# When Does Deep RL Beat Calibrated Baselines? A Benchmark Study on Adaptive Resource Control

**Guilin Zhang** [1 2] **Chuanyi Sun** [2] **Kai Zhao** [1] **Shahryar Sarkani** [2] **John Fossaceca** [2]

## Abstract

A properly calibrated rule-based autoscaler can beat every one of six mainstream deep reinforcement learning (DRL) algorithms on cost across every workload we test—so when, if ever, does DRL actually help? We study this in RLSCALEBENCH, a reproducible benchmark and evaluation protocol for DRL on adaptive resource control, where an agent allocates compute to a dynamic workload under cost and service-level constraints. The literature reports conflicting claims about whether model-free RL outperforms well-tuned rule-based controllers, with single-seed runs, uncalibrated baselines, and inconsistent training budgets confounding cross-study comparison. We evaluate PPO, DQN, A2C, SAC, TD3, and DDPG under matched network architectures, training budgets, and reward functions against a properly calibrated rule-based baseline across six workload patterns and five seeds (240 runs), instantiate the benchmark on Kubernetes Horizontal Pod Autoscaling, and probe distribution-shift generalization by training on one workload and deploying on five shifted distributions. Three findings challenge common assumptions: (i) a calibrated rule-based controller achieves the lowest cost on *all* six workloads and zero constraint violations on steady-state traffic, though it trails the best RL agents on bursty and flash patterns; (ii) discrete-action algorithms outperform continuous-action ones by one to two orders of magnitude in constraint violations due to action-space mismatch; and (iii) no single algorithm dominates across workload types, with rankings shifting by up to four positions between steady-state and bursty traffic. On bursty workloads—where RL should shine—PPO reduces constraint violations by 54% relative to the calibrated baseline, but only at 24% higher cost, suggesting that the bottleneck in RL-based resource control is not algorithm selection but baseline calibration, reward engineering, and realistic evaluation protocols.

[1]Workday AI Research [2]The George Washington University, Washington, DC, USA. Correspondence to: Guilin Zhang <guilin.zhang@gwu.edu>.

*Preprint. June 10, 2026.*

## 1. Introduction

Adaptive resource control—allocating compute resources to a dynamic workload while respecting cost and service-level constraints—is a canonical decision-making problem that combines offline training on historical traces with online adaptation to shifting traffic patterns. A growing body of work applies deep reinforcement learning (DRL) to this problem (Rossi et al., 2019; Qiu et al., 2020; Wang et al., 2022; Toka et al., 2021), typically comparing a single RL algorithm against a simple rule-based controller on one or two workload patterns. Yet the conclusions differ sharply across studies: some report that RL improves cost by 30%, others find that rule-based controllers remain competitive. This inconsistency hinders progress: without a shared evaluation protocol, practitioners cannot assess which algorithmic advances are real.

We trace the inconsistency to three gaps in current practice:

**(1) Uncalibrated baselines.** Rule-based controllers such as threshold-driven autoscalers have tunable parameters (target utilization, cool-down windows) that strongly affect performance. When RL studies compare against an uncalibrated baseline, apparent improvements may reflect baseline weakness rather than algorithmic gains (Dulac-Arnold et al., 2021).

**(2) Single-seed reporting.** DRL training exhibits high variance across random seeds, and many reported improvements fall within the noise margin of a single method (Henderson et al., 2018; Islam et al., 2017; Agarwal et al., 2021). Without error bars computed across seeds, benchmark rankings are unreliable.

**(3) Narrow workload coverage.** Most studies evaluate on one or two traffic patterns, typically a diurnal or bursty trace collected from a single deployment. Because workload characteristics strongly interact with scaling policies, narrow coverage yields conclusions that do not transfer to other

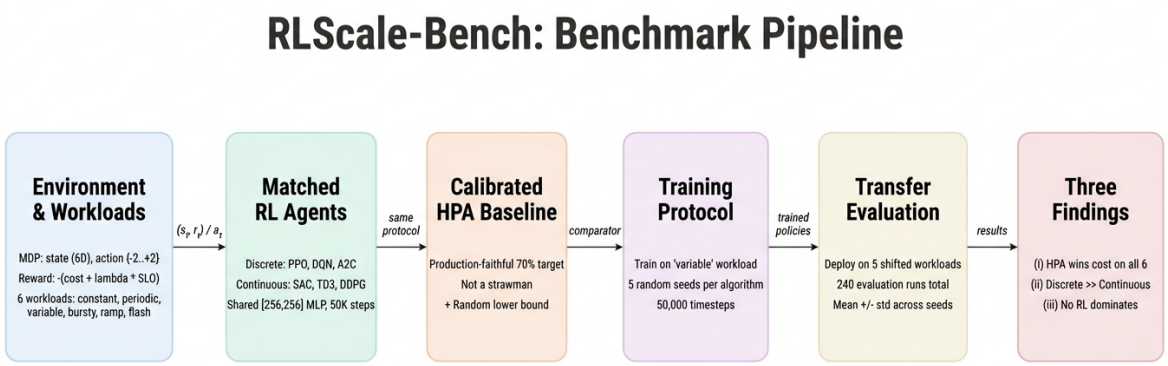


*Figure 1.* RLSCALE-BENCH pipeline. The six stages realize our contributions: matched RL agents (C1), calibrated HPA baseline (C2), 5-seed training and 240-run evaluation (C3), deployment on five shifted workloads (C4), and the three counter-intuitive findings (C5).

deployments.

We address these gaps with RLSCALE-BENCH, a benchmark that follows the reproducible-evaluation principles advocated by Agarwal et al. (2021) and extends them to a real-world decision-making setting. We instantiate the benchmark on Kubernetes Horizontal Pod Autoscaling—a canonical adaptive resource control problem deployed in production at large scale (Kubernetes Authors, 2024; Burns et al., 2016)—and release an open simulator, trained models, and evaluation data. Our contributions are:

- **C1.** A **benchmark and evaluation protocol** for adaptive resource control that matches network architectures ($[256, 256]$ MLP), training budgets (50K steps), and reward functions across PPO, DQN, A2C, SAC, TD3, and DDPG, eliminating confounds from implementation choices.
- **C2.** A **calibrated rule-based baseline** tuned to realistic production settings (70% target utilization), serving as a strong comparator rather than a strawman.
- **C3. Statistically rigorous evaluation** across 5 seeds and 6 workload patterns (constant, periodic, variable, bursty, ramp, flash), yielding 240 evaluation runs with error bars reported throughout.
- **C4.** A **distribution-shift generalization study** that trains agents on a variable workload and deploys them on five shifted workloads, revealing which algorithms adapt and which collapse.
- **C5.** Three **counter-intuitive findings**: (i) the calibrated baseline achieves lowest cost on all six workloads; (ii) discrete-action algorithms outperform continuous-action ones by orders of magnitude due to action-space mismatch; and (iii) no single RL algorithm dominates across workloads.

These findings challenge a common assumption in the decision-making literature that deep RL straightforwardly outperforms rule-based control. We argue that progress requires moving beyond algorithm novelty toward reward engineering, environment calibration, and evaluation protocols that reflect real-world deployment challenges—themes this benchmark is designed to support.

## 2. Related Work

**Decision-making benchmarks for real-world control.** A growing line of work calls for evaluation protocols that reflect the challenges of deploying RL in realistic, sequential decision-making settings. Agarwal et al. (2021) argue that single-seed results in deep RL benchmarks are statistically unreliable and propose stratified bootstrap for confidence intervals. Henderson et al. (2018) show that hyperparameter choices, random seeds, and implementation details can flip algorithm rankings, with some reported gains falling within single-method variance. Dulac-Arnold et al. (2021) identify weak baselines as a systemic issue across applied RL. Our benchmark applies these lessons to adaptive resource control, combining matched training budgets, multiple seeds, and distribution-shift evaluation in a single protocol.

**RL for cloud and container resource management.** RL has been applied to cloud resource management with increasing sophistication. Rossi et al. (2019) applied Q-learning to horizontal and vertical container scaling. Qiu et al. (2020) proposed FIRM for fine-grained microservice resource management under SLO constraints. Wang et al. (2022) introduced DeepScaling for stable CPU utilization in large-scale production systems. Toka et al. (2021) applied machine learning to Kubernetes edge cluster scaling; Zhang et al. (2025a) developed a GPU-aware Kubernetes simulator with PPO-based autoscaling, demonstrating a 75% reward improvement over CPU-only baselines; and Garí et al. (2021) survey the broader field of RL-based cloud autoscaling. These studies typically compare one or two RL algorithms against a lightly-tuned rule-based controller on a narrow workload range; cross-study comparison is difficult because network architectures, training budgets, and baseline configurations vary substantially. Our matched-budget, multi-algorithm, multi-workload protocol is designed to remove these confounds.

**Rule-based autoscaling as a baseline.** The Kubernetes HPA adjusts replica counts based on observed CPU or custom metrics (Kubernetes Authors, 2024), and KEDA (KEDA Authors, 2024) extends HPA with event-driven scaling from external sources. While rule-based controllers are often dismissed as "simple," we show that a properly calibrated HPA is a surprisingly strong comparator—consistent with observations from Booth et al. (2023) that reward misdesign and baseline weakness can produce misleading comparisons in applied RL.

**Distribution-shift generalization.** A central challenge in real-world RL deployment is generalizing from a training distribution to shifted deployment conditions—a concern that motivates the offline-to-online RL literature and broader work on distribution shift. We do not study offline-to-online RL in the strict sense (initializing from a fixed offline dataset and fine-tuning online); instead, we evaluate distribution-shift generalization by training each agent online in simulation on a single workload (*variable*) and deploying it without retraining on five shifted distributions. Our finding that the best training-time algorithm is rarely the best deployment-time algorithm connects to broader observations about distribution shift in applied RL.

**RL infrastructure and environments.** Gymnasium (Brockman et al., 2016) and Stable-Baselines3 (Raffin et al., 2021) provide standardized environments and algorithm implementations. Microservice benchmarks such as DeathStarBench (Gan et al., 2019) provide realistic workloads but lack integrated RL evaluation frameworks. RLSCALE-BENCH is, to our knowledge, the first benchmark that combines a realistic adaptive resource control environment, matched-budget multi-algorithm evaluation, and explicit distribution-shift evaluation in a single protocol.

## 3. Benchmark Design

Figure 1 summarizes the benchmark end-to-end; this section details each stage.

### 3.1. Environment

We instantiate adaptive resource control as a Markov Decision Process (MDP) following the Gymnasium interface (Brockman et al., 2016), using Kubernetes Horizontal Pod Autoscaling as a concrete testbed. The agent allocates compute resources (pod replicas) to a service handling a dynamic request stream, subject to infrastructure cost and service-level constraints. While we instantiate the benchmark on Kubernetes to ensure production realism—following prior work on simulation-based evaluation of RL autoscalers (Zhang et al., 2025a;b)—the abstraction applies broadly to cloud scheduling, database provisioning, and edge inference deployment.

**State space.** The agent observes a 6-dimensional state vector at each decision step: $s_t = [\text{CPU}_t,\ \text{Mem}_t,\ \text{QPS}_t,\ p95_t,\ \text{ErrRate}_t,\ \text{Replicas}_t]$, where $\text{CPU}_t \in [0, 100]$ is CPU utilization (%), $\text{Mem}_t \in [0, 512]$ is memory usage (MB), $\text{QPS}_t$ is the current request rate, $p95_t$ is 95th-percentile latency (ms), $\text{ErrRate}_t \in [0, 1]$ is the error rate, and $\text{Replicas}_t \in [1, 10]$ is the current replica count.

**Action space.** The action space is `Discrete(5)`, representing replica count changes: $a_t \in \{-2, -1, 0, +1, +2\}$. This reflects a fundamental constraint of physical resource allocation—replicas are indivisible units, a property shared by many real-world control problems (allocation of virtual machines, database shards, or hardware accelerators). For continuous-action algorithms (SAC, TD3, DDPG), we apply a `DiscreteToBoxWrapper` that maps $\text{Box}(-1, 1) \to \text{Discrete}(5)$ via uniform bin edges at $[-0.6, -0.2, 0.2, 0.6]$.

**Reward function.** The reward balances infrastructure cost against SLO compliance:

$$r_t = -\Big(\underbrace{c_{\text{rep}} \cdot \text{Replicas}_t}_{\text{cost}} + \lambda \cdot \underbrace{\mathbb{1}[\text{SLO violated}]}_{\text{penalty}}\Big) \quad (1)$$

where $c_{\text{rep}} = 0.01$ USD per replica per step and $\lambda = 1.0$ controls the SLO violation penalty. We normalize rewards to $[-1, 0]$ via min-max scaling based on empirical bounds, which stabilizes training across all six algorithms.

**Workload generator.** We implement six workload patterns that span the diversity of production traffic (Table 1):

*Table 1.* Workload types and their characteristics.

| Type | Pattern | Range (req/min) |
|---|---|---|
| Constant | Flat rate + noise | $100 \pm 10$ |
| Periodic | Sinusoidal (10-min cycle) | 50–200 |
| Variable | Random walk | 30–250 |
| Bursty | Baseline + Poisson spikes | 50–300 |
| Ramp | Linear increase | $50 \rightarrow 250$ |
| Flash | Sudden 3× spike | $80 \rightarrow 240 \rightarrow 80$ |

### 3.2. Algorithms

We evaluate six DRL algorithms from Stable-Baselines3 (Raffin et al., 2021) spanning three families:

- **On-policy, discrete:** PPO (Schulman et al., 2017), A2C (Mnih et al., 2016)
- **Off-policy, discrete:** DQN (Mnih et al., 2015)
- **Off-policy, continuous:** SAC (Haarnoja et al., 2018), TD3 (Fujimoto et al., 2018), DDPG (Lillicrap et al., 2016)

All algorithms use: (i) identical MLP architecture with two hidden layers of 256 units each, (ii) a training budget of 50,000 timesteps on the *variable* workload, and (iii) 5 random seeds per algorithm for statistical robustness. Algorithm-specific hyperparameters (learning rate, batch size, buffer size) follow Stable-Baselines3 defaults except where noted in the appendix.

### 3.3. Baselines

**Calibrated rule-based controller (HPA).** We implement the production-faithful Kubernetes Horizontal Pod Autoscaler with a 70% CPU utilization target: desired $= \lceil \text{current} \times (\text{CPU}/70) \rceil$, clamped to $[1, 10]$ replicas. The 70% target follows the default production configuration and provides a built-in safety margin for bursty traffic. We emphasize that this baseline is *calibrated*, not a strawman: the target utilization, clamp bounds, and decision interval match realistic production deployments, enabling a fair comparison against RL agents.

**Random.** A uniform random policy that selects actions from $\{-2, -1, 0, +1, +2\}$ with equal probability, establishing a lower bound on performance.

### 3.4. Evaluation Protocol

Each trained model is evaluated on all six workload types over 240 decision steps (60 simulated minutes at 15-second intervals). We report: **(1)** total infrastructure cost (USD), **(2)** total SLO violations (count of steps where latency exceeds threshold or error rate exceeds 5%), and **(3)** mean replica count. All metrics include mean $\pm$ std across 5 seeds. For baselines (HPA, Random), we run 5 seeds with different random noise realizations.

## 4. Experiments

### 4.1. Main Results

Table 2 presents infrastructure costs across all algorithms and workloads. Our first and most striking finding is that **HPA achieves the lowest cost on all six workloads**, outperforming every RL algorithm. This result holds because HPA scales conservatively—maintaining fewer replicas on average (2.0–3.0 vs. 2.7–3.8 for RL agents)—while still achieving reasonable SLO compliance.

Table 3 reports SLO violations. Here the picture is more nuanced: while HPA achieves zero violations on constant, periodic, and ramp workloads, it incurs 30.0 violations on bursty traffic—significantly more than PPO ($13.7 \pm 5.6$) and SAC ($8.1 \pm 7.5$). This suggests that **RL's advantage emerges specifically on unpredictable workloads** where proactive scaling can preempt SLO breaches.

### 4.2. Discrete vs. Continuous Action Spaces

Figure 2 reveals a dramatic performance gap between algorithm families. Continuous-action algorithms (SAC, TD3, DDPG) exhibit SLO violations that are **one to two orders of magnitude higher** than their discrete-action counterparts (PPO, DQN, A2C).

TD3 shows extreme variance (std = 4.38 on mean replicas), indicating that some seeds learn functional policies while others degenerate. DDPG consistently converges to a single-replica policy across all seeds, resulting in the lowest cost ($0.0011) but catastrophic SLO violations (300–989 per workload).

This failure mode arises from the *action-space mismatch*: Kubernetes scaling is inherently discrete (integer replicas), yet continuous algorithms output real-valued actions that must be bucketed at bin edges. The discretization aliases the reward signal: within a bin, small changes to the actor's output produce no change in reward (near-zero gradient); at bin edges, infinitesimal changes produce discrete jumps (high-variance gradient). The 50K budget is also short for off-policy continuous algorithms, so part of the gap may reflect undertraining—a learning-curve ablation is left to future work.

### 4.3. Cost-SLO Trade-off

Figure 3 shows the composite score (normalized cost + SLO) across algorithm-workload pairs. HPA achieves the lowest composite scores overall (0.00–0.18); among RL methods, SAC is strongest on SLO-dominated workloads and PPO

*Table 2.* Total infrastructure cost (USD) across 6 algorithms, 2 baselines, and 6 workload types. Each cell shows mean ± std over 5 random seeds. **Bold**: best RL algorithm per workload. Underline: best overall. Only algorithms with mean SLO violations < 100 are eligible for marking.

| Algorithm | Constant | Periodic | Variable | Bursty | Ramp | Flash |
|---|---|---|---|---|---|---|
| RANDOM | 0.0059 ± 0.0007 | 0.0059 ± 0.0007 | 0.0059 ± 0.0007 | 0.0059 ± 0.0007 | 0.0059 ± 0.0007 | 0.0059 ± 0.0007 |
| HPA | 0.0022 ± 0.0000 | 0.0025 ± 0.0000 | 0.0032 ± 0.0000 | 0.0025 ± 0.0000 | 0.0033 ± 0.0000 | 0.0021 ± 0.0000 |
| PPO | 0.0031 ± 0.0005 | 0.0034 ± 0.0004 | **0.0036 ± 0.0003** | 0.0031 ± 0.0007 | 0.0039 ± 0.0003 | **0.0030 ± 0.0009** |
| DQN | **0.0031 ± 0.0013** | **0.0034 ± 0.0008** | 0.0036 ± 0.0009 | **0.0029 ± 0.0013** | 0.0039 ± 0.0008 | 0.0031 ± 0.0014 |
| A2C | 0.0032 ± 0.0008 | 0.0036 ± 0.0005 | 0.0042 ± 0.0008 | 0.0038 ± 0.0011 | **0.0037 ± 0.0008** | 0.0036 ± 0.0007 |
| SAC | 0.0036 ± 0.0005 | 0.0040 ± 0.0008 | 0.0041 ± 0.0009 | 0.0037 ± 0.0005 | 0.0043 ± 0.0008 | 0.0039 ± 0.0002 |
| TD3 | 0.0052 ± 0.0055 | 0.0052 ± 0.0055 | 0.0052 ± 0.0055 | 0.0052 ± 0.0055 | 0.0052 ± 0.0055 | 0.0052 ± 0.0055 |
| DDPG | 0.0011 ± 0.0000 | 0.0011 ± 0.0000 | 0.0011 ± 0.0000 | 0.0011 ± 0.0000 | 0.0011 ± 0.0000 | 0.0011 ± 0.0000 |

*Table 3.* SLO violations across algorithms and workloads. Each cell shows mean ± std over 5 seeds. **Bold**: best RL algorithm. Lower is better; 0 indicates full SLO compliance.

| Algorithm | Constant | Periodic | Variable | Bursty | Ramp | Flash |
|---|---|---|---|---|---|---|
| RANDOM | 0.1 ± 0.1 | 152.3 ± 70.0 | 115.7 ± 47.9 | 33.4 ± 40.1 | 216.2 ± 99.5 | 68.8 ± 51.3 |
| HPA | 0.0 ± 0.0 | 0.0 ± 0.0 | 10.8 ± 0.0 | 30.0 ± 0.0 | 0.0 ± 0.0 | 15.0 ± 0.0 |
| PPO | **0.0 ± 0.0** | **0.0 ± 0.0** | 12.6 ± 12.8 | 13.7 ± 5.6 | **0.0 ± 0.0** | 19.1 ± 17.8 |
| DQN | 0.1 ± 0.2 | 5.3 ± 10.6 | 31.2 ± 31.1 | 17.4 ± 11.1 | 6.1 ± 8.3 | 32.5 ± 31.7 |
| A2C | 0.0 ± 0.0 | 0.1 ± 0.2 | 12.9 ± 12.6 | 12.4 ± 5.7 | 0.2 ± 0.3 | 23.3 ± 21.4 |
| SAC | 0.0 ± 0.0 | 0.0 ± 0.0 | **1.9 ± 2.5** | **8.1 ± 7.5** | 0.0 ± 0.0 | **18.9 ± 24.6** |
| TD3 | 0.5 ± 0.5 | 452.1 ± 412.7 | 475.2 ± 433.8 | 75.0 ± 68.5 | 593.2 ± 541.5 | 180.0 ± 164.3 |
| DDPG | 0.9 ± 0.0 | 753.5 ± 0.0 | 791.9 ± 0.0 | 125.0 ± 0.0 | 988.6 ± 0.0 | 300.0 ± 0.0 |

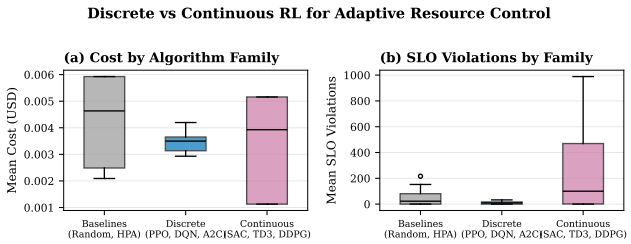


*Figure 2.* Discrete-action algorithms (PPO, DQN, A2C) achieve dramatically lower SLO violations than continuous-action algorithms (SAC, TD3, DDPG). The continuous family's median SLO count is >100× higher.

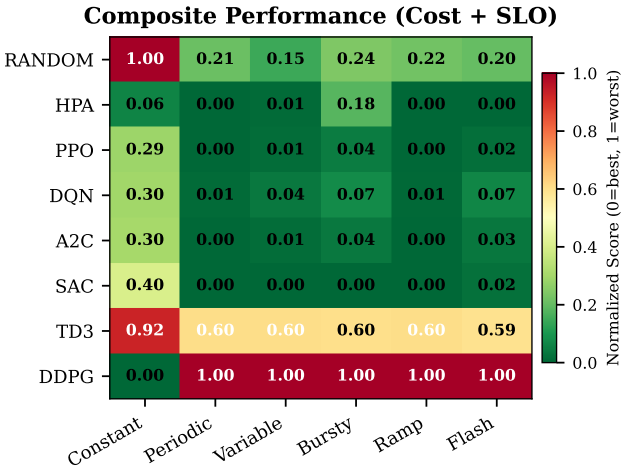


*Figure 3.* Composite performance heatmap (0 = best, 1 = worst). HPA and SAC form the Pareto frontier among viable algorithms. DDPG's degenerate policy scores 1.00 on all dynamic workloads.

on cost-dominated ones, together forming the cost–SLO frontier analyzed in Appendix D. Notably, DDPG achieves a perfect 1.00 (worst) on every non-constant workload due to its degenerate single-replica policy, confirming that **cost minimization without SLO awareness is not a viable strategy**.

### 4.4. Ranking Stability Under Workload Shift

Figure 4 tracks how algorithm rankings (by SLO violations) shift across workload types—a stress test for the common practice of training on one distribution and deploying on another. No single algorithm maintains a consistent rank: HPA is rank 1 on constant and periodic workloads but drops to rank 5 on bursty traffic; PPO maintains the most stable ranking (rank 2–3); DQN is consistently the weakest viable algorithm (rank 4–5); SAC shows the widest rank variance, excelling on variable workloads (rank 1) but performing poorly on ramp traffic (rank 4).

This instability carries a direct implication for train-to-deploy distribution shift: **the best algorithm selected on a training distribution is unlikely to remain best after workload shift**. Benchmarks that evaluate only on the

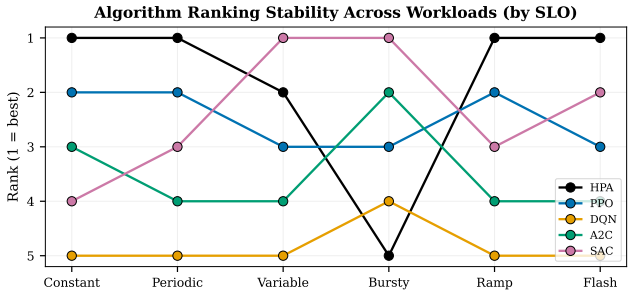


*Figure 4.* Algorithm ranking (by SLO violations) shifts across workloads. No algorithm maintains rank 1 on all patterns. Only viable algorithms shown (TD3, DDPG excluded).

training distribution systematically overestimate deployed performance.

### 4.5. Distribution-Shift Generalization

All agents were trained on the *variable* workload—a random-walk trace designed to expose the policy to diverse conditions—and deployed on the other five workloads without retraining. This probes distribution-shift generalization: can a policy trained on one traffic distribution adapt when deployment conditions shift?

Table 3 answers this question through the lens of SLO compliance. PPO, trained on *variable*, achieves 0 violations on *constant*, *periodic*, and *ramp*—it generalizes successfully to these distributions. On *bursty* and *flash*, however, violations rise to 13.7 and 19.1 respectively, showing that the policy fails to extrapolate to heavy-tail traffic it did not see during training. SAC generalizes more robustly on dynamic workloads (1.9 violations on *variable* itself, 8.1 on *bursty*) but at the cost of systematically higher resource usage (3.19–3.78 mean replicas vs. 2.68–3.18 for PPO). DQN and A2C fall between these regimes: both inherit PPO's cost efficiency on steady workloads but degrade more than PPO on bursty traffic. We call this trade-off the **transfer tax**: algorithms that generalize well under distribution shift pay for robustness with additional resource cost.

Critically, the calibrated rule-based baseline incurs no transfer tax because it does not train on any distribution—its policy is a fixed function of current CPU utilization. This explains why a properly tuned rule-based controller beats many RL agents under distribution shift: it simply does not overfit.

## 5. Discussion

**Why is the calibrated baseline so competitive?** A properly tuned rule-based controller is surprisingly hard to beat. We attribute this to three factors: (i) CPU utilization is a strong, low-latency proxy for load in most production workloads, making reactive scaling effective for steady-state and mildly dynamic traffic; (ii) the 70% target provides a built-in safety margin that prevents constraint violations under gradual load changes; (iii) the rule-based policy does not train on any distribution, so it incurs no transfer tax when workload shifts. This echoes observations in broader applied RL that simple, well-tuned baselines are routinely underestimated (Dulac-Arnold et al., 2021; Henderson et al., 2018).

**When does RL help?** RL's advantage concentrates on *unpredictable* workloads (bursty, flash) where proactive scaling—learned from experience—can preempt violations that a reactive policy cannot avoid. On bursty traffic, PPO reduces violations by 54% relative to the calibrated baseline (13.7 vs. 30.0) at 24% higher cost. The practical implication: RL is most valuable in deployments where the cost of constraint violations (revenue loss, user churn) substantially exceeds resource cost. In cost-dominated settings, the rule-based controller wins.

**The action-space mismatch problem.** TD3 and DDPG fail catastrophically because the discretization wrapper aliases the reward signal: the actor sees a piecewise-constant reward surface with flat plateaus inside bins and sharp jumps at bin edges, producing near-zero gradients most of the time and high-variance estimates at transitions. SAC partially overcomes this through entropy-regularized exploration but still pays 15–30% higher cost than discrete alternatives. The lesson generalizes beyond autoscaling: any decision-making problem with inherently discrete actions (allocation of indivisible units, discrete network flows, integer programming) should prefer discrete-action algorithms or develop differentiable abstractions.

**Implications for benchmark design in decision-making research.** Our findings carry three implications for benchmark designers in the decision-making community: (1) **Calibrate baselines.** A weak baseline inflates apparent gains; a strong baseline reveals where progress is real. Benchmarks should document baseline tuning as carefully as RL hyperparameters. (2) **Evaluate under distribution shift.** Rankings on the training distribution do not predict rankings under deployment shift. Benchmarks should include held-out distributions as a first-class requirement. (3) **Report error bars across seeds.** Single-seed runs can flip rankings between discrete and continuous algorithms, or between PPO and DQN. Claims based on one seed are unfalsifiable.

**Implications for practitioners.** For production deployments: (i) always calibrate the rule-based controller before evaluating RL—many reported gains may reflect baseline weakness rather than algorithmic superiority; (ii) prefer discrete-action algorithms (PPO, A2C) for indivisible-unit

resource allocation; (iii) evaluate on a diversity of workload patterns, because algorithm rankings are workload-dependent.

**Limitations.** The benchmark uses a simulated environment calibrated to real Kubernetes metrics from a Kind cluster. This choice enables reproducibility and scale (240 runs), but may not capture all dynamics of production clusters (network latency under load, node failures, multi-tenancy). The training budget (50K steps) is modest; longer training may narrow the gap between algorithms, though our matched-budget design ensures fair cross-algorithm comparison. We evaluate single-service scaling; multi-service and cluster-level scaling remain open challenges that we plan to address in future work via learned graph-based dynamics models.

## 6. Conclusion

We introduced RLSCALE-BENCH, a reproducible benchmark and evaluation protocol for deep reinforcement learning on adaptive resource control, instantiated on Kubernetes Horizontal Pod Autoscaling. A systematic comparison of six algorithms across six workload patterns and five random seeds reveals that (1) a calibrated rule-based controller remains the most cost-effective solution; (2) discrete-action algorithms outperform continuous-action ones by orders of magnitude due to action-space mismatch; and (3) no single algorithm dominates across workloads, with rankings shifting by up to four positions under distribution shift.

These findings challenge the prevailing narrative that deep RL straightforwardly outperforms rule-based control in resource management. We argue instead that the bottleneck lies in reward engineering, baseline calibration, and evaluation protocols that reflect real-world deployment challenges—problems the decision-making community can address through shared benchmarks and higher evaluation standards. We release the full benchmark suite, trained models, and evaluation data to support this goal.[1]

**Future work.** We plan to extend RLSCALE-BENCH in three directions: (i) learned world models for cascade-aware planning, where an agent rolls out trajectories in a predicted dynamics model to preempt chain failures before they occur; (ii) safe planning via constrained rollouts that guarantee zero violations during deployment; and (iii) graph-structured resource control across multi-service topologies, where scaling decisions propagate through a dependency graph.

## References

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., and Bellemare, M. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems*, volume 34, pp. 29304–29320, 2021.

Booth, S., Knox, W. B., Shah, J., Niekum, S., Stone, P., and Allievi, A. The perils of trial-and-error reward design: Misdesign through overfitting and invalid task specifications. In *AAAI Conference on Artificial Intelligence*, volume 37, pp. 5920–5929, 2023.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

Burns, B., Grant, B., Oppenheimer, D., Brewer, E., and Wilkes, J. Borg, omega, and Kubernetes: Lessons learned from three container-management systems over a decade. *ACM Queue*, 14(1):70–93, 2016.

Dulac-Arnold, G., Levine, N., Mankowitz, D. J., Li, J., Paduraru, C., Gowal, S., and Hester, T. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110:2419–2468, 2021.

Fujimoto, S., Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pp. 1587–1596. PMLR, 2018.

Gan, Y., Zhang, Y., Cheng, D., Shetty, A., Rathi, P., Katarki, N., Bruno, A., Hu, J., Ritchken, B., Jackson, B., et al. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 3–18, 2019.

Garí, Y., Monge, D. A., Pacini, E., Mateos, C., and García Garino, C. Reinforcement learning-based application autoscaling in the cloud: A survey. *Engineering Applications of Artificial Intelligence*, 102:104288, 2021. doi: 10.1016/j.engappai.2021.104288.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pp. 1861–1870. PMLR, 2018.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. In *AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Islam, R., Henderson, P., Gomrokchi, M., and Precup, D. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. In *ICML Workshop on Reproducibility in Machine Learning*, 2017. arXiv:1708.04133.

[1] Code available upon publication.

KEDA Authors. KEDA: Kubernetes-based event driven autoscaling. https://keda.sh/, 2024. Accessed: 2026-03-15.

Kubernetes Authors. Kubernetes horizontal pod autoscaler. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/, 2024. Accessed: 2026-03-15.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2016.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015. doi: 10.1038/nature14236.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937. PMLR, 2016.

Qiu, H., Banerjee, S. S., Jha, S., Kalbarczyk, Z. T., and Iyer, R. K. FIRM: An intelligent fine-grained resource management framework for SLO-oriented microservices. *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 805–825, 2020.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-Baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

Rossi, F., Nardelli, M., and Cardellini, V. Horizontal and vertical scaling of container-based applications using reinforcement learning. In *IEEE International Conference on Cloud Computing (CLOUD)*, pp. 329–338. IEEE, 2019.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. In *arXiv preprint arXiv:1707.06347*, 2017.

Toka, L., Dobreff, G., Fodor, B., and Sonkoly, B. Machine learning-based scaling management for Kubernetes edge clusters. *IEEE Transactions on Network and Service Management*, 18(1):958–972, 2021. doi: 10.1109/TNSM.2021.3052837.

Wang, Z., Zhu, S., Li, J., Jiang, W., Ramakrishnan, K. K., Zheng, Y., Yan, M., Zhang, X., and Liu, A. X. DeepScaling: Microservices autoscaling for stable CPU utilization in large scale cloud systems. In *ACM Symposium on Cloud Computing (SoCC)*, pp. 16–30, 2022. doi: 10.1145/3542929.3563469.

Zhang, G., Guo, W., Tan, Z., Guan, Q., and Jiang, H. KIS-S: A GPU-aware Kubernetes inference simulator with RL-based auto-scaling. In *IEEE International Performance, Computing, and Communications Conference (IPCCC)*, 2025a. doi: 10.1109/IPCCC62082.2025.11304654.

Zhang, G., Guo, W., Tan, Z., and Jiang, H. AMP4EC: Adaptive model partitioning framework for efficient deep learning inference in edge computing environments. In *IEEE International Conference on Fog and Mobile Edge Computing (FMEC)*, 2025b. doi: 10.1109/FMEC65595.2025.11119384.

## A. Hyperparameter Details

Table 4 lists the full hyperparameter configuration for each algorithm. All algorithms share the same network architecture and training budget; algorithm-specific parameters follow Stable-Baselines3 (Raffin et al., 2021) defaults except where noted.

## B. Environment Calibration

The simulated environment is calibrated against metrics collected from a real Kubernetes cluster running Online Boutique on Kind with Prometheus monitoring. Key calibration points:

- **CPU response**: $\text{CPU\%} = 5.0 + (\text{QPS}/\text{replicas}) \times 0.7$, calibrated so HPA triggers scale-up at ∼100 req/min per replica.
- **Latency model**: $p95 = 50 + (\text{QPS}/\text{replicas})^{1.5} \times 0.08$ ms, with exponential degradation under overload.
- **SLO threshold**: $p95 < 500$ ms and error rate $< 5\%$.
- **Cost model**: \$0.01 per replica per decision step (15 seconds), approximating on-demand cloud pricing at \$0.04/vCPU-hour.

## C. Full Results: Viable Algorithm Comparison

Figure 5 shows the complete cost and SLO comparison for viable algorithms (HPA, PPO, DQN, A2C, SAC) across all six workloads with 95% confidence intervals.

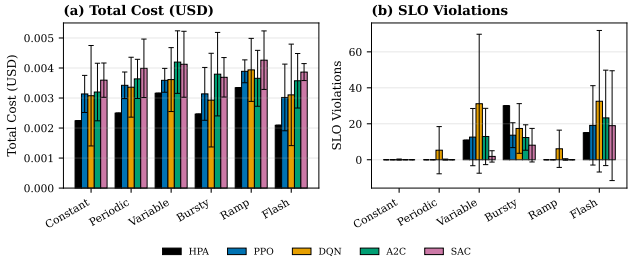


*Figure 5.* Cost and SLO violations for viable algorithms across all workloads. Error bars show 95% CI over 5 seeds. HPA achieves lowest cost on all workloads. RL algorithms show advantage only on bursty/flash SLO compliance.

## D. Cost-SLO Pareto Analysis

Figure 6 shows the cost-SLO scatter for all algorithms. Each point represents one (algorithm, workload) pair. The Pareto front connects methods that are not dominated on both cost and SLO simultaneously.

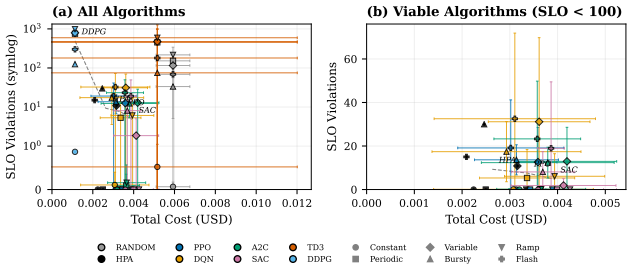


*Figure 6.* Cost-SLO Pareto front. Left panel: all algorithms (symlog scale). Right panel: viable algorithms only. HPA dominates on cost; SAC achieves lowest SLO on dynamic workloads.

*Table 4.* Hyperparameter configuration. Shared parameters are listed under "Common"; algorithm-specific parameters follow SB3 defaults.

| **Parameter** | **Value** |
|---|---|
| *Common* | |
| Network architecture | MLP $[256, 256]$ |
| Training timesteps | 50,000 |
| Training workload | Variable |
| Decision interval | 15 seconds |
| Episode length | 240 steps (60 min) |
| Reward normalization | Min-max to $[-1, 0]$ |
| Random seeds | $\{42, 123, 456, 789, 1024\}$ |
| *PPO* | |
| Learning rate | $3 \times 10^{-4}$ |
| Batch size | 64 |
| Number of epochs | 10 |
| Clip range | 0.2 |
| GAE $\lambda$ | 0.95 |
| *DQN* | |
| Learning rate | $1 \times 10^{-4}$ |
| Buffer size | 50,000 |
| Batch size | 64 |
| Target update interval | 500 |
| Exploration fraction | 0.3 |
| Final $\epsilon$ | 0.05 |
| *A2C* | |
| Learning rate | $7 \times 10^{-4}$ |
| Number of steps | 5 |
| Value function coefficient | 0.5 |
| Entropy coefficient | 0.01 |
| *SAC* | |
| Learning rate | $3 \times 10^{-4}$ |
| Buffer size | 50,000 |
| Batch size | 256 |
| $\tau$ (soft update) | 0.005 |
| Automatic entropy tuning | Yes |
| *TD3* | |
| Learning rate | $1 \times 10^{-3}$ |
| Buffer size | 50,000 |
| Batch size | 256 |
| Policy delay | 2 |
| Target noise | 0.2 |
| *DDPG* | |
| Learning rate | $1 \times 10^{-3}$ |
| Buffer size | 50,000 |
| Batch size | 256 |
| $\tau$ (soft update) | 0.005 |