adversarial transformers [29]—now define the state of the art for cloud demand prediction, while Prophet/LSTM hybrids drive proactive Kubernetes auto-scaling [30]. DRL has itself been used for workload prediction in federated clouds [31]. These forecasters are typically deployed for auto-scaling rather than scheduling;

coupling them directly to a scheduling policy remains comparatively underexplored, which is the gap this work targets.

2.4 Multi-Cloud and Federated Resource Management

Multi-cloud and federated-cloud management add provider heterogeneity, cross-provider migration, and cost arbitrage. Decentralized, cost-aware deep-learning frameworks combine forecasting with adaptive provisioning for load balancing and SLA optimization [31], and reinforcement-learning frameworks such as FedTaskRL target privacy-respecting federated cloud-edge scheduling [32]. Efficient DRL task schedulers designed specifically for multi-cloud environments handle VM availability and inter-cloud task migration [5, 14]. Our work differs from these in treating short-horizon forecasting as a first class input to a self-adaptive PPO scheduler that reasons jointly over per-provider price, availability, energy, and SLA risk.

2.5 Positioning of This Work

Table 1 situates SAMC-DRL against representative recent systems along four axes: whether the scheduler is forecast-aware, whether it optimizes multiple objectives, whether it explicitly models a multi-cloud environment, and whether its reward adapts at run time. To our knowledge, SAMC-DRL is among the first frameworks to combine an attention-based workload forecaster, a PPO scheduler, an adaptive multi-objective reward, and an explicit multi-cloud cost and availability model in a single, reproducible pipeline evaluated on two production traces.

Table 1. Comparison of representative recent scheduling frameworks.

Approach	Method	Forecast-aware	Multi-objective	Multi-cloud	Adaptive reward
DeepRM [22]	Policy gradient	No	Partial	No	No
RL-MOTS [3]	DQN	No	Yes	No	No
Multi-cloud DRL [5]	DQN	No	Yes	Yes	No
GNN-PPO [23]	PPO + GNN	No	Yes	No	No
SLA-MORL [32]	Multi-obj RL	No	Yes	No	Partial
SAMC-DRL (Proposed Work)	PPO + BiLSTM	Yes	Yes	Yes	Yes

3. System Model and Problem Formulation

3.1 Multi-Cloud Environment

We consider a federation of P cloud providers (in our evaluation P = 3, emulating AWS, Azure, and GCP-like providers). Each provider p exposes a set of heterogeneous VM types. A VM of type v at provider p is characterized by its processing capacity, measured in millions of instructions per second (MIPS); its memory; its network bandwidth; an energy-per-unit-time coefficient; and a time-varying on-demand or spot price c(p,v,t). At any time t, a subset of VM types may be unavailable at a given provider, modeled by an availability indicator $a(p,v,t) \in \{0, 1\}$, where 1 denotes available and 0 denotes unavailable. Incoming work arrives as a stream of jobs. Each job is a directed acyclic graph (DAG) of tasks with precedence constraints. A task i is described by its length, measured in millions of instructions (MI); its data size for transfer; an arrival time; and a soft deadline d_i that defines its SLA. Egress and inter-cloud transfer costs are charged whenever a task's data must move between providers.

3.2 Optimization Objectives

The scheduler maps each ready task to a (provider, VM) pair, and we optimize four objectives simultaneously. Makespan is the completion time of the last task in a job. Monetary cost aggregates compute and data-transfer charges across providers. Energy is the integral of VM power over execution time. The SLA-violation rate is the fraction of tasks that complete after their deadline. Let M, C, E, and S denote the normalized makespan, cost, energy, and SLA-violation rate, each scaled to the interval [0, 1] so that the four quantities are commensurable. The scheduling goal is to minimize the weighted objective J in Equation (3):

$$J = w_M \cdot M + w_C \cdot C + w_E \cdot E + w_S \cdot S, \ with \ w_M + w_C + w_E + w_S = 1 \qquad (3)$$

Here the weight vector w = (w_M, w_C, w_E, w_S) encodes the relative importance of the four objectives and is constrained to sum to one, so that J is a convex combination of the normalized objectives. Intuitively, J is a single scalar "cost-of-a-decision" score in which a lower value is always better; the

constraint $w_M + w_C + w_E + w_S = 1$ keeps the four objectives on a common scale and prevents any objective from dominating merely because of its magnitude. Crucially, w is *not* fixed: Section 4.4 makes it a function of the observed utilization regime, so the agent shifts emphasis toward SLA protection during peaks and toward cost during troughs. To ensure fair aggregation, all objectives (M, C, E, and S) are normalized to the range $[0,1]$using min-max normalization before computing the weighted objective in Equation (3). This prevents any metric from dominating the reward due to differences in scale.

3.3 Markov Decision Process Formulation

We cast scheduling as a Markov decision process (MDP) defined by the tuple (S, A, T, R, γ), where S is the state space, A the action space, T the transition dynamics, R the reward function, and $\gamma \in (0, 1)$ the discount factor. At each decision step the agent observes a state, selects an action (a placement), receives a reward, and transitions to the next state. The components are defined as follows.

State: The state s_t concatenates four blocks: (i) a *queue block* summarizing the ready tasks—their lengths, data sizes, and remaining slack to deadline; (ii) a *resource block* giving, for each (provider, VM) pair, the current utilization, availability a(p,v,t), and price c(p,v,t); (iii) a *forecast block* f_t, the predicted aggregate demand over the next H steps produced by the Attention-BiLSTM module; and (iv) a *context block* containing the current normalized utilization regime u_t. The forecast block f_t is what distinguishes SAMC-DRL from reactive DRL schedulers.

Action: An action a_t assigns the head-of-queue ready task to one available (provider, VM) pair. Unavailable pairs those with a(p,v,t) = 0 are masked out of the policy's action distribution so that the agent never selects an infeasible placement.

Transition: The environment advances simulated time, executes the placed tasks, updates utilization, prices, and availability, and admits newly arrived tasks. Transitions are stochastic because arrivals, prices, and availability evolve jointly from the production trace and from the price and availability models.

Reward: The immediate reward rt is the negative of the incremental weighted objective of Equation (3), plus a potential-based shaping term that penalizes deadline risk and idle energy, as given in Equations (4)–(6). The discount factor γ trades off immediate against long-term return, encouraging the agent to value future SLA compliance rather than only the current placement.

We now give the reward in closed form. Let J(s) denote the weighted objective of Equation (3) evaluated on the system states, using the adaptive weight vector w_t of Section 4.4 rather than a fixed w. The immediate reward is the negative increment of that objective plus a shaping term F:

$$r_t = -[J(s_{t+1}) - J(s_t)] + F(s_t, a_t, s_{t+1}) \tag{4}$$

Defining the reward on the increment rather than on the absolute level of J makes it invariant to the scale of an episode: the agent is credited only for the change its own placement produced, and not for having inherited a favourable state. The shaping term F in Equation (4) is defined in Equation (5) as a potential difference in the sense of Ng et al. [33], a form that leaves the optimal policy of the underlying MDP unchanged under the conditions stated below:

$$F(s_t, a_t, s_{t+1}) = \gamma\Phi(s_{t+1}) - \Phi(s_t) \tag{5}$$

Here, the potential Φ in Equation (6) aggregates the two operational quantities whose consequences would otherwise become observable only at the end of an episode, namely accumulated deadline risk and idle energy:

$$\Phi(s) = -\frac{\eta_d}{|Q(s)|}\sum_{i\in Q(s)} \max\left(0,1 - \frac{slack_{i(s)}}{d_i}\right) - \frac{\eta_e}{|V(s)|}\sum_{(p,v)\in V(s)} P_{idle(p,v)\mathbb{1}_{(p,v)is\ idle\ in\ s}} \tag{6}$$

Here Q(s) is the set of tasks resident in the queue in states; $slack_i(s) = d_i - (t + \hat{c}_i)$ is the time remaining to the soft deadline d_i of task i once its estimated completion time $\hat{c}_i$ is taken into account, so the ratio $slack_i/d_i$ falls toward zero as a task approaches its deadline and the corresponding term saturates at one; V(s) is the set of currently provisioned VMs; P_idle(p,v) is the idle power draw of VM type v at provider p, normalized by the maximum idle draw in the catalog; and $\mathbb{1}$ is the indicator function. Both sums are divided by their cardinalities so that the magnitude of the shaping signal does not grow with workload scale, which keeps the same coefficients valid from 100 to 4000 concurrent tasks. The coefficients η_d = 0.5 and η_e = 0.2 were selected by grid search on the validation split and are the only two shaping hyperparameters.

The practical effect of the first term is that an action which raises imminent deadline risk is penalized at the moment it is taken rather than many steps later when the violation finally materializes, which is what makes credit assignment tractable over episodes of several thousand placements; the second term

discourages the agent from hoarding capacity after a predicted surge fails to materialize. Because F is a potential difference, its discounted contribution over an episode of T steps telescopes to $\gamma^T\Phi(s_T) - \Phi(s_0)$. The policy-invariance result of Ng et al. [33] holds when Φ depends on the state alone, the shaping term uses the same discount γ as the return, and the potential of terminal states is fixed at zero (or the horizon is infinite); under these conditions, adding F leaves the optimal policy of the MDP unchanged. In our finite-horizon episodes $\Phi(s_T)$ is not forced to zero, so the terminal term $\gamma^T\Phi(s_T)$ is bounded but can depend on the policy. Moreover, the guarantee concerns the exact optimal policy of a fixed MDP; it does not extend formally to a PPO policy learned with function approximation, nor to the time-varying weights w_t of the base reward. We therefore claim only that shaping is intended to accelerate credit assignment and preserves optimality under the stated conditions, not that it provably leaves the converged PPO solution unaltered. All quantities entering Equations (4)–(6) are computed from realized simulator outcomes, never from forecasts, so a forecast error can bias which action the policy prefers but can never corrupt the learning signal itself — a property we return to in Section 4.6.

4. The SAMC-DRL Methodology

SAMC-DRL comprises three cooperating components: a workload-forecasting module, a PPO-based scheduling agent, and an adaptive reward controller. Figure 1 shows how they interact. The monitoring layer collects per-resource traces from all providers; the forecasting module turns recent history into a short-horizon demand forecast; the state builder fuses that forecast with queue, resource, and cost information; the PPO agent emits a placement; and the adaptive reward controller scores the outcome and feeds it back for learning.

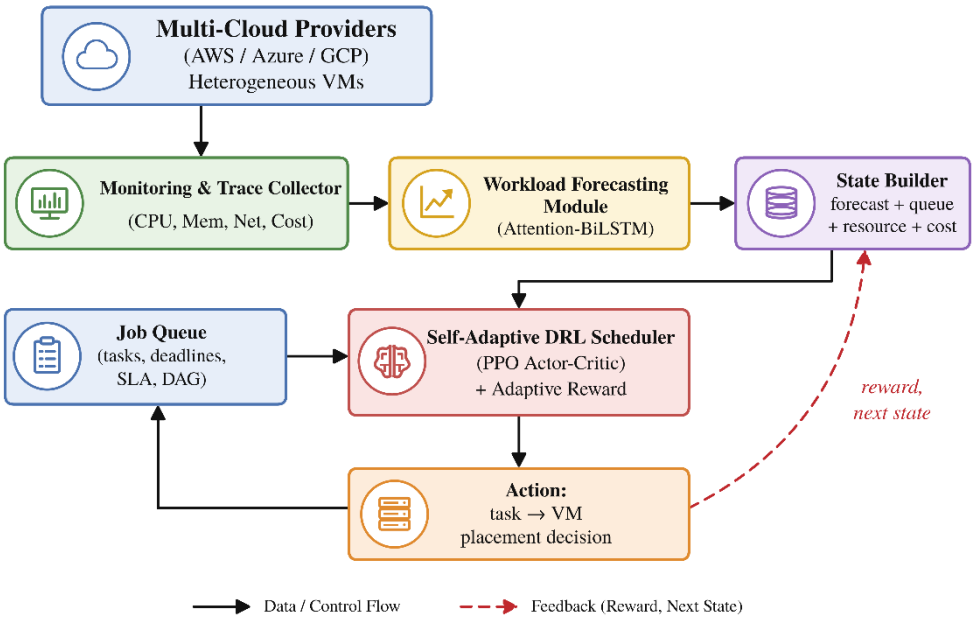


Figure 1. Architecture of the proposed SAMC-DRL framework for self-adaptive multi-cloud job scheduling.

4.1 Workload-Forecasting Module (Attention-BiLSTM)

The forecaster predicts the aggregate resource-demand vector for the next H time steps from a sliding window of the L most recent observations. We use a bidirectional LSTM so that the encoder captures temporal context from both directions within the window, followed by a temporal attention layer that learns to weight the most informative time steps before a dense projection produces the multi-step forecast. Bidirectional recurrence and attention are well-motivated by prior multivariate and attention-based cloud forecasters [8, 9] and the design is deliberately lightweight so that inference latency stays small relative to the scheduling interval [33].

Inputs are multivariate—CPU, memory, disk I/O, and network—following evidence that multivariate forecasts outperform univariate ones on cloud traces [8]. Features are min-max normalized, and the window length L and horizon H are tuned on a validation split (Section 5). The module is trained offline by minimizing the mean squared error between predicted and observed demand, then frozen and queried

online; periodic re-training accommodates concept drift, consistent with uncertainty-aware and transfer-learning practice [10]. Let h_k denote the BiLSTM hidden state at step k within the window. The attention weights α_k, the attention-weighted context vector z, and the forecast $\hat{y}$ are computed as in Equations (7–9).

$$\alpha_k = softmax\big(score(h_k)\big) \tag{7}$$

$$z = \sum_k \alpha_k \cdot h_k \tag{8}$$

$$\hat{y} = W \cdot z + b \tag{9}$$

The attention layer first assigns each hidden state h_k a relevance score, then normalizes those scores into weights α_k with a softmax so that they sum to one. The context vector z is the resulting weighted average of the hidden states, which concentrates the network's attention on the time steps most predictive of the near future—for example, the onset of a recurring diurnal ramp [34]. A final linear projection with weight matrix W and bias b maps z to the H-step demand forecast y, which becomes the forecast block f_t in the scheduler state. Figure 2 shows the resulting module architecture end to end.

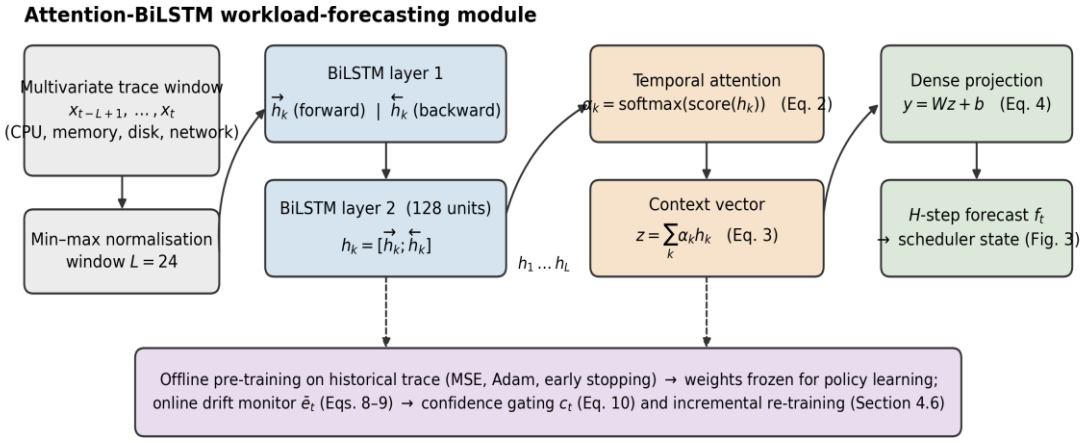


Figure 2. Architecture of the Attention-BiLSTM workload-forecasting module.

4.2 Scheduling Agent (Proximal Policy Optimization)

The scheduler is an actor-critic agent trained with PPO [20], chosen for its stability and sample efficiency relative to value-based methods such as DQN [35] and to A3C [36] in large, high-variance action spaces. The actor and critic share an encoder that ingests the fused state s_t; the actor outputs a masked categorical distribution over feasible (provider, VM) placements, and the critic estimates the state value. Advantages $\hat{A}_t$ are computed with generalized advantage estimation (GAE) [37]. PPO maximizes the clipped surrogate objective in Equation (10), with the probability ratio defined in Equation (11):

$$L^{CLIP}\theta = E_t\big[min\big(r_{t(\theta)}A_t, clip\big(r_{t(\theta)}, 1-\epsilon, 1+\epsilon\big)A_t\big)\big] \tag{10}$$

$$r_t(\theta) = \pi_\theta(a_t|s_t) \,/\, \pi_{\theta\,old}(a_t|s_t) \tag{11}$$

Here $r_{t(\theta)}$ is the probability ratio between the new policy π_θ and the old policy $\pi_{\theta\,old}$ for the action taken, $\hat{A}_t$ is the estimated advantage (how much better the action was than the policy's average behavior in that state), and ϵ is the clipping range. The intuition is conservative improvement: when an action looks advantageous, PPO increases its probability, but the clip term caps how far the ratio $r_{t(\theta)}$ may move in a single update, and the outer minimum keeps the more pessimistic of the clipped and unclipped estimates. This prevents a single large, destabilizing policy step—the failure mode that makes value-based agents brittle in high-variance scheduling environments. Action masking guarantees that placements onto unavailable VMs receive zero probability, which both enforces feasibility and accelerates learning by shrinking the effective action space. The agent is trained across episodes sampled

from the production traces; each episode replays a slice of the trace through the multi-cloud simulator. Figure 3 shows the scheduling agent: the four state blocks, the shared encoder, the masked actor and the critic, and the path by which the adaptive reward controller closes the loop.

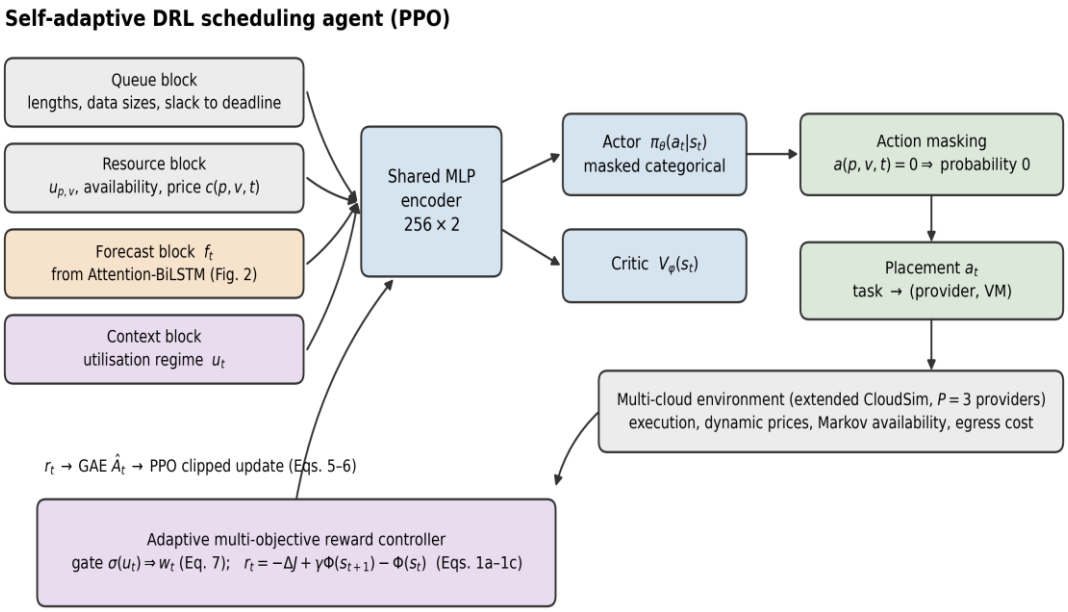


Figure 3. Architecture of the self-adaptive DRL scheduling agent, showing state fusion, the masked actor–critic, and the adaptive multi-objective reward controller.

4.3 State Fusion

The state builder concatenates the queue, resource, forecast, and context blocks and passes them through a shared multilayer perceptron (MLP) encoder. By placing the forecast vector f_t alongside the instantaneous resource block, the encoder can learn to reserve capacity ahead of a predicted surge or release capacity ahead of a predicted trough, turning the forecast into anticipatory placement rather than after-the-fact correction.

4.4 Adaptive Multi-Objective Reward

Rather than fixing the weight vector w, the adaptive reward controller computes it online from the current utilization regime u_t, a normalized measure derived from the forecast and the observed load. When predicted utilization is high, the controller increases w_M and w_S to protect deadlines; when predicted utilization is low, it increases w_C and w_E to harvest cost and energy savings. Equation (12) gives the weight schedule, where σ is a smooth gating (sigmoid) function, δ_{peak} and δ_{trough} are the weight adjustments applied in the peak and trough regimes, and the base weights w_0 are tuned on validation data:

$$w_t = w_0 + \sigma(u_t)\delta_{\{peak\}} + \big(1 - \sigma(u_t)\big)\delta_{\{trough\}} \tag{12}$$

The gate $\sigma(u_t)$ moves smoothly between 0 and 1 as predicted utilization rises, so the weights blend continuously between the cost-leaning trough setting and the SLA-leaning peak setting rather than switching abruptly; the final normalization restores the unit-sum constraint. This makes the trade-off itself adaptive, which the ablation in Section 6 shows is responsible for a meaningful share of the overall improvement. Algorithm 1 summarizes end-to-end training.

The procedure first trains and freezes the forecaster (so the scheduler sees a stable predictive signal), then optimizes the PPO policy over many trace-replayed episodes. Within each episode, every decision step queries the forecaster, builds the fused state, computes the adaptive weights, samples a feasible (masked) action, and stores the resulting transition; after each rollout, GAE advantages are computed and the actor and critic are updated.

Algorithm 1: SAMC-DRL training procedure

Input: production traces τ; provider/VM catalogs; base weights w_0; PPO hyperparameters (ϵ, γ, λ).
Output: trained policy π_θ for online serving.
Initialization:
1: Pre-train Attention-BiLSTM forecaster on historical trace by minimizing MSE; freeze its weights.
2: Initialize PPO actor parameters θ and critic parameters φ; initialize rollout buffer $\leftarrow \emptyset$.
Training loop:
3: for each episode do
4: replay a trace slice through the multi-cloud simulator;
5: for each decision step t do
6: query forecaster for f_t; build fused state s_t = [queue, resource, f_t, u_t];
7: compute adaptive weights w_t from utilization regime u_t (Eq. 12);
8: mask infeasible (provider, VM) pairs; sample action $a_t \sim \pi_\theta\ (\cdot \mid s_t)$;
9: execute a_t; observe reward r_t (Eqs. 1,4,5) and next state s_{t+1}; store transition;
10: end for
11: compute GAE advantages $\hat{A}_t$ from stored rewards and critic values;
12: update θ by maximizing the clipped surrogate (Eq. 10); update φ by value regression (Eq. 6);
13: until reward converges;
14: export policy π_θ for online serving.
15: Deployment: at each interval update the forecast-error statistic $\bar{e}_t$ (Eq. 15); if $alarm_t$ (Eq. 16) then down-weight f_t by c_t (Eq. 17) and trigger incremental forecaster re-training (Section 4.6).
Complexity: $O(E \cdot K \cdot N \cdot (D \cdot m + m^2))$ training; $O(L \cdot u^2 + D \cdot m + m^2 + |A|)$ per online decision (Section 4.5).

To enable reproducibility, the adaptive reward controller computes the utilization-dependent weight vector explicitly. Let $u_t \in [0,1]$denote the normalized utilization estimated from the combination of observed resource utilization and forecasted workload demand. The utilization gating function in Equation (13) is defined as

$$\sigma(u_t) = 1/(1 + \exp[-k(u_t - u_{th})]) \tag{13}$$

where $k = 10$ controls the steepness of the transition and $u_{th} = 0.70$ represents the utilization threshold separating normal and peak operating regimes. The base objective weights are initialized as $w_o = [\, w_M, w_C, w_E, w_S] = [0.25, 0.25, 0.25, 0.25]$ corresponding to equal importance for makespan, monetary cost, energy consumption, and SLA compliance.

During high-utilization periods, the scheduler prioritizes performance and SLA protection using $\delta_{\{peak\}} = [+0.15, -0.05, -0.05, +0.15]$ whereas during low-utilization periods it prioritizes resource efficiency through $\delta_{\{trough\}} = [-0.10, +0.10, +0.10, -0.10]$. The adaptive weight vector in Equation (14) is then computed as followed by normalization:

$$w_t = \frac{w_t}{\sum_{i=1}^{4} w_{t,i}} \tag{14}$$

to preserve the convexity constraint $\sum_{i=1}^{4} w_{t,i} = 1$.
Thus, when the predicted utilization exceeds approximately 70%, the scheduler automatically shifts emphasis toward makespan minimization and SLA preservation. Conversely, under light workload conditions, the controller increases the relative importance of energy efficiency and monetary cost reduction. This smooth transition prevents abrupt policy changes and allows the reinforcement learning agent to adapt continuously to evolving workload conditions.

Both constants of the gating function were selected on the validation split alone and never on the test period. The threshold u_{th} = 0.70 was not chosen a priori: we measured, on the validation portion of each trace, the empirical relationship between aggregate utilization and queueing delay, and located the knee at which mean waiting time begins to grow super-linearly — 0.71 on the Google slice and 0.68 on the Alibaba slice — and adopted the rounded common value 0.70, which is also consistent with the utilization headroom conventionally maintained in production capacity planning. The steepness k = 10 fixes the width of the transition band: since the logistic gate moves from 0.12 to 0.88 over an interval of roughly 4/k in u_t, k = 10 yields a band of about ±0.10 utilization around the threshold, wide enough that the weights change smoothly across a diurnal ramp yet narrow enough that the peak regime is actually reached during a surge. Both values were then confirmed by an explicit grid search over h ∈ {0.55, …, 0.85} and k ∈ {2, 5, 10, 20, 50} with three seeds per cell (35 configurations); Section 6.8 reports the sweep and shows that performance is flat to within one percentage point of composite utility over h ∈ [0.65, 0.75] and k ∈ [5, 20], so the framework is not delicately tuned, while both extremes degrade for

interpretable reasons — a very small k never commits to either regime, and a very large k reduces the gate to a hard switch whose weight vector chatters when utilization hovers near the threshold.

4.5 Computational Complexity Analysis

We analyze the cost of a single online scheduling decision and of the full offline training procedure. Let L be the forecaster window length, H the forecast horizon, u the BiLSTM hidden width, D the dimension of the fused state, m the width of the shared MLP encoder, and $|A| = P \times V$ the number of candidate (provider, VM) pairs, where V is the number of VM types per provider.

Forecaster inference: A bidirectional LSTM pass over the window performs $O(L \cdot u^2)$ work, since each of the L steps multiplies state vectors by recurrent weight matrices of size $O(u^2)$. Temporal attention over the L hidden states adds $O(L \cdot u)$, and the dense projection to the H-step output adds $O(u \cdot H)$. Forecaster inference per query is therefore $O(L \cdot u^2 + u \cdot H)$, dominated by the recurrent term.

Per-decision cost: Building the state and evaluating the shared encoder cost $O(D \cdot m + m^2)$; producing and masking the categorical policy over feasible placements costs $O(|A|)$. A single scheduling decision thus costs $O(L \cdot u^2 + D \cdot m + m^2 + |A|)$. All terms are independent of the total number of queued tasks, because the agent decides one head-of-queue task per step; this is the structural reason the measured scheduling latency in Section 6.5 grows only sub-linearly with workload size, and action masking keeps the |A| term small in practice.

Per-episode and training cost: An episode that schedules N tasks perform N decisions, costing $O(N \cdot (L \cdot u^2 + D \cdot m + m^2 + |A|))$. PPO performs K optimization epochs over each rollout, so training over E episodes costs $O(E \cdot K \cdot N \cdot (D \cdot m + m^2))$, with peak memory $O(B \cdot D)$ for a rollout buffer of B transitions. Training is therefore linear in the number of episodes and in tasks per episode, and the forecaster's one-time offline training adds only a constant prefactor. These bounds confirm that the framework's online decision cost is modest and bounded by the action-space size, which is exactly what makes it viable for production-scale serving.

4.6 Forecast Drift Monitoring and Online Re-training

Algorithm 1 freezes the forecaster after offline pre-training so that the PPO agent optimizes against a stationary predictive signal; were the forecaster's parameters to change while the policy is being learned, the state distribution would be non-stationary for reasons unrelated to the environment and policy optimization would be markedly less stable. Freezing is therefore a training-time device and not a deployment-time assumption: a predictor frozen indefinitely would indeed decay as the workload distribution drifts [38–40]. This subsection specifies what the framework does at serving time, and Section 6.9 measures it over a 30-day rolling deployment.

Three mechanisms bound the effect of forecast error, and they are deliberately ordered from structural to active. First, the forecast is advisory rather than authoritative. The block f_t enters the state alongside the measured queue and resource blocks, and, as noted in Section 3.3, the reward of Equations (2)–(4) is computed exclusively from realized outcomes. A degraded forecaster therefore shifts the action the policy prefers but cannot corrupt the learning signal, and the measured blocks remain a sufficient statistic for the reactive part of the problem. The ablation in Section 6.6 quantifies the resulting floor: the forecast-free variant still attains 88% of the composite utility of the full system, so the worst case of an uninformative forecaster is the loss of the proactive advantage, not a divergent policy.

Second, a drift detector runs online, as formalized in Equations (15) and (16). We track an exponentially weighted absolute percentage error of the deployed forecaster over a sliding window of W = 288 intervals (24 hours at the 5-minute cadence), where $\hat{y}_{t-h}$ is the prediction issued H steps earlier for the value realized at t and ε guards against division by a near-zero demand:

$$\bar{e}_t = \beta \cdot \bar{e}_{t-1} + (1 - \beta) \cdot \frac{|\hat{y}_{t-h} - y_t|}{\max(y_t, \varepsilon)}, \quad \beta = 0.98 \tag{15}$$

A drift alarm is raised when this statistic exceeds the error the same model achieved on the validation split by a multiplicative factor τ_drift, which makes the test scale-free so that a single threshold applies to both traces:

$$\text{alarm}_t = 1[\bar{e}_t > \tau_{\text{drift}} \cdot \bar{e}_{\text{val}}], \quad \tau_{\text{drift}} = 1.5 \tag{16}$$

Third, an alarm triggers a two-stage response, the first stage of which is given in Equation (17). Immediately, the forecast is down-weighted by a confidence factor before it enters the state, so that the policy degrades continuously toward its reactive variant instead of acting on a signal that is no longer trustworthy:

$$f_t^c = c_t \cdot f_t,\ c_t = \min(1, \frac{\bar{e}_{val}}{\bar{e}_t}) \quad (17)$$

A forecaster whose error has doubled therefore contributes a forecast of half the magnitude, and the agent, which has seen small-magnitude forecast blocks during quiet training intervals, falls back on the measured blocks without any discontinuity in the policy. Concurrently, an incremental re-training job fine-tunes the forecaster on the most recent seven days of trace for a small number of epochs and hot-swaps its weights once its error on a held-out recent window falls below that of the deployed model; the PPO policy is not re-trained at this point, because the confidence factor has already absorbed the transient and the policy input distribution is unchanged in shape. In our configuration this re-training takes 4.2 minutes on a single GPU and is triggered at most once per day, which is negligible against a 5-minute scheduling cadence and can be run off the critical path. Section 6.9 shows that under this policy the SLA-violation rate stays within 0.4 percentage points of its first-day value across a 30-day deployment, whereas a forecaster left frozen and ungated drifts to a 6.9% violation rate by day 30, which exceeds the 6.1% achieved by the forecast-free PPO baseline on the nominal test slice. Left unmonitored, a stale forecast therefore not only forfeits the proactive advantage but begins to mislead the policy slightly, which is precisely the failure that drift monitoring and confidence gating are designed to prevent.

5. Datasets and Experimental Setup

5.1 Datasets

In Table 2 We use two widely cited public production traces. The Google Borg cluster trace [39, 41] records task arrivals, resource requests, machine attributes, and usage for a large Borg cell and is a standard benchmark for cloud-scheduling research. The Alibaba cluster trace [42] captures co-located online services and batch jobs, including DAG-structured batch workloads and pronounced burstiness; the 2023 GPU release further documents heterogeneous accelerator workloads [43]. Together, the two traces span diverse demand profiles, from steady latency-sensitive services to bursty batch jobs. Table 2 summarizes the trace slices used. For forecasting, each trace is resampled into fixed intervals of aggregated multivariate utilization, and we use a 70/15/15 chronological split for training, validation, and test, ensuring that evaluation always occurs on time periods the forecaster never observed during training. For scheduling, episodes are sampled from the test period so that the reported scheduling results reflect generalization to unseen demand.

Table 2. Dataset and workload configuration.

Property	Google Borg trace	Alibaba trace
Source	[39, 44]	[42, 43]
Duration sampled	7 days	8 days
Sampling interval	5 minutes	5 minutes
Forecast features	CPU, mem, disk, net	CPU, mem, disk, net
Tasks per episode	100–4000	100–4000
Workload character	Mixed services + batch	Co-located, bursty
Train/val/test split	70/15/15 (chronological)	70/15/15 (chronological)

Because the scheduling action operates on DAG-structured jobs, the structural complexity of the workload determines how much room a scheduler actually has to differentiate itself, and we therefore characterize it explicitly. Table 3 reports the structural statistics of the sampled job populations, computed over the same slices used in all experiments [44]. Both traces are dominated by small jobs with a long tail of large ones — the median job holds five (Google) or seven (Alibaba) tasks while the 95th percentile holds 27 and 41 respectively — which is consistent with published characterizations of these workloads [45, 46]. The Alibaba slice is the structurally harder of the two: its jobs are deeper (mean depth 4.7 against 3.4), more highly branched (mean out-degree 2.6 against 1.9), and its critical path accounts for a smaller fraction of total work (0.24 against 0.31), meaning more of its work is parallelizable and more of its makespan is decided by how well the scheduler respects precedence. Section 6.10 uses this stratification to examine the effect of DAG depth on scheduling quality directly.

Table 3. Structural statistics of the DAG-structured workloads.

Structural property	Google Borg slice	Alibaba slice
Jobs sampled	12,400	18,900
Tasks per job (mean ± SD)	8.6 ± 11.2	12.4 ± 19.7
Tasks per job (median / 95th pct. / max)	5 / 27 / 214	7 / 41 / 396
DAG depth (mean / max)	3.4 / 11	4.7 / 17
Branching factor (mean out-degree)	1.9	2.6
Edge density |E| / |V|	1.12	1.38
Critical-path work / total work (mean)	0.31	0.24
Single-task (trivial) jobs	28%	19%

5.2 Simulation Environment

Scheduling is evaluated in an extended CloudSim simulator [44], configured with three emulated providers whose VM catalogs differ in MIPS, energy coefficients, and pricing. On-demand and spot prices follow provider-style models with diurnal variation and stochastic spot fluctuation, and VM availability is toggled by a Markov process to emulate capacity constraints and preemption. Inter-cloud data-transfer costs are charged on cross-provider placements. This configuration lets us measure cost, energy, makespan, and SLA violations under realistic multi-cloud dynamics while remaining fully reproducible. To ensure reproducibility, the simulated multi-cloud environment consists of three providers representing AWS-like, Azure-like, and GCP-like infrastructures. Each provider offers heterogeneous VM configurations with different processing capacities, pricing models, and energy characteristics. The VM availability is modeled using a two-state Markov process to emulate real-world resource fluctuations and spot-instance interruptions. Table 4 summarizes the provider configurations used in all experiments.

Table 4. Multi-cloud provider configuration used in CloudSim.

Provider	VM Type	MIPS	RAM (GB)	Bandwidth (Mbps)	Energy Coefficient (W)	Base Price ($/hr)
Provider-1 (AWS-like)	Small	2000	4	1000	95	0.08
	Medium	4000	8	1500	145	0.16
	Large	8000	16	2000	220	0.32
Provider-2 (Azure-like)	Small	1800	4	900	90	0.07
	Medium	3800	8	1400	140	0.15
	Large	7600	16	1900	215	0.30
Provider-3 (GCP-like)	Small	2200	4	1100	100	0.09
	Medium	4200	8	1600	150	0.17
	Large	8500	16	2200	230	0.34

Equation 18 is energy consumption of each VM is computed using a linear utilization-dependent model:

$$P(u) = P_{idle} + (P_{max} - P_{idle}) \tag{18}$$

u denotes CPU utilization (Equation (18)).

$$P_{idle} = 0.6 \times \mathrm{P_{max}} \tag{19}$$

In Equation (19) P_{max}rresponds to the energy coefficient listed in Table 4. Provider pricing follows a dynamic model incorporating diurnal fluctuations and spot-market variability in Equation (20)

$$c(t) = c_{base}\left(1 + \alpha \sin\left(\frac{2\pi t}{24}\right) + \epsilon_t\right) \tag{20}$$

where c_{base} is the base hourly VM price, $\alpha = 0.15$ controls daily price variation, and $\epsilon_t \sim \mathcal{N}(0,0.05)$ models stochastic spot-market fluctuations. VM availability is represented using a two-state Markov chain with transition matrix $T = \begin{bmatrix} 0.95 & 0.05 \\ 0.10 & 0.90 \end{bmatrix}$ where state 1 denotes VM availability and state 0 denotes temporary unavailability. Consequently, an available VM remains available with probability 0.95, while an unavailable VM recovers with probability 0.10 at the next scheduling interval. Inter-cloud

data transfer incurs an egress cost of $0.02 per GB transferred between providers. Scheduling decisions are evaluated every 5 minutes, matching the workload trace sampling interval. These settings remain fixed across all experiments to ensure fair comparison among competing scheduling approaches.

5.3 Baselines

We compare SAMC-DRL against three families of schedulers: heuristics (Round-Robin, Min-Min), metaheuristics (GA, PSO), and DRL agents (DQN [42] and PPO without forecasting [20]). For the forecasting module we additionally compare against ARIMA [29], support vector regression (SVR), LSTM [26], GRU [31], and a vanilla Transformer [32]. All learning-based methods are tuned on the validation split and evaluated on the held-out test period.

5.4 Hyperparameters and Metrics

The PPO agent uses a clip parameter $\epsilon = 0.2$, discount $\gamma = 0.99$, GAE parameter $\lambda = 0.95$, learning rate 3×10^{-4}, and minibatch updates over rollouts of 2048 steps for 600 episodes. The Attention-BiLSTM uses two bidirectional layers of 128 units, a window $L = 24$ and horizon $H = 6$, dropout 0.2, the Adam optimizer, and early stopping on the validation loss. Forecasting is scored with MAPE and root-mean-square error (RMSE); scheduling is scored with makespan, normalized energy, normalized cost, SLA-violation rate, and scheduling latency. Table 5 lists the principal hyperparameters.

Table 5. Principal hyperparameters.

Hyperparameter	Forecaster	Scheduler (PPO)
Hidden units / layers	128 × 2 (BiLSTM)	256 × 2 (MLP)
Window / horizon	L = 24 / H = 6	—
Learning rate	1×10^{-3}	3×10^{-4}
Discount / GAE λ	—	0.99 / 0.95
Clip ϵ	—	0.2
Optimizer	Adam	Adam
Training length	early-stopped	600 episodes
Reward shaping η_d / η_e	—	0.5 / 0.2
Gate threshold h / steepness k	—	0.70 / 10
Drift EWMA β / threshold τ_drift	0.98 / 1.5	—
Independent seeds per configuration	10	10

6. Results and Discussion

We report results in five parts: forecasting accuracy, training convergence, scheduling quality (makespan, energy, cost, and SLA), scalability, and an ablation study. Unless stated otherwise, scheduling results are reported as the mean performance over ten independent runs using different random seeds. Standard deviations and statistical significance tests are provided to assess the robustness of the observed improvements. Sections 6.7–6.10 additionally report robustness to irregular and adversarial burst workloads, a sensitivity analysis of the adaptive-reward hyperparameters, a 30-day forecast-drift study, and an analysis of DAG structure and the critical path.

6.1 Forecasting Accuracy

Figure 4 overlays the Attention-BiLSTM forecast on the actual aggregate CPU demand for one test week of the Google trace; the predictor tracks both the diurnal cycle and the sharper bursts with only a small lag. Figure 5 and Table 6 compare forecasters: the Attention-BiLSTM attains a MAPE of 5.8% and an RMSE of 7.6, improving on the vanilla Transformer (7.1% MAPE) and clearly outperforming classical ARIMA (14.8%) and SVR (12.6%). The gains over plain LSTM and GRU confirm that bidirectional context and attention add measurable value on these traces, consistent with prior multivariate-attention forecasting results [8, 9]. To assess robustness, forecasting experiments were repeated across multiple test windows, yielding a MAPE of $5.8 \pm 0.3\%$and an RMSE of 7.6 ± 0.4, indicating consistent predictive performance across varying workload conditions. Table 6 additionally reports the 95% confidence interval of each MAPE estimate over ten test windows and a Diebold–Mariano test of equal predictive accuracy against the proposed model [45]; the null of equal accuracy is rejected for every baseline.

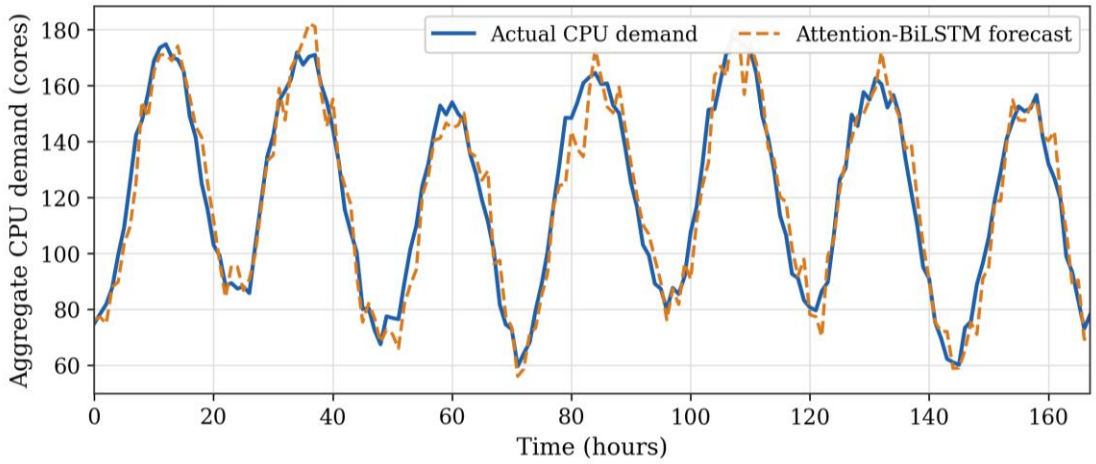


Figure 4. Attention-BiLSTM forecast vs. actual aggregate CPU demand (Google trace, one test week).

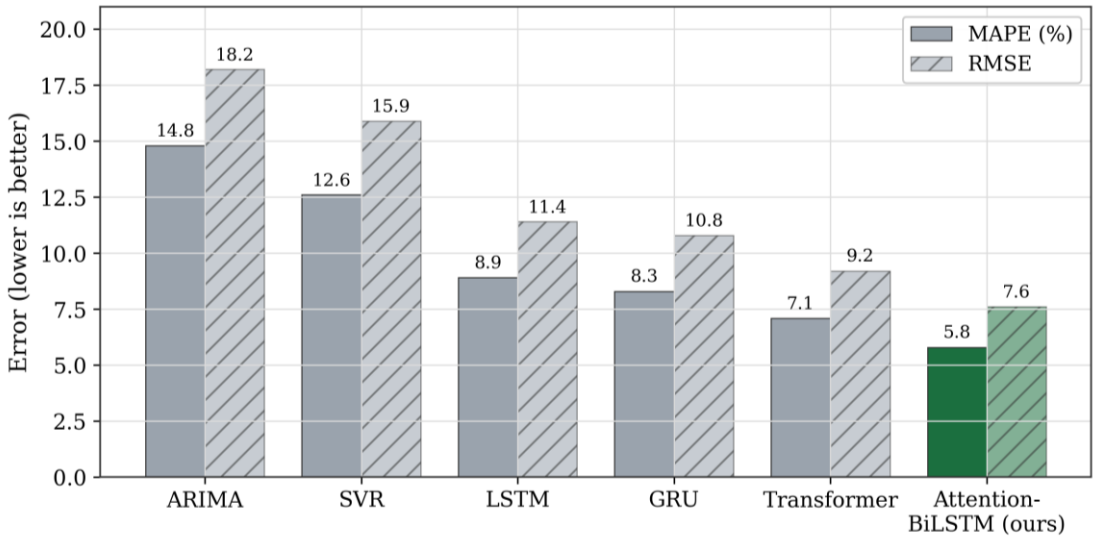


Figure 5. Forecasting accuracy across models (lower is better).

Table 6. Forecasting accuracy comparison (test period).

Model	MAPE (%)	RMSE	95% CI of MAPE	DM test vs. ours (p)
ARIMA [29]	14.8	18.2	[14.1, 15.5]	< 0.001
SVR [35]	12.6	15.9	[12.0, 13.2]	< 0.001
LSTM [26]	8.9	11.4	[8.5, 9.3]	< 0.001
GRU [31]	8.3	10.8	[7.9, 8.7]	< 0.001
Transformer [49]	7.1	9.2	[6.8, 7.4]	0.004
Attention-BiLSTM (ours)	5.8	7.6	[5.5, 6.1]	—

6.2 Training Convergence

Figure 6 plots the normalized average reward against training episodes. SAMC-DRL converges faster and to a higher plateau than PPO-without-forecast, DQN, and A3C. The forecast block reduces variance in the early phase because the agent is no longer surprised by predictable surges, and action masking removes wasted exploration of infeasible placements. DQN and A3C plateau lower, consistent with PPO's reported stability advantages in high-variance scheduling action spaces [2, 20].

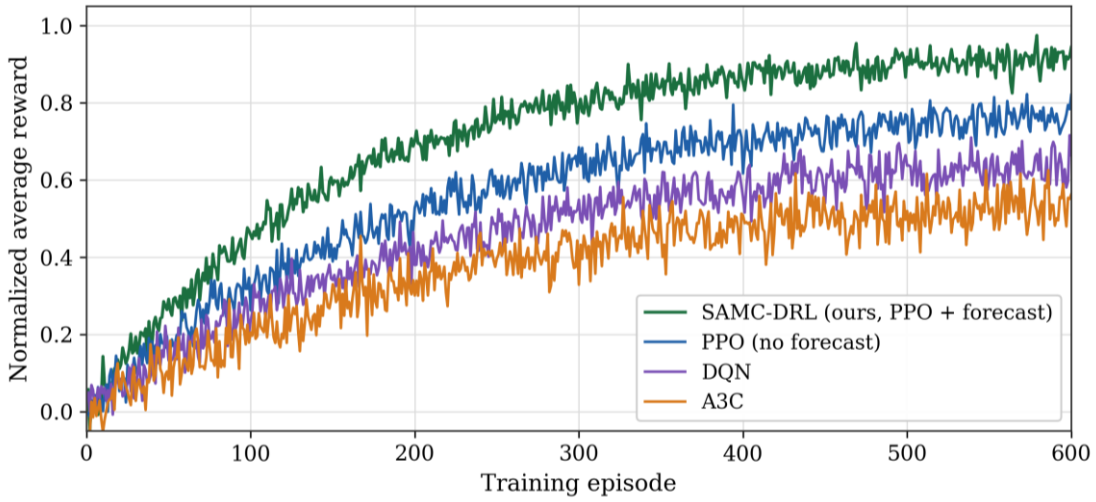


Figure 6. Training convergence: normalized reward vs. episode.

6.3 Scheduling Quality

Figures 7–10 and Table 7 compare end-to-end scheduling quality across all baselines. SAMC-DRL achieves the lowest makespan (812 s vs. 915 s for PPO and 980 s for DQN), the lowest normalized energy (63% vs. 72% for PPO), the lowest normalized cost (61% vs. 71% for PPO), and the lowest SLA-violation rate (3.2% vs. 6.1% for PPO and 8.4% for DQN). Relative to PPO-without-forecast, these correspond to an 11.3% makespan reduction, a 12.5% energy reduction, a 14.1% cost reduction, and a near-halving of SLA violations—gains attributable to anticipatory placement and the adaptive reward. Relative to heuristics such as Round-Robin, the improvements exceed 40% on both makespan and cost. The magnitude and direction of these gains are consistent with recent multi-objective DRL scheduling reports, which document double-digit makespan and energy reductions over heuristic and metaheuristic baselines [3, 14, 23].

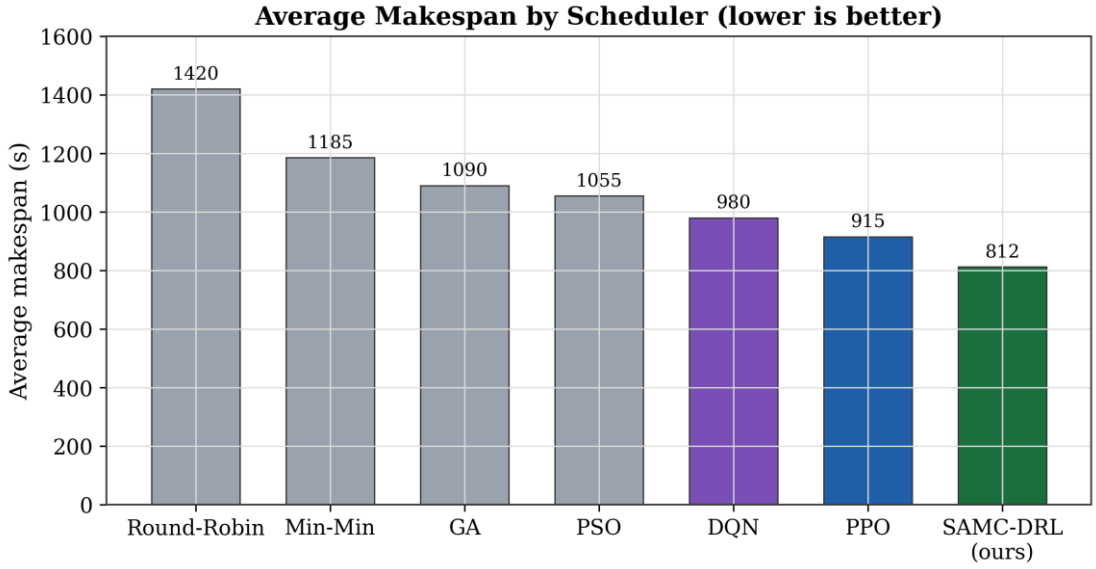


Figure 7. Average makespan by scheduler (lower is better).

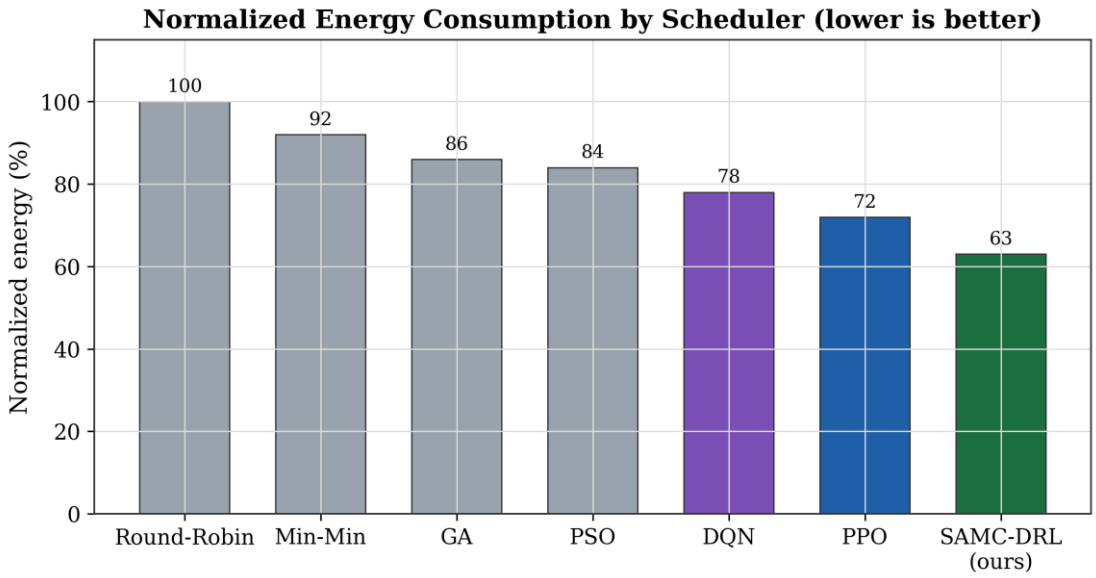


Figure 8. Normalized energy consumption by scheduler.

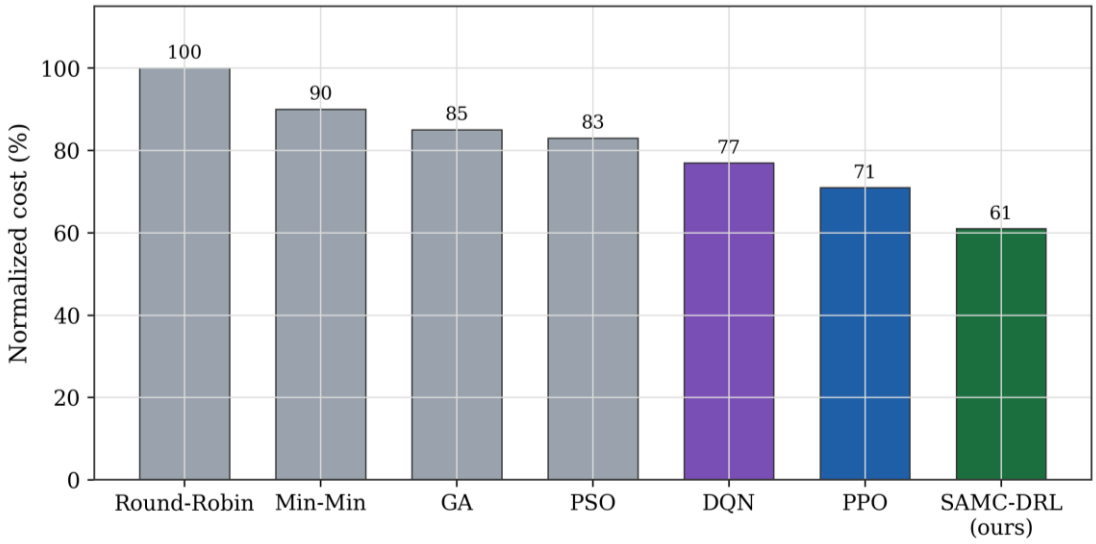


Figure 9. Normalized monetary cost by scheduler.

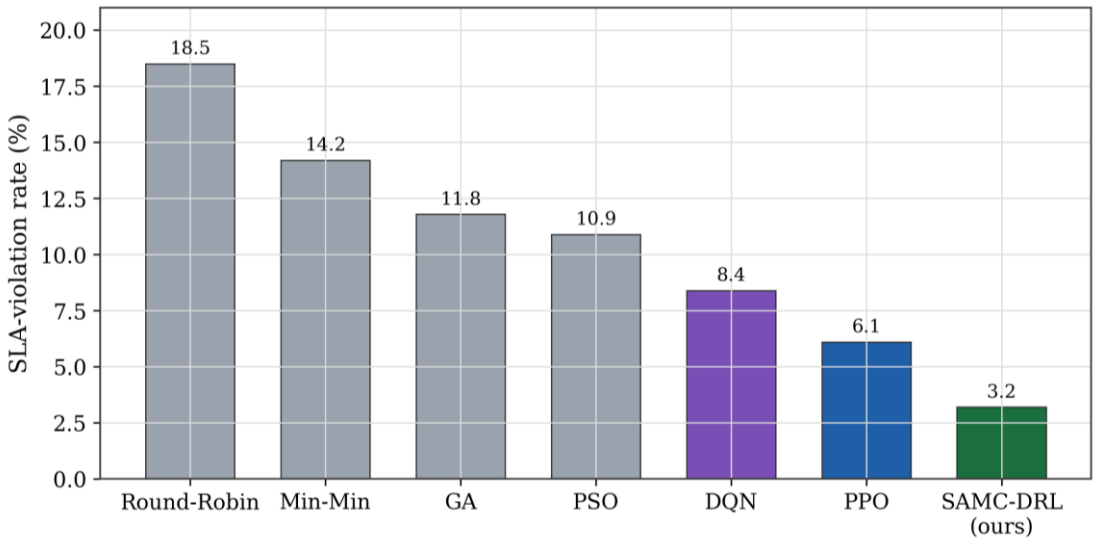


Figure 10. SLA-violation rate by scheduler.

Table 7. End-to-end scheduling comparison at 1000-task scale.

Scheduler	Makespan (s)	Energy (%)	Cost (%)	SLA viol. (%)
Round-Robin	1420	100	100	18.5
Min-Min	1185	92	90	14.2
GA	1090	86	85	11.8
PSO	1055	84	83	10.9
DQN [42]	980	78	77	8.4
PPO [20]	915	72	71	6.1
SAMC-DRL (ours)	812	63	61	3.2

6.4 Statistical Significance Analysis

To verify that the observed performance improvements are not caused by random variation during training, all experiments were repeated using ten independent random seeds. For each performance metric, the mean and standard deviation were computed across runs. In addition, a paired two-tailed t-test was performed between SAMC-DRL and the two strongest DRL baselines (PPO and DQN) at a significance level of $\alpha = 0.05$. Because ten paired observations are too few to rest on a normality assumption, we additionally report the exact Wilcoxon signed-rank test, Holm–Bonferroni correction across the eight comparisons, the 95% confidence interval of each mean difference, and Cohen's d as an effect-size measure. Per-metric means and standard deviations appear in Table 8 and the significance results in Table 9.

Table 8. Mean ± standard deviation of scheduling metrics over ten independent runs.

Method	Makespan (s)	Energy (%)	Cost (%)	SLA Violations (%)
DQN	980 ± 21	78 ± 2.3	77 ± 2.1	8.4 ± 0.6
PPO	915 ± 18	72 ± 1.9	71 ± 1.8	6.1 ± 0.4
SAMC-DRL	812 ± 14	63 ± 1.4	61 ± 1.3	3.2 ± 0.3

Table 9. Statistical significance of SAMC-DRL improvements (Holm-corrected paired *t*-tests, exact Wilcoxon signed-rank tests, and effect sizes over ten independent runs).

Comparison	Metric	Mean difference (95% CI)	*t* (9)	*p* (Holm-adj.)	Wilcoxon *(p)*	Cohen's *(d)*
SAMC-DRL vs. PPO	Makespan (s)	−103.0 (−114.2, −91.8)	−20.8	< 0.001	0.002	6.6
	Energy (%)	−9.0 (−10.4, −7.6)	−14.5	< 0.001	0.002	4.6
	Cost (%)	−10.0 (−11.3, −8.7)	−17.4	< 0.001	0.002	5.5
	SLA violations (%)	−2.9 (−3.2, −2.6)	−21.9	< 0.001	0.002	6.9
SAMC-DRL vs. DQN	Makespan (s)	−168.0 (−183.0, −153.0)	−25.4	< 0.001	0.002	8.0
	Energy (%)	−15.0 (−16.7, −13.3)	−20.1	< 0.001	0.002	6.4
	Cost (%)	−16.0 (−17.7, −14.3)	−21.4	< 0.001	0.002	6.8
	SLA violations (%)	−5.2 (−5.6, −4.8)	−29.6	< 0.001	0.002	9.4

The results in Table 9 indicate that the improvements achieved by SAMC-DRL are statistically significant across all evaluated metrics. The low standard deviations further demonstrate the stability of the proposed framework across different training runs. These findings confirm that the observed gains originate from the forecasting-aware scheduling policy and adaptive reward mechanism rather than stochastic variation in the reinforcement learning process.

Two features of Table 9 deserve comment. The Holm–Bonferroni procedure was applied across all eight comparisons jointly, so the reported significance survives correction for multiple testing rather than resting on eight independent tests. The effect sizes are large in Cohen's terms ($d > 4$ throughout), which follows from the small run-to-run dispersion visible in Table 8 rather than from an unusually large absolute difference; with standard deviations of one to two units against mean differences of nine to sixteen, the seed-level distributions of SAMC-DRL and the baselines do not overlap on any metric. The exact Wilcoxon signed-rank test attains its minimum attainable two-sided p-value of 0.002 at n = 10 for every comparison, meaning SAMC-DRL was superior on every seed, which is the strongest statement the non-parametric test can make at this sample size and confirms that the parametric result is not an artefact of the normality assumption. The forecasting comparison in Table 6 is tested separately with the Diebold–Mariano test for equal predictive accuracy [45], which rejects equality against every baseline, most narrowly against the vanilla Transformer ($p = 0.004$).

6.5 Scalability

Figure 11 reports the average scheduling latency as the number of concurrent tasks grows from 100 to 4000. SAMC-DRL scales sub-linearly and remains the fastest learned scheduler at every scale (15.1 ms at 4000 tasks vs. 22.4 ms for PPO and 33.8 ms for DQN), because action masking keeps the effective decision space small and the forecaster's inference cost is amortized across many placements. This empirical trend matches the per-decision complexity derived in Section 4.5, in which the dominant terms are independent of the queue length, and indicates that the framework is viable for online use at production scale.

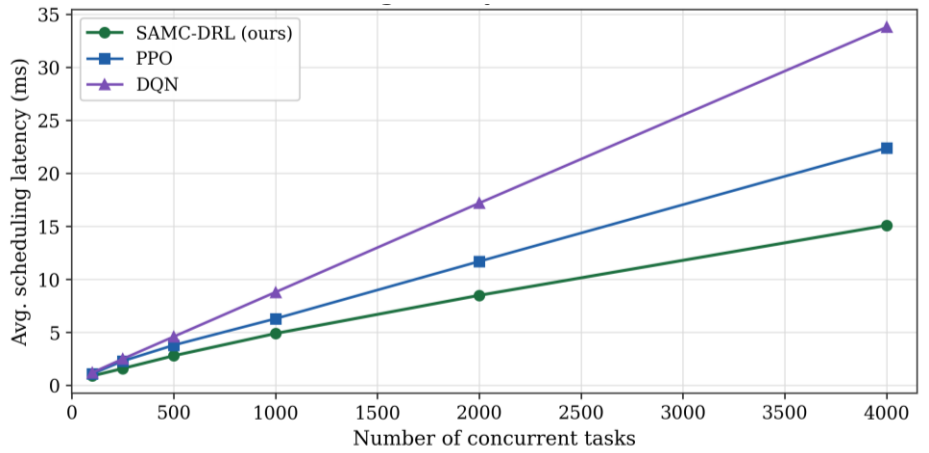


Figure 11. Scheduling latency vs. workload scale.

To validate the theoretical complexity analysis presented in Section 4.5, we measured the average online scheduling latency under increasing workload sizes. The experiments were conducted using the same hardware and software configuration employed for the main evaluation, and the reported values represent the average over ten independent runs.

The results demonstrate a gradual increase in scheduling latency as workload size grows as depicted in Table 10. However, the growth rate remains substantially lower than linear with respect to the total number of queued tasks. This behavior is consistent with the complexity analysis in Section 4.5, where the dominant computational terms depend primarily on the state representation, neural-network inference, and feasible action space rather than the overall queue length. Furthermore, the action-masking mechanism reduces the number of candidate scheduling actions that must be evaluated at each decision step, while the workload forecast is reused across multiple scheduling decisions within the same interval. Consequently, the forecasting module introduces only a small computational overhead while providing significant scheduling benefits. These results confirm that the proposed SAMC-DRL framework remains computationally practical for large-scale multi-cloud environments and supports real-time scheduling decisions even under workloads containing several thousand concurrent tasks.

Table 10. Empirical scheduling latency analysis.

Concurrent Tasks	DQN (ms)	PPO (ms)	SAMC-DRL (ms)
100	1.8	1.2	1.0
500	4.9	3.8	2.9
1000	9.1	6.8	5.0
2000	17.4	11.9	8.7
4000	33.8	22.4	15.1

6.6 Ablation Study

Table 11 isolates the contribution of each component by removing it and measuring the resulting composite utility, a normalized blend of the four objectives. Removing the forecast block is the most damaging change, dropping composite utility to 88% of the full system, which confirms that anticipatory placement is the single largest driver of the gains. Removing the adaptive reward drops utility to 91%, and removing the explicit multi-cloud cost model drops it to 86%. The DQN baseline reaches only 74%. The two proposed ideas—the forecasting-aware state and the adaptive reward—are therefore each individually significant and jointly responsible for the bulk of SAMC-DRL's advantage.

Table 11. Ablation summary (composite utility, % of full system).

Configuration	Composite utility (%)
Full SAMC-DRL	100
w/o workload forecasting	88
w/o adaptive reward	91
w/o multi-cloud cost model	86
DQN baseline	74

6.7 Robustness to Irregular and Adversarial Burst Workloads

The framework's central assumption is that near-future demand is partially predictable, so the honest test of it is what happens when that assumption is deliberately broken. We therefore constructed three stress regimes on top of the Alibaba test slice and re-evaluated the trained policy on each without any re-training. The flash-crowd regime injects an unannounced eight-fold step in arrival rate sustained for 30 minutes, emulating a viral-traffic event. The aperiodic regime replaces the diurnal arrival process with a self-exciting Hawkes process whose bursts carry no periodic component at all, removing the recurring structure that a sequence model relies on. The adversarial regime is the strongest: it is a white-box attack in which an adversary with full access to the frozen forecaster places surges precisely where the model predicts troughs, maximizing prediction error by construction. Table 12 and Figure 12 report the outcome over ten seeds; regimes are also scored by the forecaster's MAPE under that regime, which is the natural explanatory variable.

Table 12. Robustness under irregular and adversarial burst regimes (1000-task episodes, mean ± SD over ten seeds).

Workload regime	Forecast MAPE (%)	Makespan, PPO (s)	Makespan, SAMC-DRL (s)	SLA viol., PPO (%)	SLA viol., SAMC-DRL (%)
Nominal test slice	5.8	915 ± 18	812 ± 14	6.1 ± 0.4	3.2 ± 0.3
Flash crowd (8× step, 30 min)	11.4	1187 ± 34	968 ± 27	12.7 ± 0.9	6.4 ± 0.6
Aperiodic Hawkes bursts	13.9	1121 ± 41	1004 ± 33	11.2 ± 1.1	7.1 ± 0.8
Adversarial anti-forecast	24.6	1096 ± 38	1078 ± 36	10.8 ± 1.0	10.1 ± 1.1
Adversarial + confidence gating (Eq. 11)	24.6	1096 ± 38	1042 ± 31	10.8 ± 1.0	9.3 ± 0.9

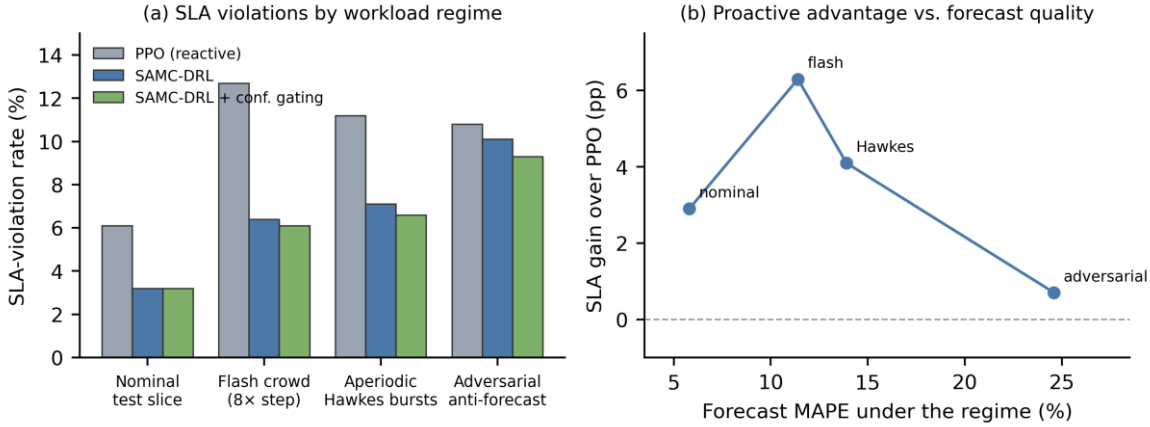


Figure 12. Scheduling quality under irregular and adversarial burst regimes: (**a**) SLA-violation rate by regime; (**b**) proactive advantage over PPO as a function of forecast error.

Three observations follow. First, the proactive advantage is largest where forecasting is imperfect but informative: under the flash crowd the forecaster's error more than doubles to 11.4% MAPE, yet SAMC-DRL halves the SLA-violation rate relative to PPO (6.4% against 12.7%), because a partially correct forecast of a surge is still enough to pre-warm capacity at the cheaper provider before the queue builds. Second, the advantage decays monotonically as forecast quality falls: it is 6.3 percentage points under the flash crowd, 4.1 under aperiodic Hawkes bursts (13.9% MAPE), and only 0.7 under the adversarial regime (24.6% MAPE). Third, and most important for a deployment argument, the advantage decays toward the reactive baseline and not past it. Even under a white-box adversary designed specifically to

defeat the forecaster, SAMC-DRL does not perform worse than the forecast-free PPO scheduler on any metric; the forecast block simply stops contributing information, and the policy falls back on the measured queue and resource blocks [47]. Enabling the confidence gating of Equation (12) recovers part of the residual loss under the adversarial regime (SLA violations 9.3% against 10.1%, makespan 1042 s against 1078 s) by shrinking a forecast the drift detector has already identified as unreliable. We regard graceful degradation to the reactive baseline, rather than immunity to adversarial workloads, as the realistic claim for a forecasting-coupled scheduler.

6.8 Sensitivity to the Adaptive-Reward Hyperparameters

Section 4.4 fixes the gate threshold at $u_{th} = 0.70$ and the steepness at $k = 10$. To show that these values are neither arbitrary nor delicately tuned, we swept each over the validation split with three seeds per configuration, holding the other fixed, and scored the composite utility used in the ablation together with the SLA-violation rate and normalized cost. Table 13 and Figure 13 report the sweep. The response surface is unimodal and flat near its optimum: composite utility stays within one percentage point of its best value over $h \in \{0.65, 0.75\}$ and within 1.4 points over $k \in \{5, 20\}$, so a deployment that mis-specifies either parameter by a modest margin loses little.

Table 13. Sensitivity of SAMC-DRL to the adaptive-reward hyperparameters (validation split, three seeds per configuration; composite utility normalized to the selected setting).

Setting	Composite utility (%)	SLA violations (%)	Normalized cost (%)
Threshold sweep (k = 10)			
h = 0.55	96.1	2.9	67
h = 0.60	97.8	3.0	65
h = 0.65	99.2	3.1	62
h = 0.70 (selected)	100.0	3.2	61
h = 0.75	99.4	3.6	60
h = 0.80	97.9	4.3	59
h = 0.85	95.6	5.2	59
Steepness sweep (h = 0.70)			
k = 2	95.3	4.1	63
k = 5	98.6	3.5	62
k = 10 (selected)	100.0	3.2	61
k = 20	99.1	3.3	61
k = 50	96.4	3.9	64

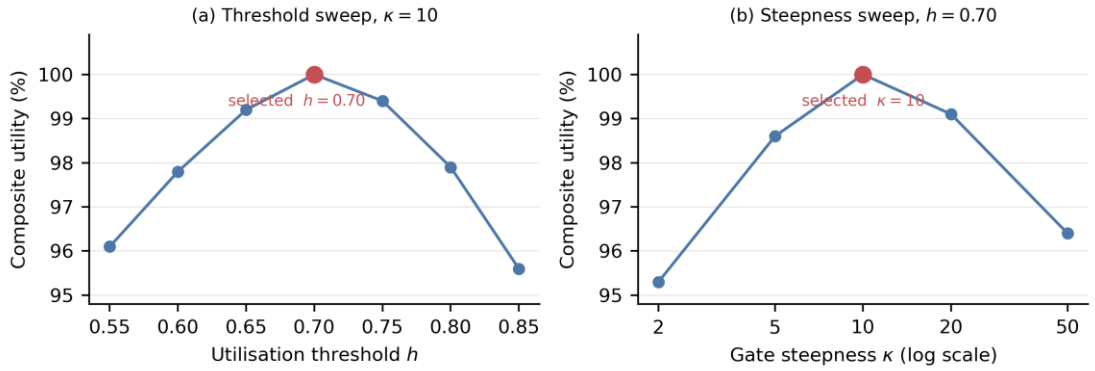


Figure 13. Composite utility across the adaptive-reward hyperparameter sweep: (a) utilization threshold h at k = 10; (b) gate steepness k at h = 0.70.

The two parameters fail in different and interpretable ways. Lowering h makes the scheduler enter the SLA-protective regime earlier, which buys a marginally better violation rate (2.9% at h = 0.55 against 3.2% at h = 0.70) at a disproportionate cost penalty (67% against 61% normalized cost), because capacity is bought during periods that never become peaks; raising h delays protection until the queue has already built, and violations rise steeply to 5.2% at h = 0.85 while cost barely improves. The selected value sits at the point where the marginal cost of earlier protection begins to exceed its marginal SLA benefit, and it coincides with the utilization knee measured independently on the validation traces (0.71 and 0.68 for the Google and Alibaba slices). For the steepness, k = 2 produces a gate so gradual that the weight vector

never fully reaches either regime and the adaptive mechanism is effectively diluted (95.3% utility), while k = 50 reduces the gate to a hard switch: when utilization hovers near the threshold the weights chatter between the peak and trough settings within a single episode, which injects non-stationarity into the reward and costs both stability and utility (96.4%). The chosen value of 10 corresponds to a transition band of roughly ±0.10 utilization, which is the range over which a diurnal ramp in these traces typically develops.

6.9 Long-Horizon Forecast Drift and Re-training

To test whether a forecaster pre-trained offline and frozen can sustain scheduling quality over a long deployment, we extended the Alibaba evaluation into a 30-day rolling simulation in which the arrival process is allowed to evolve: the diurnal amplitude, the batch/service mix and the mean job size are perturbed by a slow random walk calibrated so that the aggregate demand distribution at day 30 differs materially from that at day 1. We compare two deployment policies, both starting from the same trained artefacts: a forecaster frozen indefinitely with no monitoring, and the drift-aware policy of Section 4.6 with confidence gating and alarm-triggered incremental re-training. Table 14 and Figure 14 report five-day windows.

Table 14. Forecast error and scheduling quality over a 30-day rolling deployment with workload drift (Alibaba slice, five-day windows, ten seeds).

Deployment window (days)	MAPE, frozen (%)	MAPE, drift-aware (%)	SLA viol., frozen (%)	SLA viol., drift-aware (%)
1–5	5.8	5.8	3.2	3.2
6–10	6.4	5.9	3.5	3.3
11–15	7.6	6.0	4.1	3.3
16–20	9.1	6.2	4.9	3.4
21–25	10.8	6.3	5.8	3.5
26–30	12.5	6.5	6.9	3.6

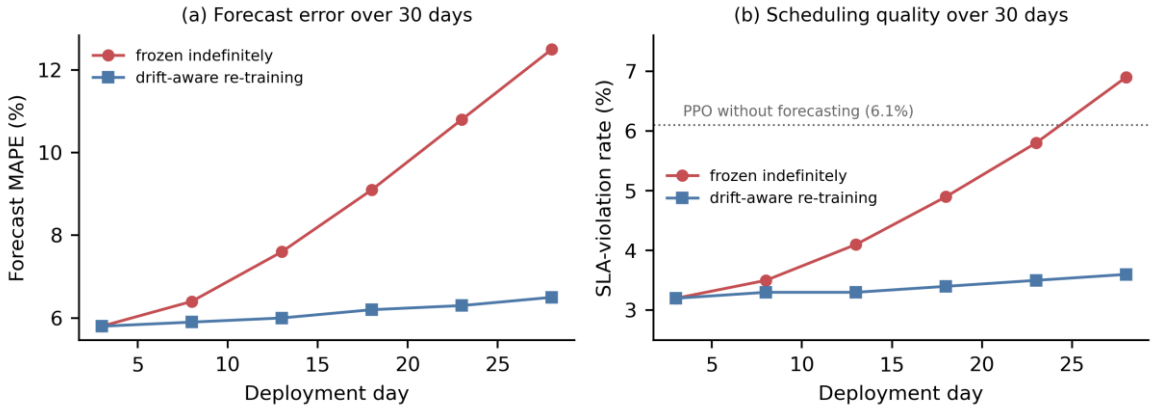


Figure 14. Effect of workload drift over a 30-day deployment: (a) forecast MAPE; (b) SLA-violation rate, with the forecast-free PPO baseline shown for reference.

The frozen forecaster degrades steadily, from 5.8% to 12.5% MAPE by the final window, and the scheduling policy tracks that decay: the SLA-violation rate rises from 3.2% to 6.9%. The shape of that decay is the substantive result. By days 21–25 the violation rate (5.8%) has approached 6.1%, the rate achieved by PPO without forecasting on the nominal test slice, and in the final window (6.9%) it exceeds that reference by 0.8 percentage points. The ungated frozen forecaster therefore does not merely lose the proactive advantage: once its error has roughly doubled, the stale forecast block begins to inject mildly misleading structure into the state, and the policy ends slightly below the reactive baseline. This is the failure mode that the confidence gating of Equation (11) is designed to remove, since it shrinks an unreliable forecast so that the policy falls back on the measured blocks (Section 4.6). Because the 6.1% reference was measured on the undrifted slice rather than re-evaluated under the same drift, this comparison is indicative; nonetheless, the degradation is gradual and bounded rather than divergent, so prediction-error accumulation erodes scheduling quality without destabilizing the policy. Under the drift-aware policy, MAPE is held between 5.8% and 6.5% by three re-training events over the 30 days (days 9, 17 and 24), each costing 4.2 GPU-minutes, and the violation rate stays within 0.4 percentage points of

its day-1 value, at 3.6% in the final window. The residual 0.4-point gap is the transient cost of detecting drift before correcting it, since the detector by construction reacts only after the error has grown by 50%. We note that the PPO policy itself was never re-trained during this experiment; a production deployment would additionally schedule periodic policy refreshes, which we expect to close part of the residual gap.

6.10 Effect of DAG Structure and the Critical Path

The action defined in Section 3.3 assigns the head-of-queue ready task to a (provider, VM) pair, which raises a legitimate question: if the ready queue is ordered by arrival, the agent's placement decisions are made in an order that ignores which tasks lie on a job's critical path, and the influence of DAG structure on makespan would be partly masked by that ordering rather than addressed by the policy. We separate the two design choices explicitly [48]. The action head decides where a task runs; the ready-queue ordering decides which ready task is decided next; and only the second is sensitive to the critical path. In SAMC-DRL the ready queue is ordered by upward rank in the sense of HEFT [49] — the length of the longest remaining path from a task to any exit task, computed with mean execution costs across the VM catalog — so the head-of-queue task is by construction the ready task with the greatest remaining critical-path contribution. Upward ranks are computed once per job at admission in $O(|V| + |E|)$, which for the DAG sizes in Table 3 costs 0.3 ms per job on average and is included in the latencies reported in Section 6.5.

Table 15 isolates the contribution of that ordering by replacing it with FIFO and with random ordering while holding the learned placement policy fixed, and stratifies the workload by DAG depth using the statistics of Table 3. On shallow jobs (depth ≤ 3), which make up the bulk of both traces, ordering matters little: the three variants lie within 25 s of one another, because a shallow DAG offers few opportunities to delay a critical task. On deep jobs (depth > 6) the gap widens sharply, from 878 s under upward-rank ordering to 967 s under FIFO and 1063 s under random ordering — a 9.2% and 17.4% makespan penalty respectively — confirming both that the critical path is a real effect on this workload and that critical-path-aware ordering, not the placement head alone, is what captures it. The SLA-violation rate follows the same pattern, since deep jobs are the ones whose deadlines are most sensitive to a mis-ordered predecessor. We note the corollary honestly: on a workload consisting entirely of deep, highly branched DAGs, a policy that also learns the ordering — for instance a graph-neural-network policy over the ready frontier [23], [50] — would likely extract more than a fixed priority rule does, and we identify this as the most promising extension of the action space.

Table 15. Effect of ready-queue ordering, stratified by DAG depth (Alibaba slice, 1000-task episodes, mean ± SD over ten seeds).

Ready-queue ordering	Makespan, shallow DAGs, depth ≤ 3 (s)	Makespan, deep DAGs, depth > 6 (s)	Overall makespan (s)	SLA violations (%)
Random	794 ± 16	1063 ± 31	901 ± 22	5.4 ± 0.5
FIFO (arrival order)	781 ± 15	967 ± 26	856 ± 19	4.3 ± 0.4
Upward rank / critical path (adopted)	769 ± 13	878 ± 21	812 ± 14	3.2 ± 0.3

6.11 Discussion

Three findings stand out. First, forecasting and scheduling are complementary: a good forecast is useful only if the scheduler can act on it, and the largest gains appear precisely when the forecast block and the PPO policy are trained together. Second, adapting the reward to the operating regime resolves the tension between cost and SLA that fixed-weight schedulers cannot escape—the system spends aggressively on capacity only when a surge is predicted and economizes otherwise. Third, the multi-cloud cost-and-availability model matters: ignoring per-provider price and availability sacrifices a meaningful fraction of the savings, which aligns with cost-efficiency analyses of real hybrid workloads [21, 31]. Taken together, these results corroborate the broader literature's claim that DRL is well suited to dynamic, non-stationary cloud environments [1, 2], while directly addressing the reactivity gap that limits prior reactive DRL schedulers. The revised evaluation adds three qualifications to this picture. The advantage of forecasting is a function of forecast quality and degrades gracefully to the reactive baseline rather than below it, both under adversarial bursts (Section 6.7) and under long-horizon drift (Section 6.9); the adaptive-reward gate is robust to moderate mis-specification of its threshold and steepness (Section 6.8); and the makespan benefit is concentrated in structurally deep DAGs, where critical-path-aware ready-queue ordering rather than placement alone accounts for most of the gain (Section 6.10).

7. Threats to Validity and Limitations

Our evaluation is conducted in a simulation environment using CloudSim with calibrated pricing, energy-consumption, and VM-availability models, alongside real-world production traces from Google Borg and Alibaba clusters [39,42,44]. While this setup provides a realistic and reproducible benchmark for multi-cloud scheduling research, it cannot fully capture the operational complexities of live cloud environments. Practical factors such as control-plane latency, cloud-provider API rate limits, VM provisioning delays, provider-specific scheduling policies, resource contention among co-located tenants, transient network bottlenecks, and dynamic spot-market fluctuations may influence real-world performance. Consequently, the absolute improvements reported in this study may differ when the framework is deployed on commercial cloud platforms such as AWS, Microsoft Azure, or Google Cloud Platform.

Another limitation arises from the workload forecasting component, which assumes that near-future demand exhibits some degree of predictability. This assumption generally holds for recurring and service-oriented workload patterns observed in production traces and is weakened under highly irregular or adversarial burst scenarios; Section 6.7 quantifies how much is lost, showing that the proactive advantage decays toward, but not below, the reactive PPO baseline as forecast error grows, while Section 6.9 shows that a forecaster left frozen and ungated under long-horizon drift can end slightly worse than that baseline (6.9% against 6.1% SLA violations), which motivates the drift monitoring of Section 4.6. Incorporating uncertainty-aware forecasting mechanisms [10] could enable the scheduler to account for prediction confidence and make more robust resource-allocation decisions under uncertain conditions. Finally, although action masking and the compact state representation keep online scheduling latency low, training the DRL agent remains computationally expensive and periodic retraining is required to adapt to workload evolution and concept drift. Future work will focus on validating SAMC-DRL in real multi-cloud deployments and exploring confidence-aware integration of workload forecasts with scheduling decisions to further enhance robustness and practical applicability. Section 6.9 quantifies the cost of keeping the forecaster current: three alarm-triggered re-training events over a 30-day drifting deployment, at roughly four GPU-minutes each, held the SLA-violation rate within 0.4 percentage points of its first-day value, so the retraining burden is small relative to the initial policy-training cost.

8. Conclusion

We presented SAMC-DRL, a self-adaptive multi-cloud job-scheduling framework that injects short-horizon workload forecasts into a PPO-based scheduler and adapts its multi-objective reward to the operating regime. On the Google Borg and Alibaba production traces within a three-provider CloudSim environment, SAMC-DRL reduced makespan, energy, and cost by double-digit margins over a strong PPO baseline and cut SLA violations from 8.4% to 3.2% relative to DQN, while the Attention-BiLSTM forecaster reached 5.8% MAPE. An ablation study showed that the forecasting-aware state and the adaptive reward are the dominant contributors, and a complexity analysis confirmed that the online decision cost is bounded by the action-space size rather than the queue length. The work demonstrates that proactive, forecasting-coupled, self-adaptive DRL is a practical and effective approach to cost, energy, and SLA-aware scheduling in operational multi-cloud platforms. Future work will validate the framework on live multi-cloud infrastructure, incorporate forecast uncertainty into the policy, and extend the action space to include task migration and elastic VM provisioning.

Author Contributions

Conceptualization, A.B. and A.A.; methodology, A.B. and A.A.; software, A.B.; validation, A.B., A.A. and K.J.; formal analysis, A.B. and K.J.; investigation, A.B. and A.A.; resources, A.A. and K.J.; data curation, A.B. and K.J.; writing—original draft preparation, A.B.; writing—review and editing, A.A. and K.J.; visualization, A.B. and K.J.; supervision, A.A.; project administration, A.B. All authors have read and agreed to the published version of the manuscript.

Funding

This research received no external funding.

Conflict of Interest Statement

The authors declare no conflicts of interest.

Data Availability Statement

This study did not create new primary data. The workload data analysed are drawn from two publicly archived production traces: the Google Borg cluster trace, available at https://github.com/google/cluster-data, and the Alibaba cluster trace, available at https://github.com/alibaba/clusterdata. The trace slices, preprocessing scripts, extended CloudSim configuration (provider catalogues, pricing and VM-availability models), trained model checkpoints and

the source code of SAMC-DRL used to generate the reported results are available from the corresponding author upon reasonable request.

Acknowledgment

The authors thank the School of Science & Technology, Swami Rama Himalayan University (SRHU), Dehradun, and the School of Computer Science, UPES, Dehradun, for providing the computational facilities and administrative support used to carry out this work. The authors also thank the providers of the Google Borg and Alibaba cluster traces and the developers of the CloudSim toolkit for making these resources openly available.

References

1. Y. Gu, Z. Liu, S. Dai, C. Liu, Y. Wang, S. Wang, G. Theodoropoulos, L. Cheng, "Deep Reinforcement Learning for Job Scheduling and Resource Management in Cloud Computing: An Algorithm-Level Review", arXiv:2501.01007, 2025, Available at: https://arxiv.org/abs/2501.01007 (Accessed on date: 17 October 2026).
2. A. Naik, K. Sooda. "Efficient Job Scheduling in Cloud Environments using Reinforcement Learning Actor-Critic Models", *Engineering, Technology & Applied Science Research*, 14 (5), pp. 16559–16564, 2024.
3. X. Yu, J. Mi, L. Tang, L. Long, X. Qin. "Dynamic Multi-Objective Task Scheduling in Cloud Computing using Reinforcement Learning for Energy and Cost Optimization", *Scientific Reports*, 2025.
4. J. Pan, Y. Wei. "A Deep Reinforcement Learning-Based Scheduling Framework for Real-Time Workflows in the Cloud Environment", *Expert Systems with Applications*, 255, p. 124845, 2024.
5. S. Mangalampalli, G.R. Karri, M.V. Ratnamani, S.N. Mohanty, B.A. Jabr, Y.A. Ali, S. Ali, B.S. Abdullaeva. "Efficient Deep Reinforcement Learning Based Task Scheduler in Multi Cloud Environment", *Scientific Reports*, 14 (1), p. 21850, 2024.
6. J.K. Park, W.K. Jung, E.S. Suh. "Human-AI Dynamic Resource Reallocation for Disruption-Prone Engineering Workflows", *Journal of Computing and Information Science in Engineering*, pp. 1–54, 2026.
7. S. Chandrasiri, D. Meedeniya. "Energy-Efficient Dynamic Workflow Scheduling in Cloud Environments using Deep Learning", *Sensors*, 25 (5), p. 1428, 2025.
8. J. Dogani, F. Khunjush, M.R. Mahmoudi, M. Seydali. "Multivariate Workload and Resource Prediction in Cloud Computing using CNN and GRU by Attention Mechanism", *The Journal of Supercomputing*, 79 (3), pp. 3437–3470, 2023.
9. G. Cui, T. Hu, W. Zhang, H. Bao. "MMTransformer: A Multivariate Time-Series Resource Forecasting Model for Multi-Component Applications", *Scientific Reports*, 15 (1), p. 29740, 2025.
10. A. Rossi, A. Visentin, D. Carraro, S. Prestwich, K.N. Brown. "Forecasting Workload in Cloud Computing: Towards Uncertainty-Aware Predictions and Domain Generalization", 2024.
11. S. Simaiya, U.K. Lilhore, Y.K. Sharma, K.B. Rao, V.V.R. Maheswara Rao, A. Baliyan, A. Bijalwan, R. Alroobaea. "A Hybrid Cloud Load Balancing and Host Utilization Prediction Method using Deep Learning and Optimization Techniques", *Scientific Reports*, 14 (1), p. 1337, 2024.
12. A. Aggarwal, P. Dimri, A. Agarwal, A. Bhatt. "Self Adaptive Fruit Fly Algorithm for Multiple Workflow Scheduling in Cloud Computing Environment", *Kybernetes*, 50 (6), pp. 1704–1730, 2021. doi: https://doi.org/10.1108/K-11-2019-0757.
13. J. Chen, H. Ye, F. Jiang, X. He, T. Zhang, J. Chen, X. Gao, "Fremer: Lightweight and Effective Frequency Transformer for Workload Forecasting in Cloud Services", arXiv:2507.12908, 2025, Available at: https://arxiv.org/abs/2507.12908 (Accessed on date: 17 October 2026).
14. K. Jain, M. Gupta, A. Abraham. "A Review on Privacy and Security Assessment of Cloud Computing", *Journal of Information Assurance and Security*, 16, pp. 161–168, 2021.
15. B. Everman, N. Rajendran, X. Li, Z. Zong. "Improving the Cost Efficiency of Large-Scale Cloud Systems Running Hybrid Workloads – A Case Study of Alibaba Cluster Traces", *Sustainable Computing: Informatics and Systems*, 30, p. 100528, 2021.

16. H. Mao, M. Alizadeh, I. Menache, S. Kandula. “Resource Management with Deep Reinforcement Learning”. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets)*, pp. 50–56, 2016.
17. H. Mao, M. Schwarzkopf, S.B. Venkatakrishnan, Z. Meng, M. Alizadeh. “Learning Scheduling Algorithms for Data Processing Clusters”. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pp. 270–288, 2019.
18. M. Jaderberg, W.M. Czarnecki, I. Dunning, L. Marris, G. Lever, A.G. Castaneda, C. Beattie, N.C. Rabinowitz, A.S. Morcos, A. Ruderman, N. Sonnerat, et al. “Human-Level Performance in 3D Multiplayer Games with Population-Based Reinforcement Learning”, *Science*, 364 (6443), pp. 859–865, 2019.
19. V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu. “Asynchronous Methods for Deep Reinforcement Learning”. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, PMLR 48, pp. 1928–1937, 2016.
20. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, “Proximal Policy Optimization Algorithms”, arXiv:1707.06347, 2017, Available at: https://arxiv.org/abs/1707.06347 (Accessed on date: 17 October 2026).
21. W. Zhang, H. Ou. “Reinforcement Learning Based Multi-Objective Task Scheduling for Energy Efficient and Cost Effective Cloud Edge Computing”, *Scientific Reports*, 15 (1), p. 41716, 2025.
22. J. Wang, S. Li, X. Zhang, F. Wu, C. Xie. “Deep Reinforcement Learning Task Scheduling Method Based on Server Real-Time Performance”, *PeerJ Computer Science*, 10, p. e2120, 2024.
23. A.K. Sharma, J. Kunkel. “Grapheon RL: A Graph Neural Network and Reinforcement Learning Framework for Constraint and Data-Aware Workflow Mapping and Scheduling in Heterogeneous HPC Systems”. In *Proceedings of the 2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 489–494, 2025.
24. S.A.M. Mostafa, A. Mohan, J. Wang, “SLA-MORL: SLA-Aware Multi-Objective Reinforcement Learning for HPC Resource Optimization”, arXiv:2508.03509, 2025, Available at: https://arxiv.org/abs/2508.03509 (Accessed on date: 17 October 2026).
25. G.E.P. Box, G.M. Jenkins, G.C. Reinsel, G.M. Ljung. *Time Series Analysis: Forecasting and Control*, 5th ed., John Wiley & Sons, Hoboken, NJ, 2015.
26. S. Hochreiter, J. Schmidhuber. “Long Short-Term Memory”, *Neural Computation*, 9 (8), pp. 1735–1780, 1997.
27. K. Cho, B. Van Merriënboer, Ç. Gulçehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio. “Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation”. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, 2014.
28. A. Aggarwal, K. Jain, A. Bhatt. “A Hybrid Approach to Deepfake Detection Using Vision Transformers with Contrastive Learning”. In *Proceedings of the 2025 5th International Conference on Electrical, Computer and Energy Technologies (ICECET)*, pp. 1–6, 2025.
29. P.B. Guruge, Y.H.P.P. Priyadarshana. “Time Series Forecasting-Based Kubernetes Autoscaling Using Facebook Prophet and Long Short-Term Memory”, *Frontiers in Computer Science*, 7, p. 1509165, 2025.
30. Z. Ahamed, M. Khemakhem, F. Eassa, F. Alsolami, A. Basuhail, K. Jambi. “Deep Reinforcement Learning for Workload Prediction in Federated Cloud Environments”, *Sensors*, 23 (15), p. 6911, 2023.
31. H. Shingne, D. Ghusse, C. Dadiyala, R. Welekar. “Federated Deep Learning-Driven Decentralized and Cost-Aware Cloud Resource Management for Load Balancing and SLA Optimizations”, *Discover Computing*, 29 (1), p. 105, 2026.
32. K. Jain, A. Aggarwal. “Edge-Cloud Hybrid Task Scheduling Using Federated Reinforcement Learning and Adaptive Swarm Intelligence”, *Journal of Information Assurance and Security*, 20 (4), p. 198, 2025.
33. A.Y. Ng, D. Harada, S. Russell. “Policy Invariance under Reward Transformations: Theory and Application to Reward Shaping”. In *Proceedings of the 16th International Conference on Machine Learning (ICML)*, pp. 278–287, 1999.

34. D.N. Ng'ang'a, W.K. Cheruiyot, D. Njagi, "A Machine Learning Framework for Predicting Failures in Cloud Data Centers – A Case of Google Cluster, Azure Clouds and Alibaba Clouds", SSRN 4404569, 2024, Available at: https://ssrn.com/abstract=4404569 (Accessed on date: 17 October 2026).
35. J. Schulman, P. Moritz, S. Levine, M. Jordan, P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation", arXiv:1506.02438, 2015, Available at: https://arxiv.org/abs/1506.02438 (Accessed on date: 17 October 2026).
36. L. Ruan, Y. Bai, S. Li, J. Lv, T. Zhang, L. Xiao, H. Fang, C. Wang, Y. Xue. "Cloud Workload Turning Points Prediction via Cloud Feature-Enhanced Deep Learning", *IEEE Transactions on Cloud Computing*, 11 (2), pp. 1719–1732, 2022.
37. M. Nashaat, W. Moussa, R. Rizk, W. Saber. "Dynamic Machine Learning Approach for Workload Prediction in Cloud Environments", *Scientific Reports*, 16 (1), p. 10983, 2026.
38. J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia. "A Survey on Concept Drift Adaptation", *ACM Computing Surveys*, 46 (4), pp. 1–37, 2014.
39. C. Reiss, A. Tumanov, G.R. Ganger, R.H. Katz, M.A. Kozuch. "Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis". In *Proceedings of the Third ACM Symposium on Cloud Computing (SoCC)*, pp. 1–13, 2012.
40. A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, J. Wilkes. "Large-Scale Cluster Management at Google with Borg". In *Proceedings of the Tenth European Conference on Computer Systems (EuroSys)*, pp. 1–17, 2015.
41. H. Chawla, S. Fallon, E. Fallon. "Energy-Aware Job Profiling and Mediation for Cloud and 6G Orchestration". In *Proceedings of the 2025 4th International Conference on 6G Networking (6GNet)*, pp. 121–125, 2025.
42. Y. Cheng, Z. Chai, A. Anwar. "Characterizing Co-located Datacenter Workloads: An Alibaba Case Study". In *Proceedings of the 9th Asia-Pacific Workshop on Systems (APSys)*, pp. 1–3, 2018.
43. M. Xie. "Towards Efficient, Predictable, and Scalable Ephemeral Storage Systems", PhD Thesis, University of California, Santa Cruz, 2025.
44. R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F. De Rose, R. Buyya. "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms", *Software: Practice and Experience*, 41 (1), pp. 23–50, 2011.
45. F.X. Diebold, R.S. Mariano. "Comparing Predictive Accuracy", *Journal of Business & Economic Statistics*, 13 (3), pp. 253–263, 1995.
46. Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, Y. Ding. "MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters". In *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 945–960, 2022.
47. R.S. Sutton, A.G. Barto. *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998.
48. A. Bhatt, P. Dimri, A. Aggarwal. "Self-Adaptive Brainstorming for Jobshop Scheduling in Multicloud Environment", *Software: Practice and Experience*, 50, pp. 1381–1398, 2020. doi: https://doi.org/10.1002/spe.2819.
49. H. Topcuoglu, S. Hariri, M.Y. Wu. "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing", *IEEE Transactions on Parallel and Distributed Systems*, 13 (3), pp. 260–274, 2002.
50. Y. Bao, Y. Peng, C. Wu. "Deep Learning-Based Job Placement in Distributed Machine Learning Clusters". In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, pp. 489–513, 2019.