SAIR: Cost-Efficient Multi-Stage ML Pipeline Autoscaling via In-Context Reinforcement Learning

Jianchang Su[1*] **Yifan Zhang**[1*] **Shengkai Lin**[2] **Shizhen Zhao**[2]
Yusheng Zheng[3] **Yiwei Yang**[3] **Wei Zhang**[1†]
[1]University of Connecticut [2]Shanghai Jiao Tong University [3]UC Santa Cruz

Abstract

Multi-stage ML inference pipelines are difficult to autoscale because of heterogeneous resources, cross-stage coupling, and dynamic bottleneck migration. Production controllers ignore cross-stage effects, and learning-based autoscalers require offline training data that is unavailable for new pipelines. We present SAIR, an autoscaler for the zero-shot deployment setting that uses an LLM as an in-context reinforcement learning controller, improving its policy online from reward-labeled experience tuples stored in-context. SAIR combines Pareto-dominance reward shaping with a provable separation margin between dominated and non-dominated actions, Bayesian-surprise experience retrieval, and a user-space CUDA interception layer that delivers continuous GPU rate control with sub-millisecond actuation overhead. We prove a regret bound that decomposes into retrieval coverage and LLM selection components, each mapping to a concrete SAIR mechanism. On four ML serving pipelines under three workload patterns, SAIR achieves the best or tied-best P99 latency in every configuration (up to 50% reduction over the next baseline); billable cost is reduced through replica consolidation, and effective cost (which captures GPU rate-control savings under co-location or MIG) by up to 97% under GPU sharing (best case; representative 30–60% over the next-best baseline), validated by co-location measurements on the physical testbed. SAIR matches the steady-state performance of a warm-started controller within 2–3 decision intervals and achieves 86% bottleneck detection accuracy, all from online experience alone.

1 Introduction

Modern machine learning serving systems increasingly decompose monolithic inference into multi-stage pipelines to make better use of heterogeneous hardware [Crankshaw et al., 2017, 2020, Gunasekaran et al., 2022]. A typical pipeline consists of CPU/memory-bound preprocessing (data transformation), GPU-accelerated inference (neural network execution), and CPU/memory-bound postprocessing (result aggregation), connected through message queues. This architecture improves throughput and modularity, but creates an autoscaling problem that existing controllers cannot handle well.

Three coupled phenomena cause this problem. First, *dynamic bottleneck migration* shifts the performance-limiting stage with workload intensity. Second, *cross-stage coupling* propagates scaling decisions through the pipeline with variable delays: scaling preprocessing alone can trigger a delayed GPU contention spike 1 to 2 intervals later. Third, *temporal causality confounds root-cause identification*: high GPU utilization may indicate either GPU shortage or downstream congestion.

Existing autoscalers fall into three categories, each inadequate for this setting. Rule-based controllers (HPA [The Kubernetes Authors, 2026], VPA) operate on single-stage metrics and lack cross-stage

*Equal contribution. †Corresponding author: wei.13.zhang@uconn.edu.

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

Table 1: Design space of autoscalers for ML inference pipelines along axes that matter for the zero-shot deployment setting.

Approach	Offline data	Cross-stage	GPU rate ctrl	Strongest bound
HPA / VPA [The Kubernetes Authors, 2026]	none	no	no	none
Threshold (rule-based)	none	no	no	none
FIRM / AWARE [Qiu et al., 2020, 2023]	10^3 episodes	yes	no	none
Sinan [Zhang et al., 2021]	per-app profile	partial	no	none
InferLine [Crankshaw et al., 2020]	per-pipeline profile	yes	no	queueing
SAIR (this work)	**none**	**yes**	**yes**	**regret decomp.**

reasoning. Learning-based controllers [Rzadca et al., 2020, Qiu et al., 2020, 2023] apply deep RL but need offline training data and fail to generalize under distribution shift. Heuristic methods require extensive per-workload tuning. We target a deployment setting that all three handle poorly: *zero-shot* capacity scaling on a new pipeline, starting from an empty trace history under active bottleneck migration. We present SAIR, which uses an LLM as an *in-context reinforcement learning* (ICRL) controller. Knowledge accumulates as (x, a, r) experience tuples in the LLM's context window, so the controller deploys directly and adapts online from observed rewards. Our contributions:

- An autoscaler for multi-stage pipelines under bottleneck migration. We formulate the control problem as a contextual bandit with an evolving Pareto frontier, and combine three components: Pareto-dominance reward shaping, Bayesian-surprise experience retrieval, and a user-space `LD_PRELOAD` CUDA interception layer that gives continuous GPU rate control with sub-millisecond actuation overhead (Figure 1).
- A regret bound that decomposes into retrieval coverage error ξ_t and LLM selection error η_t, with a provable constant separation margin for the Pareto reward; each term in the bound corresponds to a concrete SAIR component (retrieval, validator, exploration schedule).
- Empirical validation on four pipelines and three workload patterns: SAIR achieves the best or tied-best P99 latency in every configuration with 86% bottleneck detection accuracy from online experience alone, and ablations confirm each component's contribution. We quantify the cold-start cost: cold-started SAIR converges to warm-started performance within 2–3 decisions, while offline-trained FIRM/AWARE policies transfer as non-adaptive policies on real hardware. Co-location measurements on the physical testbed validate the effective-cost model, and a 4-GPU scale experiment confirms the controller generalizes to larger deployments.

2 Related works

Autoscaling for cloud systems. Autopilot [Rzadca et al., 2020] introduced ML-based vertical scaling at Google but trains its predictor on weeks of historical traces. FIRM [Qiu et al., 2020] and AWARE [Qiu et al., 2023] apply deep RL with safety constraints and recover quickly under distribution shift, but both require thousands of offline episodes to bootstrap usable policies. We additionally implement FIRM and AWARE faithfully (native state dimensions, action spaces, and reward functions) and train each to convergence in a queueing-model simulator; the frozen policies transfer as non-adaptive policies on our testbed (Section 6.4), a measured instance of the sim-to-real gap. Sinan [Zhang et al., 2021] achieves QoS-aware scaling for microservices through learned performance models that need offline profiling per application. HPA [The Kubernetes Authors, 2026] remains the production default and lacks cross-stage reasoning. Table 1 summarizes the design space along four axes that matter for our setting. The key limitation that motivates SAIR (an offline data dependency, or a long online warm-up before the controller is safe to deploy) applies equally to more recent gradient-trained autoscalers.

ML inference serving and LLM scheduling. Clipper [Crankshaw et al., 2017] introduced adaptive batching, InferLine [Crankshaw et al., 2020] uses queueing-theoretic models, and Cocktail [Gunasekaran et al., 2022] addresses multi-dimensional optimization. InferLine is the closest pipeline-aware predictor to SAIR but requires offline workload characterization, which is unavailable in the zero-shot setting we target. For LLM serving, Orca [Yu et al., 2022], vLLM [Kwon et al., 2023], Llumnix [Sun et al., 2024], QLM [Patke et al., 2024], and Chiron [Patke et al., 2025] focus on schedul-

ing within serving clusters; SAIR addresses multi-stage *pipeline* autoscaling across heterogeneous stages. FastMatch [Su et al., 2024] addresses similar CPU–GPU imbalance during training.

In-context learning, RL, and example selection. LLMs can learn from few-shot examples without parameter updates [Brown et al., 2020, Garg et al., 2022, Xie et al., 2022]. For RL, Algorithm Distillation [Laskin et al., 2023] and supervised pretraining [Lee et al., 2023] enable in-context RL. Krishnamurthy et al. [2024] showed that LLMs require explicit interventions to explore reliably in-context; Song et al. [2025] demonstrated reward feedback alone enables ICRL. Related work also studies federated RL [Lan et al., 2025a], contextual integrity [Lan et al., 2025b], and preference optimization [Lan et al., 2025c]. SAIR addresses these failure modes via forced probes, positive-only memory, and a Pareto reward margin (Proposition 5.2). For example selection, KATE [Liu et al., 2022] and IDEAL [Zhang et al., 2024] optimize selection over a fixed pool; we instead select from a *growing* experience buffer and combine similarity with a Bayesian-surprise term that captures information gain (Appendix B).

Multi-objective optimization and GPU sharing. NSGA-II [Deb et al., 2002] introduced Pareto-front approximation; our Pareto reward uses hypervolume contribution [Zitzler and Thiele, 1998] (see Hayes et al. [2022] for an MORL survey). Our retrieval score is a residual analog of Bayesian surprise [Itti and Baldi, 2005]. For GPU sharing, Gandiva [Xiao et al., 2018], AntMan [Xiao et al., 2020], Salus [Yu and Chowdhury, 2020], and Dilu [Lv et al., 2025] provide kernel- or driver-level mechanisms; SAIR uses a lighter user-space `LD_PRELOAD` interception that throttles kernel launches via token-bucket rate limiting.

3 Problem formulation

Consider an ML inference pipeline $\mathcal{S} = \{s_1, s_2, \ldots, s_N\}$ with N stages connected through message queues. Each stage s_i processes requests at service rate μ_i (requests/s) and maintains an input queue with depth q_i (queued requests). The end-to-end latency decomposes as $L_{e2e} = \sum_{i=1}^{N}(L_i^{proc} + L_i^{queue})$, where L_i^{proc} is processing latency and L_i^{queue} is queueing delay at stage i. Each stage has resource configuration $\mathbf{r}_i = (n_i, c_i, m_i, \rho_i)$ where $n_i \in \mathbb{Z}^+$ is replica count, $c_i \in \mathbb{R}^+$ (CPU millicores) and $m_i \in \mathbb{R}^+$ (memory MB) are CPU and memory allocations, and $\rho_i \in [0, 1]$ is the GPU rate ratio (the fraction of physical GPU compute the stage may consume; see Section 4). Stage-level utilizations are $u_i^{cpu}, u_i^{gpu} \in [0, 1]$, and L_{p99} denotes the 99th-percentile end-to-end latency observed at the round.

We formalize multi-stage autoscaling as a *contextual bandit*: each decision interval Δt (default 30 s) is one round t. The abstraction is justified because each stage's queue reaches a quasi-steady-state within Δt once Δt exceeds the longest stage's relaxation time (settling window 5–10 s); it becomes invalid when Δt is too short (prevented by the cooldowns in Table 3) or under within-interval adversarial shifts (Section 7). The context is $x_t = (\{\mathbf{r}_i, q_i, u_i^{cpu}, u_i^{gpu}, L_{p99}\}_{i=1}^{N},\ \mathcal{P}_t)$, where $\mathcal{P}_t$ is the Pareto frontier (Section 4); including $\mathcal{P}_t$ in x_t keeps $r_t = R(x_t, a_t)$ a function of the round-t pair alone. Actions are discretized: $\mathcal{A} = \prod_i \mathcal{A}_i$. For CPU stages $\mathcal{A}_i^{cpu} = \{\Delta n_i, \Delta c_i, \Delta m_i\}$ with $\Delta n_i \in \{-1, 0, +1, +2\}$, $\Delta c_i \in \{\pm\gamma_c, 0\}$, $\Delta m_i \in \{\pm\gamma_m, 0\}$; for GPU stages $\mathcal{A}_i^{gpu} = \{\Delta n_i, \Delta\rho_i\}$ with $\Delta\rho_i \in \{-0.1, 0, +0.1, +0.2\}$. Step granularities are $\gamma_c = 500$ m and $\gamma_m = 256$ MB (Table 3). We seek a policy $\pi : \mathcal{X} \to \mathcal{A}$ minimizing cumulative regret.

We make four assumptions that characterize the problem structure:

Assumption 3.1 (Bounded Reward Range)**.** The shaped reward satisfies $|R_t| \leq R_{max}$ for all rounds. This holds because each reward component is normalized and clipped to a fixed range (see Algorithm 2 in Appendix E).

Assumption 3.2 (Bottleneck Identifiability)**.** There exists a mapping $\phi : \mathcal{X} \to \{1, \ldots, N\}$ identifying the bottleneck stage such that scaling stage $\phi(x)$ yields expected reward improvement at least $\Delta > 0$, while scaling other stages yields improvement at most $\Delta/2$. This reflects queueing dynamics: in a tandem queue, the stage with highest utilization limits throughput, and scaling it yields the largest marginal gain.

Assumption 3.3 (Retrieval Coverage Error)**.** Given context x_t, the experience selection returns a set $\mathcal{E}_t$ that induces a candidate action set $\mathcal{A}(\mathcal{E}_t) \subseteq \mathcal{A}$. The *coverage gap* is

$$\xi_t = \max\Big(0,\ \mathbb{E}[R(x_t, a_t^*)] - \max_{a \in \mathcal{A}(\mathcal{E}_t)} \mathbb{E}[R(x_t, a)]\Big), \tag{1}$$

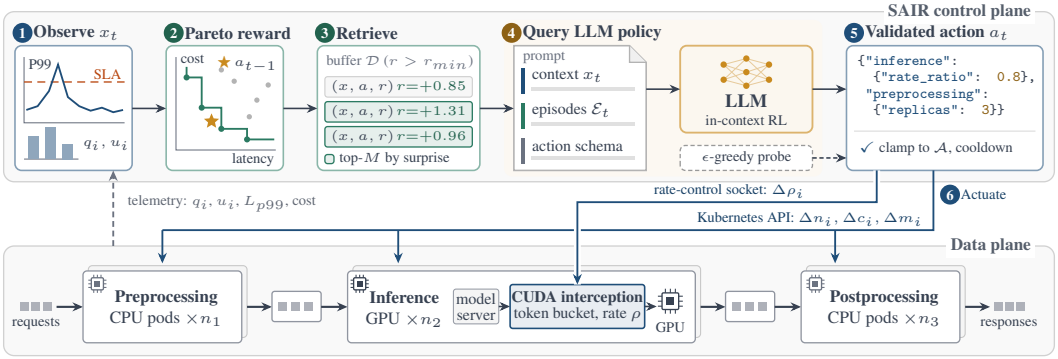


Figure 1: SAIR architecture. The control plane runs one in-context RL round per decision interval Δt over the multi-stage pipeline in the data plane; circled numbers mark the steps of a round.

measuring how much optimality is lost because the retrieved context lacks relevant action patterns.

Assumption 3.4 (LLM Selection Error on Retrieved Set)**.** Conditioned on retrieved experiences $\mathcal{E}_t$, the LLM outputs $a_t \in \mathcal{A}(\mathcal{E}_t)$ such that

$$\max_{a \in \mathcal{A}(\mathcal{E}_t)} \mathbb{E}[R(x_t, a)] - \mathbb{E}[R(x_t, a_t)] \leq \eta_t \tag{2}$$

with probability at least $1 - \delta_{LLM}$. This captures the LLM's bounded suboptimality *within* the actions suggested by retrieved experiences. The constraint $a_t \in \mathcal{A}(\mathcal{E}_t) \subseteq \mathcal{A}$ is enforced by design: SAIR presents a discrete action schema via structured JSON output and applies an action validator (Appendix D) that clamps any out-of-range proposal to the nearest feasible action in $\mathcal{A}$.

Scope. SAIR targets coarse-grained capacity scaling under zero-shot deployment with decision intervals of tens of seconds. Three settings are out of scope: (i) per-request scheduling within a serving cluster, addressed by Orca [Yu et al., 2022], vLLM [Kwon et al., 2023], Llumnix [Sun et al., 2024]; (ii) deployments with abundant offline traces where FIRM, AWARE, or InferLine can be trained or profiled and will outperform a zero-shot controller; (iii) sub-second control loops, since LLM decision latency is 1–2 s.

4 SAIR: In-context RL for autoscaling

Overview. Figure 1 shows the architecture. In each round, SAIR (1) observes the context x_t from stage telemetry, (2) scores the previous action with the Pareto-shaped reward and stores positive episodes in the experience buffer $\mathcal{D}$, (3) retrieves the top-M episodes by Bayesian surprise, (4) queries the LLM with a prompt that combines x_t, the retrieved episodes $\mathcal{E}_t$, and the action schema, (5) validates the returned JSON action, and (6) applies replica and CPU/memory changes through the Kubernetes API and GPU rate changes through the CUDA interception layer. Algorithm 1 gives the complete loop, including the forced probes taken with probability ϵ_t.

In-context RL policy. We define the policy implicitly through the LLM's conditional distribution:

$$\pi_{SAIR}(a|x) = P_{LLM}(a \mid x, \mathcal{E}_{1:M}), \tag{3}$$

where $\mathcal{E}_{1:M} = \{(x_i, a_i, r_i)\}_{i=1}^{M}$ are past episodes provided in-context. Tuples come from live system interactions; the policy improves as more relevant experiences are retrieved (decreasing ξ_t, Theorem 5.1), and the reward signal determines action selection. Naive ICRL can fail at exploration [Krishnamurthy et al., 2024], so SAIR adds forced ϵ_t-greedy probes, positive-only memory, and the Pareto reward margin below.

Pareto-based reward shaping. Threshold-based rewards lead to reactive policies that scale only after SLA violations and oscillate between under- and over-provisioning. We design a multi-objective reward that encourages proactive scaling while balancing latency and cost:

$$R(x, a) = R_{latency} + R_{cost} + R_{SLA} + R_{proactive} + R_{pareto}. \tag{4}$$

The latency component rewards improvement, $R_{latency} = w_L(L_{before} - L_{after})/L_{baseline}$, and the cost component penalizes resource increase, $R_{cost} = -w_C(C_{after} - C_{before})/C_{budget}$. SLA violations grow quadratically:

$$R_{SLA} = \begin{cases} -\left(\frac{L_{after}}{T_{SLA}}\right)^2 + 1 & \text{if } L_{after} > T_{SLA} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

For balancing the fundamental latency-cost tradeoff, we use Pareto-dominance-based reward shaping:

$$R_{pareto}(a) = \begin{cases} 1 + H(a) & \text{if } a \in \mathcal{P} \text{ (non-dominated)} \\ \frac{0.8}{1+d(a,\mathcal{P})} & \text{otherwise} \end{cases} \tag{6}$$

where $\mathcal{P}$ is the current Pareto frontier, $H(a)$ is the hypervolume contribution of a, and $d(a, \mathcal{P})$ is the Euclidean distance to the frontier in normalized objective space.

The proactive scaling bonus encourages multi-stage coordinated scaling during violations:

$$R_{proactive} = \sigma(x) \cdot \mu(a) \cdot w_{proactive} \tag{7}$$

where $\sigma(x) = \max(0, L_{p99}/T_{SLA} - 1)$ is violation severity, T_{SLA} is the per-pipeline SLA latency target, and $\mu(a) = \sum_i |\Delta n_i| + \alpha \sum_i (|\Delta c_i|/\gamma_c + |\Delta m_i|/\gamma_m + |\Delta \rho_i|) + \frac{1}{2}|\text{stages_scaled}|$ aggregates all action dimensions with $\alpha = 0.5$.

The weight settings ($w_L = 0.7$, $w_C = 0.3$, $w_{proactive} = 0.3$, fully listed in Appendix F) reflect the priority that latency violations are user-visible while resource costs are amortized. The ablation rows "Equal Wt." and "Cost Focus" (Table 2) show that removing or inverting the asymmetry degrades performance; the design relies on the relative ordering of latency over cost. The Pareto reward R_{pareto} remains the primary structural signal (Proposition 5.2).

Bayesian-surprise retrieval. The LLM's context window restricts how many experiences can be supplied; with potentially thousands of accumulated experiences in $\mathcal{D}$, selecting the most informative M is critical. We select to maximize information gain:

$$\mathcal{E}^* = \arg \max_{\mathcal{E} \subset \mathcal{D}, |\mathcal{E}| = M} I(\mathcal{E}; \pi^* \mid x_{curr}) \tag{8}$$

We approximate this using a leave-one-out Bayesian-surprise score:

$$\text{score}(e) = \text{sim}(x_e, x_{curr}) \cdot \underbrace{|r_e - \mathbb{E}[r \mid \mathcal{D}_{-e}]|}_{\text{surprise}} \tag{9}$$

where $\text{sim}(x_e, x_{curr}) = \exp(-\|\tilde{x}_e - \tilde{x}_{curr}\|^2/2\sigma^2)$, $\tilde{x}$ is the per-stage $(q_i, u_i^{cpu}, u_i^{gpu})$ vector plus L_{p99} z-score-normalized over the buffer (so distances are unitless), σ=1.0 in this normalized space, and $\mathcal{P}_t$ is summarized by its min-latency/min-cost extreme points appended to $\tilde{x}$. The surprise term identifies unexpected outcomes (Appendix B). With diversity regularization:

$$\mathcal{E}^* = \arg\max_{\mathcal{E}} \left[\sum_{e \in \mathcal{E}} \text{score}(e) - \lambda_{div} \sum_{e_i, e_j \in \mathcal{E}} \text{sim}(e_i, e_j) \right] \tag{10}$$

The combined objective is submodular but non-monotone, so the $(1 - 1/e)$ guarantee of Nemhauser et al. [1978] fails to apply. We use deterministic greedy selection with $\lambda_{div} = 0.1$, which empirically outperforms both random selection and pure similarity-based retrieval (Table 2).

Continuous GPU control via CUDA interception. We intercept CUDA kernel launches via `LD_PRELOAD`, implementing token-bucket throttling with rate ratio $\rho \in [0, 1]$:

$$\text{tokens}_\tau = T_{max} \cdot \rho \tag{11}$$

where each window τ (10ms) grants tokens proportional to ρ. Kernel launches consume tokens proportional to their parallelism (grid blocks); when exhausted, launches block until refill. This achieves three properties critical for effective RL: (1) *Fine-grained action space*: The mechanism supports continuous ρ, but SAIR uses bounded discrete increments $\Delta\rho_i \in \{-0.1, 0, +0.1, +0.2\}$ as

Algorithm 1 SAIR decision loop

Input: LLM policy P_{LLM}, context size M, reward threshold r_{min}, exploration $(\epsilon_0, \epsilon_{min}, \lambda)$, interval Δt

$\mathcal{D} \leftarrow \emptyset$; $\mathcal{P} \leftarrow \emptyset$; $\epsilon \leftarrow \epsilon_0$ ▷ experience buffer, Pareto frontier
for each decision round $t = 1, 2, \ldots$ (every Δt) **do**
  $x_t \leftarrow$ OBSERVECONTEXT() ▷ per-stage $(\mathbf{r}_i, q_i, u_i^{cpu}, u_i^{gpu})$, L_{p99}, and $\mathcal{P}$
  $\mathcal{E}_t \leftarrow$ SELECTEXPERIENCES$(\mathcal{D}, x_t, M)$ ▷ greedy Bayesian-surprise selection, Eqs. (9)–(10)
  if $u \sim \mathcal{U}(0,1) < \epsilon$ **then**
    $a_t \leftarrow$ RANDOMPROBE() ▷ forced exploration
  else
    $a_t \sim P_{LLM}(\cdot \mid x_t, \mathcal{E}_t, \texttt{constraints})$ ▷ in-context policy π_{SAIR}, Eq. (3)
  $a_t \leftarrow$ VALIDATE(a_t) ▷ clamp to $\mathcal{A}$; no-op during cooldown (Table 3)
  ACTUATE(a_t); wait settling window; measure L_t, C_t ▷ Kubernetes (CPU), rate-control socket (GPU)
  $r_t \leftarrow R(x_t, a_t; \mathcal{P})$ ▷ Pareto-shaped reward, Eq. (4), Algorithm 2
  if $r_t > r_{min}$ **then**
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x_t, a_t, r_t)\}$ ▷ positive-only memory
  $\mathcal{P} \leftarrow$ UPDATEFRONTIER$(\mathcal{P}, (L_t, C_t))$ ▷ keep non-dominated points
  $\epsilon \leftarrow \max(\epsilon_{min}, \lambda\,\epsilon)$ ▷ exploration decay

a safety layer; (2) *Immediate effect*: Rate changes via Unix socket take effect within milliseconds (measured overhead <2ms); (3) *Instant reversibility*: SAIR can increase or decrease ρ instantly, unlike replica scaling with asymmetric scale-up/down times.

GPU utilization is normalized to quota: $u_{quota} = \min(1, u_{actual}/\rho)$, ensuring Assumption 3.2 holds regardless of rate limit.

Positive-only episode filtering. We store only episodes with positive reward ($r > r_{min}$), spending limited context budget on informative successes; approximately 30% of decisions yield negative reward and are filtered out. The trade-off between omitting instructive failures and reducing noise is measured by the Store-All ablation (+12.1% P99; Table 2).

5 Theoretical analysis

The pipeline reaches a quasi-steady-state within each decision interval, so we analyze SAIR as a contextual bandit. The regret definition applies round-wise regardless of whether contexts are stochastic, adversarial, or history-dependent: at each round t we compare $\pi(x_t)$ to $a_t^* = \arg\max_a R(x_t, a)$ for that specific context. Full proofs and two further results (sample complexity for bottleneck identification, marginal-gain lemma) are in Appendix A.

5.1 Regret framework

Theorem 5.1 (SAIR Regret Bound)**.** *Under Assumptions 3.1–3.4, let ϵ_t be the exploration probability at round t. SAIR achieves:*

$$\textit{Regret}(T) \leq \sum_{t=1}^{T} (1-\epsilon_t)(\xi_t + \eta_t) + \left(\sum_{t=1}^{T} \epsilon_t + \delta_{LLM} T \right) R_{max} \tag{12}$$

where ξ_t is the retrieval coverage gap, η_t is the LLM selection error, and $\sum_t \epsilon_t$ accounts for exploration rounds.

Proof sketch. With probability ϵ_t the agent explores (at most R_{max} regret); otherwise the LLM policy incurs coverage gap ξ_t plus selection error η_t (with δ_{LLM} failure probability). Summing over T rounds gives the bound. Full proof in Appendix A.

Theorem 5.1 is a per-round regret decomposition: with constant η_t, δ_{LLM}, or ϵ_{min}, the bound grows linearly in T. Sublinear total regret requires $\sum_t \eta_t$, $\sum_t \delta_{LLM}$, and $\sum_t \epsilon_t = o(T)$; a geometric decay schedule for ϵ_t contributes $O(1)$, and ϵ_{min} is annealable. The practical value is that each term maps to a concrete SAIR mechanism with an empirical proxy: ξ_t to the No-ICL gap (+17% P99); η_t to the constant Pareto margin (Proposition 5.2), tightened by the validator and JSON-constrained output; the exploration cost to $\epsilon_{min} = 0.05$. The ablation in Section 6 measures the sign and magnitude of each term.

5.2 Component bounds

Proposition 5.2 (Pareto Reward: Frontier Separation)**.** *Consider the Pareto reward in Eq. 6 with non-dominated branch $\alpha + H(a)$ and dominated branch $\beta/(1 + d(a, \mathcal{P}))$, where $\alpha > \beta > 0$, normalized objectives $\tilde{L}, \tilde{C} \in [0, 1]$, and hypervolume taken with respect to reference point $(1, 1)$. Any non-dominated action receives reward at least α, any dominated action strictly less than β, and*

$$a \in \mathcal{P},\ b \notin \mathcal{P} \quad \Rightarrow \quad R_{pareto}(a) - R_{pareto}(b) > \alpha - \beta. \tag{13}$$

With normalized objectives, $H(a) \in [0, 1]$, so $R_{pareto}(a) \in [0,\ \alpha + 1]$, satisfying Assumption 3.1.

Proof. Non-dominated actions satisfy $R_{pareto}(a) = \alpha + H(a) \geq \alpha$ since $H(a) \geq 0$. Dominated actions have $d(b, \mathcal{P}) > 0$, so $R_{pareto}(b) = \beta/(1 + d(b, \mathcal{P})) < \beta$. The constant gap $\alpha - \beta$ lets the LLM separate non-dominated from dominated actions given sufficient context. Boundedness follows from $\tilde{L} = L/L_{max}$ and $\tilde{C} = C/C_{max}$. SAIR uses $\alpha = 1$ and $\beta = 0.8$ in all experiments (Appendix F); the values are calibrated above LLM reward-perception noise and below the magnitude of the latency/cost reward weights.

Lemma 5.3 (Retrieval Coverage Under Smoothness)**.** *If $\mathbb{E}[R(x, a)]$ is ℓ-Lipschitz in x (where x is z-score-normalized per the retrieval kernel) and $\mathcal{E}_t$ contains (x_e, a_e, r_e) with $\|x_e - x_t\| \leq d_t$ and a_e optimal at x_e, then $\xi_t \leq 2\ell d_t$ (proof in Appendix A).*

Corollary 5.4 (Sublinear Cumulative Coverage Regret)**.** Under Lemma 5.3, if contexts lie in a bounded d-dimensional subspace and $|\mathcal{D}|$ grows by one per round, then $d_t = O(|\mathcal{D}_t|^{-1/d})$ and $\sum_t \xi_t = O(\ell T^{1-1/d})$, sublinear in T.

The diversity term decreases expected retrieval distance while the similarity term keeps retrievals close to x_t, so SAIR's policy improves as the buffer grows. Theorem 5.1 thus accounts for performance: ξ_t is reduced by selection (Corollary 5.4); η_t by the JSON-constrained output, validator (Table 3), and the Pareto margin (Proposition 5.2); the exploration penalty by the decaying schedule. Ablation confirms that asymmetric Pareto weighting is critical (Equal Wt. -3.9%, Cost Focus $+27\%$ P99; Table 2).

6 Experiments

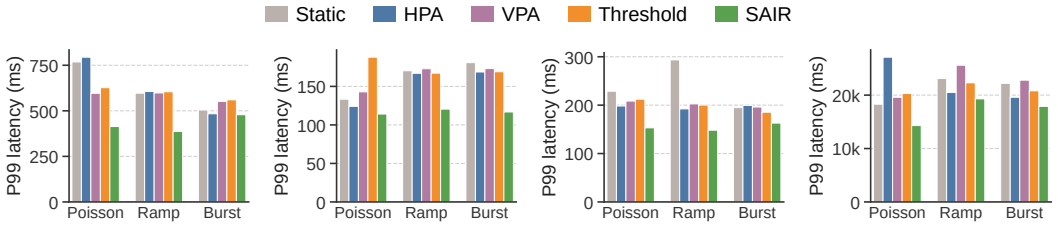
The 2-GPU testbed is the minimum configuration that exercises bottleneck migration across heterogeneous CPU/GPU stages; cluster-scale efficiency is orthogonal to per-pipeline correctness, since each PipelineAutoscaler is an independent control object. Experiments span four ML serving pipelines (latency 60 ms–18 s) on three workload patterns; each processes about 1M requests with 3 seeds (means reported). Inter-seed P99 variation is at most 5% (Image Classification), 8% (NLP and Text Generation), and 12% (Video Analysis), all below cross-baseline gaps reported below.

6.1 Experimental setup

Hardware. Kubernetes cluster with 2× NVIDIA RTX A6000 GPUs (49 GB each), 64 CPU cores, 256 GB RAM. For scale experiments (Section 6.5), a second node with 2 additional A6000 GPUs joins as a k3s agent, forming a 4-GPU, 2-node cluster.

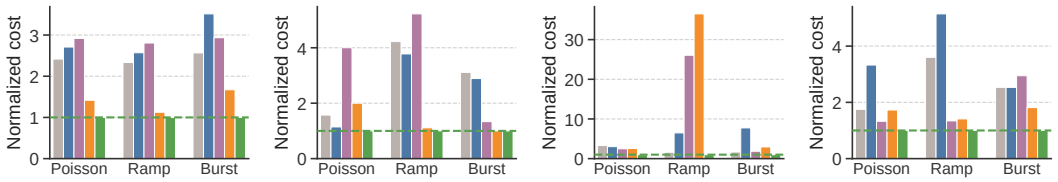
LLM backend and decision cost. Main results use GPT-5.1-mini (temperature 1.0, DSPy structured output). Each decision sends 4–6K input tokens and at most 300 output tokens; at $\Delta t = 30$ s this is about 120 calls per hour per pipeline, or $0.01 to $0.05 per pipeline-hour at list pricing. This is orders of magnitude below the savings in Table 5, so the LLM call cost is negligible in the cost comparison. We also cross-checked decisions against Claude Haiku and local Qwen-32B on Image Classification: the action proposals after the validator clamp were similar in distribution and produced P99 latencies within 5% of the GPT-5.1-mini run. The backend only affects η_t in Theorem 5.1.

Applications. We evaluate four ML serving pipelines with different computational profiles: *Image Classification* (MobileNetV2 with CPU-heavy preprocessing such as image decoding, resizing, and normalization, GPU inference, and lightweight postprocessing), *NLP Analysis* (BERT-based text analysis with a balanced CPU/GPU workload and dynamic bottleneck migration), *Text Generation* (a Transformer decoder whose GPU-intensive autoregressive generation creates variable latency),



(a) ImgCls (b) NLP (c) TextGen (d) Video

Figure 2: P99 latency across applications and workload patterns. Lower is better; SAIR (rightmost bar in each subplot) wins or ties on every configuration.



(a) ImgCls (b) NLP (c) TextGen (d) Video

Figure 3: Normalized effective cost (SAIR=1). Values >1 indicate higher cost than SAIR. Same legend as Figure 2.

and *Video Analysis* (multi-frame processing with high memory requirements and a 10–20 s latency profile).

Workload patterns. Three patterns stress different autoscaling capabilities: *Poisson* (steady-state with random arrivals), *Ramp* (gradually increasing load), and *Burst* (periodic traffic spikes).

Baselines. We compare against four production autoscalers: *Static* (one replica per stage), *HPA-CPU* (Kubernetes HPA with a 70% CPU utilization target and a 60 s stabilization window), *VPA* (Vertical Pod Autoscaler for resource sizing in `Auto` update mode), and *Threshold* (rule-based P99 latency thresholds of 100 ms for CPU and 200 ms for GPU stages, with a 60 s cooldown). All baselines use per-application tuned parameters; HPA and Threshold targets were swept over $\{50, 60, 70, 80\}\%$ and $\{50, 100, 200, 500\}$ms respectively, selecting best-performing values. Learning-based controllers (FIRM [Qiu et al., 2020], AWARE [Qiu et al., 2023]) require offline pretraining, making them impractical for zero-shot evaluation; we train faithful implementations to convergence and evaluate their frozen policies in Section 6.4.

Metrics. (1) P99 end-to-end latency; (2) effective resource cost per 1M requests. We distinguish two cost models. *Billable cost* reflects whole-GPU pricing: $C_{bill} = \sum_{i \in \text{CPU}} n_i c_i p_{cpu} + \sum_{i \in \text{GPU}} n_i p_{gpu}$, where each GPU incurs full cost regardless of utilization. *Effective cost* accounts for GPU rate control: $C_{eff} = \sum_{i \in \text{CPU}} n_i c_i p_{cpu} + \sum_{i \in \text{GPU}} n_i \rho_i p_{gpu}$, where throttling to $\rho_i < 1$ frees capacity for co-located workloads. We report C_{eff} using AWS on-demand pricing (p3.2xlarge GPU, m5.xlarge CPU); this metric is appropriate when GPUs are shared via time-slicing or MIG.

6.2 Main results

SAIR achieves the lowest P99 latency in every configuration (Figure 2): Image Classification 35–46%, NLP Analysis 8–31%, Text Generation 12–50%, and Video Analysis 6–47% (the only method that completes the high-latency pipeline within SLA on Burst). The margin over HPA-CPU is largest on workloads where the bottleneck is GPU-bound (NLP, TextGen), which is direct evidence that single-metric reactive controllers produce incorrect scaling under the cross-stage coupling of Section 3. The margin over Threshold is largest on Burst patterns, where Bayesian-surprise retrieval over-represents prior burst episodes and lets the LLM reuse a successful response pattern instead of recomputing it.

SAIR also achieves the best effective cost (Figure 3): 30–60% lower than the next-best baseline and up to 81% lower than HPA on Video Analysis. The Video gain comes primarily from rate-control;

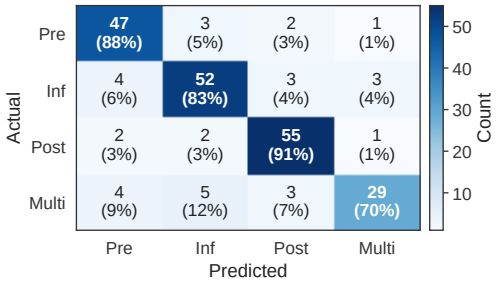


Figure 4: Bottleneck detection on 200 injected scenarios. Overall accuracy 86%; 88–91% single-stage, 70% multi-stage. Per-class precision/recall in Appendix H.

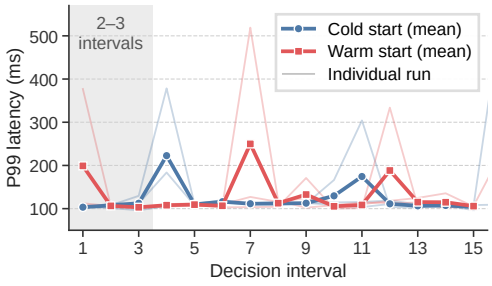


Figure 5: Cold-start convergence: per-interval P99 of 3 cold-started and 3 warm-started runs and their means. Cold start reaches warm-start P99 within 2–3 decision intervals.

Table 2: Ablation on Text Generation, 3 seeds. ΔT is throughput change, ΔL is P99 latency change (%), both relative to Full SAIR.

Configuration	Throughput (rps)	P99 (ms)	ΔT (%)	ΔL (%)
Full SAIR	**84.0**	**902**	–	–
No ICL	82.6	1054	−1.7	+16.8
Store All	83.0	1012	−1.2	+12.1
Linear Pen.	80.6	962	−4.1	+6.6
No Bonus	81.7	1006	−2.7	+11.4
Equal Wt.	80.7	936	−3.9	+3.7
Cost Focus	81.4	1149	−3.1	+27.4
Wt. Sum	83.5	976	−0.6	+8.2
Pre Only	81.9	1054	−2.5	+16.8

in single-tenant mode the gap reduces to the smaller P99-vs-replica margin (Appendix G) but the latency advantage is unaffected.

Effective-cost validation. We validated the effective-cost model by running a co-tenant at maximum drive on the same GPU as a throttled primary (Text Generation at 57 rps). At moderate load, the co-tenant achieved 97.5–98.9% of standalone throughput at every ρ setting, with the primary's P99 unchanged. Under saturation, the primary's throughput tracked ρ near-proportionally (measured shares 1.00/0.78/0.59/0.38/0.14) while the co-tenant held standalone throughput. Interception overhead: 0.6%; a second machine replicated the result (98.9–99.6%, overhead 0.07%). Full tables in Appendix L. The 97% figure is a best case (the co-tenant consumes freed capacity at low primary utilization); representative savings are 30–60% over the next-best baseline (Table 5).

6.3 Bottleneck detection and ablation study

A pre-condition for the regret bound is that the agent can identify which stage to scale. We evaluate bottleneck detection on 200 scenarios with ground-truth bottlenecks injected by throttling individual stages. Figure 4 reports the confusion matrix: 88–91% accuracy on single-stage bottlenecks enables the targeted scaling that accounts for the main-results latency advantage. Multi-stage accuracy of 70% (2.8× the 25% random baseline) reflects the discrete action space; $R_{proactive}$ compensates by rewarding multi-stage actions during severe violations.

For the ablation (Table 2), 9 configurations all degrade from Full SAIR. Reward shaping is the largest factor: Linear Pen. (-4.1%) performs worst because removing the quadratic shape eliminates the larger penalty after a violation. Equal Wt. (-3.9%) and Cost Focus (-3.1%, $+27\%$ P99) show the latency/cost asymmetry matters beyond the magnitudes. Pre-Only (-2.5%) cannot follow the bottleneck once preprocessing scaling shifts the queue downstream.

Isolating the ICRL contribution. No-ICL (+16.8% P99) removes experience retrieval while keeping the same LLM, reward, and validator; Store-All (+12.1%) keeps retrieval but removes positive-only filtering. These deltas confirm that in-context experience, and its curation, account for the

performance gap. A nearest-neighbor replay baseline achieves steady-state P99 comparable to cold-started SAIR under stationary load (mean 125.7 vs. 136.4 ms, n=3); under a 1.7× workload shift, both configurations show comparable P99, but SAIR produces 4–8× fewer request timeouts in 2 of 3 paired runs, attributable to the LLM's ability to compose novel actions beyond replaying stored ones. The Wt. Sum row (+8.2% P99) isolates the Pareto reward structure by replacing it with a weighted sum of the same component rewards.

6.4 Cold-start convergence and offline baselines

The zero-shot deployment setting means SAIR starts with an empty experience buffer. We quantify the cost of this cold start and the alternative of offline-trained policies.

Warm vs. cold start. We run 3 interleaved pairs of warm-started (buffer pre-seeded with 13 episodes) and cold-started SAIR on Text Generation at 100 rps, 300 s per run. Full-run P99: cold 136.4 ms vs. warm 132.7 ms, a 2.8% gap within ±15% inter-run spread. Every cold run reached the warm-started steady-state P99 within 46.6–58.9 s (2–3 decision intervals; Figure 5). Per-run values are in Appendix J.

Offline-trained RL baselines. We implemented FIRM (DDPG, 10-dim state, 15 actions) and AWARE (PPO, 13-dim state, 7 actions) with their native reward functions, trained each to convergence in an M/M/1 queueing-model simulator per pipeline (seed 42), and evaluated the frozen policies on the live testbed (120 s × 3 reps × {Poisson, Burst}). FIRM's DDPG policy never scaled from its initial allocation across all runs. AWARE's PPO policy took sporadic scaling actions under burst load but produced burst-tail P99 up to 556 ms (2.8× the 200 ms SLA). Both policies transferred as non-adaptive policies that held minimal allocations, a measured instance of the sim-to-real gap that zero-shot deployment avoids. Full protocol in Appendix K.

6.5 System scale

We grew the system to a 2-node, 4-GPU deployment. A 240 rps Poisson load that saturates the single-GPU inference stage (P99 27.0 s) is served at P99 107 ms with 4 cross-node replicas. SAIR controlled this deployment with policy logic, action schema, and prompts unchanged, sustaining full offered load in 3/3 runs at ≥99.99% success. Control-plane concurrency (N=2, 8, 16) shows per-decision latency flat in N (N=16: mean 0.64 s, 416 decisions), cost linear. Per-run details in Appendix M.

7 Discussion and conclusion

Cold start and failure modes. With an empty buffer, ϵ_t-greedy probes, validator caps, and Bayesian surprise upweighting populate the buffer within $O(N \log N/\Delta^2)$ probes; measured cold-start overhead is 2.8% of full-run P99, converging in 2–3 decision intervals (Section 6.4). Three failure modes remain: (i) stable Poisson load, where the P99 advantage drops below 10%; (ii) the first 2–3 decisions, dominated by exploration; (iii) multi-stage detection at 70%, where misidentifications are absorbed by validator caps.

Prefill/decode serving. Disaggregated LLM serving [Zhong et al., 2024, Patel et al., 2024] maps to SAIR's pipeline abstraction (prefill produces KV-cache entries, decode consumes them via a handoff queue). Adapting SAIR requires KV-transfer and memory-pressure signals, validation that Assumption 3.2 holds under batching, and benchmarking the CUDA interception layer on newer GPU generations; we leave these extensions to future work.

Scope, limitations, conclusion. SAIR targets ≥ 3 heterogeneous stages with bottlenecks migrating within $O(\Delta t)$ absent offline traces. Effective-cost savings of up to 97% require GPU sharing (validated in Section 6); in single-tenant deployments, savings are 30–60% over next-best. Cross-cluster scheduling [Su et al., 2026] and $\Delta t < 10$ s are out of scope. Overall, an LLM controller autoscales multi-stage pipelines from online experience alone, with best-or-tied P99 in every configuration, and the regret framework transfers beyond autoscaling.

Acknowledgments and Disclosure of Funding

We thank the anonymous reviewers and the area chair for their feedback. This work is supported in part by the National Science Foundation under Award #2530909 and by NEUTC.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual.*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Daniel Crankshaw, Xin Wang, Giulio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. Clipper: A Low-Latency Online Prediction Serving System. In Aditya Akella and Jon Howell, editors, *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 613–627. USENIX Association, 2017. ISBN 978-1-931971-37-9. URL https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw.

Daniel Crankshaw, Gur-Eyal Sela, Xiangxi Mo, Corey Zumar, Ion Stoica, Joseph Gonzalez, and Alexey Tumanov. InferLine: latency-aware provisioning and scaling for prediction serving pipelines. In Rodrigo Fonseca, Christina Delimitrou, and Beng Chin Ooi, editors, *SoCC '20: ACM Symposium on Cloud Computing, Virtual Event, USA, October 19-21, 2020*, pages 477–491. ACM, 2020. ISBN 978-1-4503-8137-6. doi: 10.1145/3419111.3421285. URL https://doi.org/10.1145/3419111.3421285.

Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.*, 6(2):182–197, 2002. doi: 10.1109/4235.996017. URL https://doi.org/10.1109/4235.996017.

Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. What Can Transformers Learn In-Context? A Case Study of Simple Function Classes. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022.*, 2022. ISBN 9781713871088. URL http://papers.nips.cc/paper_files/paper/2022/hash/c529dba08a146ea8d6cf715ae8930cbe-Abstract-Conference.html.

Jashwant Raj Gunasekaran, Cyan Subhra Mishra, Prashanth Thinakaran, Bikash Sharma, Mahmut Taylan Kandemir, and Chita R. Das. Cocktail: A Multidimensional Optimization for Model Serving in Cloud. In Amar Phanishayee and Vyas Sekar, editors, *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022*, pages 1041–1057. USENIX Association, 2022. ISBN 978-1-939133-27-4. URL https://www.usenix.org/conference/nsdi22/presentation/gunasekaran.

Conor F. Hayes, Roxana Radulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M. Zintgraf, Richard Dazeley, Fredrik Heintz, Enda Howley, Athirai A. Irissappane, Patrick Mannion, Ann Nowé, Gabriel de Oliveira Ramos, Marcello Restelli, Peter Vamplew, and Diederik M. Roijers. A practical guide to multi-objective reinforcement learning and planning. *Auton. Agents Multi Agent Syst.*, 36(1):26, 2022. doi: 10.1007/s10458-022-09552-y. URL https://doi.org/10.1007/s10458-022-09552-y.

Laurent Itti and Pierre Baldi. Bayesian Surprise Attracts Human Attention. In *Advances in Neural Information Processing Systems 18 [Neural Information Processing*

Systems, NIPS 2005, December 5-8, 2005, Vancouver, British Columbia, Canada], pages 547–554, 2005. URL https://proceedings.neurips.cc/paper/2005/hash/0172d289da48c48de8c5ebf3de9f7ee1-Abstract.html.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling Declarative Language Model Calls into State-of-the-Art Pipelines. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=sY5N0zY5Od.

Akshay Krishnamurthy, Keegan Harris, Dylan J. Foster, Cyril Zhang, and Aleksandrs Slivkins. Can large language models explore in-context? In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024.*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/d951f73c521d069fefbb73396df01424-Abstract-Conference.html.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient Memory Management for Large Language Model Serving with PagedAttention. In Jason Flinn, Margo I. Seltzer, Peter Druschel, Antoine Kaufmann, and Jonathan Mace, editors, *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pages 611–626. ACM, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Guangchen Lan, Dong-Jun Han, Abolfazl Hashemi, Vaneet Aggarwal, and Christopher Brinton. Asynchronous Federated Reinforcement Learning with Policy Gradient Updates: Algorithm Design and Convergence Analysis. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025a. URL https://openreview.net/forum?id=5DUekOKWcS.

Guangchen Lan, Huseyin A. Inan, Sahar Abdelnabi, Janardhan Kulkarni, Lukas Wutschitz, Reza Shokri, Christopher Brinton, and Robert Sim. Contextual Integrity in LLMs via Reasoning and Reinforcement Learning. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025.*, 2025b. URL http://papers.nips.cc/paper_files/paper/2025/hash/968591afcfe4ba889eccffb6a4ca2d45-Abstract-Conference.html.

Guangchen Lan, Sipeng Zhang, Tianle Wang, Yuwei Zhang, Daoan Zhang, Xinpeng Wei, Xiaoman Pan, Hongming Zhang, Dong-Jun Han, and Christopher G. Brinton. MaPPO: Maximum a Posteriori Preference Optimization with Prior Knowledge. *CoRR*, abs/2507.21183, July 2025c. doi: 10.48550/arXiv.2507.21183. URL https://doi.org/10.48550/arXiv.2507.21183.

Michael Laskin, Luyu Wang, Junhyuk Oh, Emilio Parisotto, Stephen Spencer, Richie Steigerwald, DJ Strouse, Steven Stenberg Hansen, Angelos Filos, Ethan Brooks, Maxime Gazeau, Himanshu Sahni, Satinder Singh, and Volodymyr Mnih. In-context Reinforcement Learning with Algorithm Distillation. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=hy0a5MMPUv.

Jonathan Lee, Annie Xie, Aldo Pacchiano, Yash Chandak, Chelsea Finn, Ofir Nachum, and Emma Brunskill. Supervised Pretraining Can Learn In-Context Reinforcement Learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023.*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/8644b61a9bc87bf7844750a015feb600-Abstract-Conference.html.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What Makes Good In-Context Examples for GPT-3? In Eneko Agirre, Marianna Apidianaki, and Ivan Vulic, editors, *Proceedings of Deep Learning Inside Out: The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures, DeeLIO@ACL 2022, Dublin, Ireland and Online, May 27, 2022*, pages 100–114. Association for Computational Linguistics, 2022. ISBN 978-1-955917-32-2. doi: 10.18653/v1/2022.deelio-1.10. URL https://doi.org/10.18653/v1/2022.deelio-1.10.

Cunchi Lv, Xiao Shi, Zhengyu Lei, Jinyue Huang, Wenting Tan, Xiaohui Zheng, and Xiaofang Zhao. Dilu: Enabling GPU Resourcing-on-Demand for Serverless DL Serving via Introspective Elasticity. In Lieven Eeckhout, Georgios Smaragdakis, Kaitai Liang, Adrian Sampson, Martha A. Kim, and Christopher J. Rossbach, editors, *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1, ASPLOS 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*, pages 311–325. ACM, 2025. ISBN 979-8-4007-0698-1. doi: 10.1145/3669940.3707251. URL https://doi.org/10.1145/3669940.3707251.

George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.*, 14(1):265–294, 1978. doi: 10.1007/BF01588971. URL https://doi.org/10.1007/BF01588971.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Í nigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *51st ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2024, Buenos Aires, Argentina, June 29 - July 3, 2024*, pages 118–132. IEEE, 2024. ISBN 979-8-3503-2658-1. doi: 10.1109/ISCA59077.2024.00019. URL https://doi.org/10.1109/ISCA59077.2024.00019.

Archit Patke, Dhemath Reddy, Saurabh Jha, Haoran Qiu, Christian Pinto, Chandra Narayanaswami, Zbigniew Kalbarczyk, and Ravishankar K. Iyer. Queue Management for SLO-Oriented Large Language Model Serving. In *Proceedings of the 2024 ACM Symposium on Cloud Computing, SoCC 2024, Redmond, WA, USA, November 20-22, 2024*, pages 18–35. ACM, 2024. ISBN 979-8-4007-1286-9. doi: 10.1145/3698038.3698523. URL https://doi.org/10.1145/3698038.3698523.

Archit Patke, Dhemath Reddy, Saurabh Jha, Chandra Narayanaswami, Zbigniew Kalbarczyk, and Ravishankar K. Iyer. Hierarchical Autoscaling for Large Language Model Serving with Chiron. *CoRR*, abs/2501.08090, January 2025. doi: 10.48550/arXiv.2501.08090. URL https://doi.org/10.48550/arXiv.2501.08090.

Haoran Qiu, Subho S. Banerjee, Saurabh Jha, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. FIRM: An Intelligent Fine-grained Resource Management Framework for SLO-Oriented Microservices. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020.*, pages 805–825. USENIX Association, 2020. ISBN 978-1-939133-19-9. URL https://www.usenix.org/conference/osdi20/presentation/qiu.

Haoran Qiu, Weichao Mao, Chen Wang, Hubertus Franke, Alaa Youssef, Zbigniew T. Kalbarczyk, Tamer Basar, and Ravishankar K. Iyer. AWARE: Automate Workload Autoscaling with Reinforcement Learning in Production Cloud Systems. In Julia Lawall and Dan Williams, editors, *Proceedings of the 2023 USENIX Annual Technical Conference, USENIX ATC 2023, Boston, MA, USA, July 10-12, 2023.*, pages 387–402. USENIX Association, 2023. URL https://www.usenix.org/conference/atc23/presentation/qiu-haoran.

Krzysztof Rzadca, Pawel Findeisen, Jacek Swiderski, Przemyslaw Zych, Przemyslaw Broniek, Jarek Kusmierek, Pawel Nowak, Beata Strack, Piotr Witusowski, Steven Hand, and John Wilkes. Autopilot: workload autoscaling at Google. In Angelos Bilas, Kostas Magoutis, Evangelos P. Markatos, Dejan Kostic, and Margo I. Seltzer, editors, *EuroSys '20: Fifteenth EuroSys Conference 2020, Heraklion, Greece, April 27-30, 2020*, pages 16:1–16:16. ACM, 2020. ISBN 978-1-4503-6882-7. doi: 10.1145/3342195.3387524. URL https://doi.org/10.1145/3342195.3387524.

Kefan Song, Amir Moeini, Peng Wang, Lei Gong, Rohan Chandra, Yanjun Qi, and Shangtong Zhang. Reward Is Enough: LLMs Are In-Context Reinforcement Learners. *CoRR*, abs/2506.06303,

June 2025. doi: 10.48550/arXiv.2506.06303. URL https://doi.org/10.48550/arXiv.2506.06303.

Jianchang Su, Masoud Rahimi Jafari, Yifan Zhang, and Wei Zhang. FastMatch: Enhancing Data Pipeline Efficiency for Accelerated Distributed Training. In *42nd IEEE International Conference on Computer Design, ICCD 2024, Milan, Italy, November 18-20, 2024*, pages 239–246. IEEE, 2024. ISBN 979-8-3503-8040-8. doi: 10.1109/ICCD63220.2024.00044. URL https://doi.org/10.1109/ICCD63220.2024.00044.

Jianchang Su, Wenhui Zhang, Yifan Zhang, Kexin Chu, Hao Guo, Youyou Lu, and Wei Zhang. TrustWeave: Integrity Measurement and Attestation For Multi-Cloud LLMs. In Antonio Barbalace, Luo Mai, Roxana Geambasu, and Peter R. Pietzuch, editors, *Proceedings of the 21st European Conference on Computer Systems, EuroSys 2026, McEwan Hall/The University of Edinburgh, Edinburgh, Scotland, UK, April 27-30, 2026*, pages 641–656. ACM, 2026. ISBN 979-8-4007-2212-7. doi: 10.1145/3767295.3803586. URL https://doi.org/10.1145/3767295.3803586.

Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. Llumnix: Dynamic Scheduling for Large Language Model Serving. In Ada Gavrilovska and Douglas B. Terry, editors, *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024.*, pages 173–191. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/sun-biao.

The Kubernetes Authors. Kubernetes, 2026. URL https://kubernetes.io/. Website last updated 2026-08-17; accessed 2026-10-19.

Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. Gandiva: Introspective Cluster Scheduling for Deep Learning. In Andrea C. Arpaci-Dusseau and Geoff Voelker, editors, *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018.*, pages 595–610. USENIX Association, 2018. URL https://www.usenix.org/conference/osdi18/presentation/xiao.

Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. AntMan: Dynamic Scaling on GPU Clusters for Deep Learning. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020.*, pages 533–548. USENIX Association, 2020. ISBN 978-1-939133-19-9. URL https://www.usenix.org/conference/osdi20/presentation/xiao.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An Explanation of In-context Learning as Implicit Bayesian Inference. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=RdJVFCHjUMI.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A Distributed Serving System for Transformer-Based Generative Models. In Marcos K. Aguilera and Hakim Weatherspoon, editors, *16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2022, Carlsbad, CA, USA, July 11-13, 2022*, pages 521–538. USENIX Association, 2022. ISBN 978-1-939133-28-1. URL https://www.usenix.org/conference/osdi22/presentation/yu.

Peifeng Yu and Mosharaf Chowdhury. Fine-Grained GPU Sharing Primitives for Deep Learning Applications. In Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze, editors, *Proceedings of the Third Conference on Machine Learning and Systems, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. mlsys.org, 2020. URL https://proceedings.mlsys.org/paper_files/paper/2020/hash/d9cd83bc91b8c36a0c7c0fcca59228f2-Abstract.html.

Shaokun Zhang, Xiaobo Xia, Zhaoqing Wang, Ling-Hao Chen, Jiale Liu, Qingyun Wu, and Tongliang Liu. IDEAL: Influence-Driven Selective Annotations Empower In-Context Learners in Large Language Models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=Spp2i1hKwV.

Yanqi Zhang, Weizhe Hua, Zhuangzhuang Zhou, G. Edward Suh, and Christina Delimitrou. Sinan: ML-based and QoS-aware resource management for cloud microservices. In Tim Sherwood, Emery D. Berger, and Christos Kozyrakis, editors, *ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual Event, USA, April 19-23, 2021*, pages 167–181. ACM, 2021. ISBN 978-1-4503-8317-2. doi: 10.1145/3445814.3446693. URL https://doi.org/10.1145/3445814.3446693.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. In Ada Gavrilovska and Douglas B. Terry, editors, *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024.*, pages 193–210. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin.

Eckart Zitzler and Lothar Thiele. Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study. In A. E. Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature - PPSN V, 5th International Conference, Amsterdam, The Netherlands, September 27-30, 1998, Proceedings*, volume 1498 of *Lecture Notes in Computer Science*, pages 292–304. Springer, 1998. ISBN 3-540-65078-4. doi: 10.1007/BFb0056872. URL https://doi.org/10.1007/BFb0056872.

A Proof details

A.1 Proof of Theorem 5.1 (SAIR regret bound)

Proof. Let $E_t \in \{0, 1\}$ indicate whether round t is a forced exploration round (probability ϵ_t). When $E_t = 1$, the agent takes a random probe and incurs at most R_{max} regret (Assumption 3.1).

When $E_t = 0$ (LLM policy), the per-round regret decomposes as:

$$\mathbb{E}[R(x_t, a_t^*)] - \mathbb{E}[R(x_t, a_t)] \tag{14}$$

$$= \underbrace{\mathbb{E}[R(x_t, a_t^*)] - \max_{a \in \mathcal{A}(\mathcal{E}_t)} \mathbb{E}[R(x_t, a)]}_{\xi_t \text{ (coverage gap, Assumption 3.3)}} \tag{15}$$

$$+ \underbrace{\max_{a \in \mathcal{A}(\mathcal{E}_t)} \mathbb{E}[R(x_t, a)] - \mathbb{E}[R(x_t, a_t)]}_{\leq \eta_t \text{ w.p. } 1-\delta_{LLM} \text{ (Assumption 3.4)}} \tag{16}$$

With probability $1 - \delta_{LLM}$, the LLM-policy regret is at most $\xi_t + \eta_t$. With probability δ_{LLM}, the regret is bounded by R_{max}.

Combining exploration and exploitation rounds over T rounds:

$$\begin{aligned} \text{Regret}(T) &\leq \textstyle\sum_t \left[\epsilon_t R_{max} + (1 - \epsilon_t)((\xi_t + \eta_t) + \delta_{LLM} R_{max})\right] \\ &\leq \textstyle\sum_t (1 - \epsilon_t)(\xi_t + \eta_t) + (\sum_t \epsilon_t + \delta_{LLM} T) R_{max} \end{aligned} \tag{17}$$

The decomposition shows that SAIR's regret depends on three terms: (i) retrieval coverage quality ξ_t, reducible through Bayesian-surprise selection; (ii) LLM selection quality η_t, reducible through prompt constraints and action validation; and (iii) exploration cost $\sum_t \epsilon_t \cdot R_{max}$, controlled by decaying ϵ_t. □

A.2 Proof of Proposition 5.2 (frontier separation)

Proof. We normalize objectives to $\tilde{L} = L/L_{max} \in [0, 1]$ and $\tilde{C} = C/C_{max} \in [0, 1]$, and compute hypervolume with respect to reference point $(1, 1)$. Let $\alpha > \beta > 0$ be the constants in Eq. 6.

Non-dominated actions ($a \in \mathcal{P}$): $R_{pareto}(a) = \alpha + H(a)$. Since $H(a) \geq 0$, $R_{pareto}(a) \geq \alpha$. With normalized objectives, $H(a) \leq 1$, so $R_{pareto}(a) \leq \alpha + 1$.

Dominated actions ($b \notin \mathcal{P}$): $R_{pareto}(b) = \beta/(1 + d(b, \mathcal{P}))$. Since $d(b, \mathcal{P}) > 0$, $R_{pareto}(b) < \beta$.

Separation margin: For any $a \in \mathcal{P}$ and $b \notin \mathcal{P}$,

$$R_{pareto}(a) - R_{pareto}(b) > \alpha - \beta. \tag{18}$$

Boundedness: $R_{pareto} \in [0,\ \alpha{+}1]$, so the total reward has $R_{max} = \alpha{+}1$, satisfying Assumption 3.1. SAIR uses $\alpha = 1$, $\beta = 0.8$ (Appendix F). □

A.3 Bottleneck detection sample complexity

Theorem A.1 (Bottleneck Detection Sample Complexity)**.** *Under Assumption 3.2, suppose the forced exploration probes in Algorithm 1 generate m scaling experiments per stage under stationary reward distributions. Then SAIR correctly identifies the bottleneck stage with probability at least $1 - \delta$ using $m = O(R_{max}^2 N^2/\Delta^2 \cdot \log(N/\delta))$ probes.*

The empirical 86% accuracy in Section 6 is consistent with this bound for $N{=}4$.

Proof. The proof applies to the *forced exploration subsequence* of Algorithm 1, where the agent takes random probing actions with probability ϵ_t.

Step 1: Signal from probes. Under Assumption 3.2, probing the true bottleneck $\phi(x)$ yields reward improvement at least Δ, while probing non-bottleneck stage $j \neq \phi(x)$ yields improvement at most $\Delta/2$. We assume reward distributions are stationary during the probing period.

Step 2: Concentration for stage identification. After m probes scaling stage i, let $\hat{r}_i$ be the empirical mean reward improvement. By Hoeffding's inequality for bounded rewards (Assumption 3.1):

$$\mathbb{P}\left[|\hat{r}_i - \mathbb{E}[r_i]| > \frac{\Delta}{4}\right] \leq 2\exp\left(-\frac{m\Delta^2}{8R_{max}^2}\right) \tag{19}$$

Step 3: Union bound over stage pairs. To correctly identify the bottleneck, we need $\hat{r}_{\phi(x)} > \hat{r}_j + \Delta/2$ for all $j \neq \phi(x)$. With concentration $\Delta/4$ for each stage, this holds when true gaps are at least $\Delta/2$.

Union bound over all $\binom{N}{2} \leq N^2/2$ pairs gives failure probability:

$$\mathbb{P}[\text{error}] \leq N^2 \cdot \exp\left(-\frac{m\Delta^2}{8R_{max}^2}\right) \tag{20}$$

Setting this to target failure probability δ and solving for m:

$$m = O\left(\frac{R_{max}^2 N^2}{\Delta^2} \log \frac{N}{\delta}\right) \tag{21}$$

The total number of decision steps needed is $T_{explore} = m \cdot N/\bar{\epsilon}$ where $\bar{\epsilon}$ is the average exploration rate, since each probe is assigned to a stage uniformly. □

A.4 Bottleneck marginal-gain lemma

Lemma A.2 (Bottleneck Marginal Gain)**.** *In a tandem queueing system with stages having utilizations $u_1, \ldots, u_N$, the stage $i^* = \arg\max_i u_i$ with highest utilization yields the largest marginal latency reduction when scaled.*

This standard queueing-theoretic result justifies Assumption 3.2 in the main body: pipeline bottlenecks create observable symptoms (high utilization, queue buildup) that distinguish them from non-bottleneck stages.

A.5 Proof of Lemma 5.3 (retrieval coverage under smoothness)

Proof. Let $a_t^* \in \arg\max_a \mathbb{E}[R(x_t, a)]$. By ℓ-Lipschitz continuity of $\mathbb{E}[R(\cdot, a)]$ and the optimality of a_e at x_e, $\mathbb{E}[R(x_t, a_t^*)] \leq \mathbb{E}[R(x_e, a_t^*)] + \ell d_t \leq \mathbb{E}[R(x_e, a_e)] + \ell d_t \leq \mathbb{E}[R(x_t, a_e)] + 2\ell d_t$. Since $a_e \in \mathcal{A}(\mathcal{E}_t)$, $\max_{a \in \mathcal{A}(\mathcal{E}_t)} \mathbb{E}[R(x_t, a)] \geq \mathbb{E}[R(x_t, a_e)]$, so $\xi_t \leq 2\ell d_t$. □

B Bayesian-surprise score as information-gain proxy

We justify the leave-one-out Bayesian-surprise score (Eq. 7) as a proxy for information gain under a simple Gaussian model.

Lemma B.1 (Bayesian-Surprise/Information-Gain Connection)**.** *Suppose rewards in a local neighborhood of context space follow $r \sim \mathcal{N}(\mu, \sigma^2)$ with conjugate prior $\mu \sim \mathcal{N}(\mu_0, \tau^2)$. Let $\mathcal{D}_{-e}$ denote the buffer excluding experience e, and let μ_{-e} be the posterior mean given $\mathcal{D}_{-e}$. Then the KL divergence between posteriors (a standard measure of Bayesian surprise) satisfies:*

$$D_{KL}\big(p(\mu \mid \mathcal{D}) \,\|\, p(\mu \mid \mathcal{D}_{-e})\big) \propto (r_e - \mu_{-e})^2. \tag{22}$$

Therefore, the leave-one-out residual $|r_e - \mathbb{E}[r \mid \mathcal{D}_{-e}]|$ is monotone in information gain under this model.

Proof. Under the conjugate Gaussian model, the posterior given n observations is $\mathcal{N}(\mu_n, \sigma_n^2)$ where $\mu_n = (\tau^{-2}\mu_0 + \sigma^{-2}\sum r_i)/(\tau^{-2} + n\sigma^{-2})$. The KL divergence between two Gaussians is $D_{KL} = \frac{1}{2}[(\mu_n - \mu_{n-1})^2/\sigma_n^2 + \sigma_{n-1}^2/\sigma_n^2 - 1 + \log(\sigma_n^2/\sigma_{n-1}^2)]$. Since σ_n^2 and σ_{n-1}^2 depend only on n (not on observations), the only observation-dependent term is $(\mu_n - \mu_{n-1})^2 \propto (r_e - \mu_{-e})^2$. □

This provides theoretical grounding for the score: experiences with large leave-one-out residuals correspond to high information gain about the local reward structure, making them most useful for in-context RL.

C System implementation details

Kubernetes operator. SAIR is implemented as a Kubernetes operator using the Kopf framework. The operator watches PipelineAutoscaler custom resources and manages the SAIR agent lifecycle. The agent runs as a sidecar container alongside the pipeline stages.

Metrics collection. Context observation uses three sources: (1) GPU metrics via the `LD_PRELOAD` library that intercepts CUDA calls and reports active/idle time to a Unix socket; (2) CPU metrics via cgroup accounting from the Kubernetes metrics API; (3) queue depth via the monitor endpoint.

Action execution. CPU stage scaling uses Kubernetes Deployment API with rolling updates. GPU rate control uses direct Unix socket communication with the rate-limiting library, bypassing Kubernetes for millisecond response time.

LLM integration. SAIR can use any model such as GPT-5.1-mini, Claude Haiku, or local Qwen-32B with temperature 1.0 (required for reasoning models) and structured output via DSPy. The prompt includes: current context (metrics), selected experiences (5–15 episodes), and constraints (resource limits, cost budget). The LLM returns a structured JSON action that is parsed and validated before execution.

D Action validator and constraints

Before executing any LLM-proposed action, SAIR applies the following validation rules to ensure safety and feasibility:

If the LLM proposes an action violating any constraint, the validator clamps the action to the nearest feasible value. For example, a proposed $\Delta n = +3$ is clamped to $+2$. Actions during cooldown periods are set to no-op for the affected stage.

E Reward computation pseudocode

Table 3: Action validation constraints applied to every LLM decision.

Constraint	Value
Max replica change $\lvert\Delta n_i\rvert$ per step (all)	≤ 2
Max CPU change $\lvert\Delta c_i\rvert$ per step, γ_c (CPU)	500 m
Max memory change $\lvert\Delta m_i\rvert$ per step, γ_m (CPU)	256 MB
Max GPU rate change $\lvert\Delta \rho_i\rvert$ per step (GPU)	≤ 0.2
Min GPU rate ratio ρ_{min} (GPU)	0.1
Max replicas per stage n_{max}	8
Cooldown between scale-ups	60 s
Cooldown between scale-downs	120 s
Min replicas per stage n_{min}	1

Algorithm 2 Pareto-shaped reward $R(x, a; \mathcal{P})$

Input: context x, action a, frontier $\mathcal{P}$, SLA target T_{SLA}, weights w_L=0.7, w_C=0.3, $w_{proactive}$=0.3
Output: shaped reward $R \in [-R_{max}, R_{max}]$

$(L_{before}, C_{before}) \leftarrow$ P99 latency and cost before the action
$(L_{after}, C_{after}) \leftarrow$ P99 latency and cost after the settling window
$R_{latency} \leftarrow w_L(L_{before} - L_{after})/L_{baseline}$ ▷ latency improvement
$R_{cost} \leftarrow -w_C(C_{after} - C_{before})/C_{budget}$ ▷ resource-increase penalty
$R_{SLA} \leftarrow 1 - (L_{after}/T_{SLA})^2$ if $L_{after} > T_{SLA}$, else 0 ▷ quadratic SLA penalty, Eq. (5)
$\sigma \leftarrow \max(0,\ L_{p99}/T_{SLA} - 1)$ ▷ violation severity
$\mu \leftarrow \sum_i |\Delta n_i| + 0.5 \sum_i (|\Delta c_i|/\gamma_c + |\Delta m_i|/\gamma_m + |\Delta \rho_i|) + \frac{1}{2}|\text{stages_scaled}|$ ▷ action magnitude
$R_{proactive} \leftarrow \sigma \cdot \mu \cdot w_{proactive}$ ▷ proactive bonus, Eq. (7)
$\tilde{L} \leftarrow L_{after}/L_{max}$; $\tilde{C} \leftarrow C_{after}/C_{max}$ ▷ normalize to $[0, 1]$
if $(\tilde{L}, \tilde{C})$ is non-dominated in $\mathcal{P}$ **then**
    $R_{pareto} \leftarrow 1 + H(a)$ ▷ hypervolume contribution w.r.t. $(1, 1)$, Eq. (6)
else
    $R_{pareto} \leftarrow 0.8/(1 + d(a, \mathcal{P}))$ ▷ Euclidean distance to $\mathcal{P}$, Eq. (6)
$R \leftarrow \text{clip}(R_{latency} + R_{cost} + R_{SLA} + R_{proactive} + R_{pareto},\ -R_{max},\ R_{max})$ ▷ Eq. (4)
return R

F Hyperparameters

G Raw cost data

Table 5 presents the raw cost per 1000 requests for all experiments.

H Bottleneck detection metrics

Table 6 presents the per-class precision, recall, and F1-score for bottleneck detection.

I Prompt template

The SAIR agent uses a structured prompt with three components, implemented via DSPy [Khattab et al., 2024] signatures. We reproduce the complete template below for reproducibility.

Input fields.

- `sampled_episodes`: Past episodes formatted as learning history with reward labels. Each episode shows: context → action → reward.
- `current_input`: Current pipeline metrics formatted as stage-level statistics (replicas, queue depth, CPU/GPU utilization, allocated resources, end-to-end latency).
- `constraints`: Physical infrastructure limits and scaling guidelines (see below).

Output schema. The LLM returns a structured JSON action:

Table 4: Hyperparameter settings for all experiments.

Parameter	Value
Context window size M	15 episodes
Reward threshold r_{min}	0
Latency reward weight w_L	0.7
Cost reward weight w_C	0.3
Proactive bonus weight $w_{proactive}$	0.3
Pareto reward non-dominated anchor α	1.0
Pareto reward dominated anchor β	0.8
Diversity regularization λ_{div}	0.1
Gaussian kernel bandwidth σ (z-score space)	1.0
Initial exploration rate ϵ_0	0.15
Exploration decay λ	0.95
Minimum exploration ϵ_{min}	0.05
Decision interval Δt	30 seconds
Settling window	5–10 seconds
LLM backend (main results)	GPT-5.1-mini
LLM temperature	1.0

Table 5: Raw cost per 1K requests ($, AWS on-demand: p3.2xlarge GPU, m5.xlarge CPU).

App	Pattern	Static	HPA	VPA	Thresh.	SAIR
ImgCls	Poisson	.058	.065	.070	.034	**.024**
	Ramp	.168	.185	.202	.081	**.072**
	Burst	.195	.267	.223	.127	**.076**
NLP	Poisson	.011	.008	.028	.014	**.007**
	Ramp	.038	.034	.047	.010	**.009**
	Burst	.028	.026	.012	**.009**	**.009**
TextGen	Poisson	.018	.017	.014	.014	**.006**
	Ramp	.012	.047	.188	.263	**.007**
	Burst	.012	.053	.013	.020	**.007**
Video	Poisson	.928	1.764	.704	.916	**.530**
	Ramp	1.764	2.520	.657	.691	**.490**
	Burst	1.411	1.411	1.642	1.010	**.557**

```
{"preprocessing": {
   "action": "scale_replicas"
     |"scale_resources"|"scale_both"
     |"none",
   "replicas": <int>,
   "cpu_millicores": <int>,
   "memory_mb": <int>},
 "inference": {
   "action": "adjust_rate"|"none",
   "rate_ratio": <float in [0,1]>},
 "postprocessing": {
   // same as preprocessing}}
```

Absolute-to-delta mapping. The LLM outputs *absolute* target values (e.g., `replicas: 3`, `cpu_millicores: 1000`). The executor converts these to deltas relative to the current state, then the action validator (Table 3) clamps each delta to the per-step discretization grid $\mathcal{A}_i$ before execution. This ensures Assumption 3.4 holds: regardless of the LLM's raw output, the executed action is always in $\mathcal{A}$.

Constraints prompt (excerpt).

> *Physical Constraints: Max GPUs: 2, Max CPU cores: 64, Max cost budget: $100/hour.*
> *Scaling Guidelines: Current resource allocations are shown in state. Aggressive reduction (>50%) is dangerous. If p99 > SLA and utilization is low, a burst already passed but the*

Table 6: Per-class precision and recall for bottleneck detection.

Class	Precision	Recall	F1-Score
Preprocessing	87.0%	88.7%	87.8%
Inference	83.9%	89.7%	86.7%
Postprocessing	89.8%	91.7%	90.7%
Multiple	82.9%	70.7%	76.3%
Overall accuracy		86.0%	

system failed to handle it; needs more capacity. Trust the latency: p99 > SLA means the system failed.

Goals: Primary: Keep p99 latency below SLA target. Secondary: Minimize cost. Learn from outcomes: if scaling a stage didn't improve latency, avoid scaling it next time.

Episode format. Each in-context episode is formatted as:

```
Episode (Reward: +0.85 GOOD):
Input:
  preprocessing: replicas=2, queue=15
    CPU_usage=78.3%, ...
  inference: replicas=1, queue=3
    GPU_usage=45.2%, rate_ratio=0.80
  E2E Latency: p99=342ms
Prediction:
  {"preprocessing": {"action":
   "scale_replicas", "replicas": 3},
   "inference": {"action": "none"}}
Reward: +0.85
```

Episodes are ordered by reward (curriculum ordering) so the LLM observes a learning progression from lower to higher rewards.

J Warm/cold start protocol and per-run data

We ran three interleaved pairs (C1, W1, C2, W2, C3, W3) on Text Generation at 100 rps steady Poisson, 300 s per run, 15 s decision intervals. The warm-started controller was pre-seeded with the 13 positive episodes from cold run 1. LLM backend: GPT-5.2 (zero API failures across $\sim$90 decisions). Convergence criterion (fixed before inspection): the first time the cold-started controller's rolling-window P99 enters a 5% band around the warm-started controller's steady-state mean and stays for two consecutive windows.

Table 7: Warm vs. cold start per-run results.

Run	Full-run P99 (ms)	Convergence time (s)	Paired diff (ms)
Cold 1	146.1	46.6	+20.6
Warm 1	125.5	–	
Cold 2	112.1	54.4	−21.2
Warm 2	133.3	–	
Cold 3	151.1	58.9	+11.7
Warm 3	139.4	–	
Mean cold	136.4	53.3	+3.7
Mean warm	132.7	–	

The paired differences vary in sign across runs. The early-60s window (first 4 decisions) shows cold P99 of 115.0 ms (mean) vs. warm 122.1 ms; the cold start incurs zero penalty under this load because the initial provisioning is adequate and the zero-shot prior acts sensibly from decision 1.

K FIRM/AWARE training and frozen evaluation

We implemented FIRM (DDPG) and AWARE (PPO) as adapter modules with native state dimensions (FIRM: 10, AWARE: 13), native action spaces (FIRM: 15 actions including 6 no-op I/O and network actions with -0.25 penalty; AWARE: 7 actions), and native reward functions. Both share the same environment wrapper and the evaluation harness computes all external KPIs (latency, SLA compliance, cost) uniformly using the same cost constants.

Training was conducted in an M/M/1 queueing-model simulator (analytic per-stage latency, randomized per-episode load and initial provisioning). Both algorithms learned and each roughly halved SLA violations vs. an untrained network ($\sim$40%$\rightarrow\sim$21% on Text Generation). Both over-provision at idle, consistent with their reward functions (both reward functions omit a cost term).

Frozen evaluation used the live testbed: we loaded checkpoints via the harness eval path, 120 s $\times$ 3 reps $\times$ {Poisson, Burst}, with the harness's per-app load profile.

Table 8: Frozen RL baseline evaluation (Text Generation, live testbed).

Policy	Workload	Avg P99 (ms)	Max P99 (ms)	Behavior
FIRM-DDPG	Poisson	61.7 / 62.2 / 63.3	66	never scales
FIRM-DDPG	Burst	163.1 / 163.9 / 166.2	187	never scales
AWARE-PPO	Poisson	60.9 / 61.3 / 66.0	88	never scales
AWARE-PPO	Burst	166.5 / 236.3 / 280.0	556	sporadic

FIRM's DDPG policy converged to a no-op in all conditions. AWARE's PPO policy attempted sporadic scaling under burst (preprocessing replicas up to 2.6 avg) but produced burst-tail P99 up to 2.8$\times$ the 200 ms SLA. The 10 s decision interval (baselines) vs. 15 s (SAIR) favors the baselines. Training seed: 42.

L Co-location measurements

Co-location experiments measure whether the GPU capacity that effective-cost accounting treats as freed is physically consumable. The setup uses a Text Generation (GPT-2) inference container with `LD_PRELOAD` token-bucket interception on a dedicated GPU. The primary runs at 57 rps (30% of 190 rps maximum); the co-tenant is a second unthrottled instance on the same GPU at closed-loop maximum drive.

Table 9: Standard condition: co-tenant share of standalone throughput.

ρ	Co-tenant (rps)	Share of standalone	Primary P99 (ms)
1.0	188.0	98.9%	67.7
0.7	187.9	98.8%	68.9
0.5	185.4	97.5%	69.5
0.3	185.5	97.5%	68.6
0.1	185.7	97.7%	69.1

Standalone primary P99 was 67.3 ms (ρ=0.5) and 67.9 ms (ρ=0.1), indistinguishable from shared-GPU values.

For the calibrated saturation condition, we walked the token cap down until throughput fell below $0.5\times$ baseline, yielding a knee at 3×10^{-4}; the calibrated cap is $\rho^* = 6\times10^{-4}$.

Primary throughput tracks ρ near-proportionally (0.14–1.00 measured vs. 0.1–1.0 nominal). The co-tenant is insulated (180–184 rps vs. 181.8 standalone). On a second machine, co-tenant shares are 98.9–99.6% (GPU 0) and 99.0–99.1% (GPU 1), overhead 0.07%, with the same proportional shares under calibrated contention.

Table 10: Calibrated saturation: proportional GPU partitioning under contention.

ρ_{rel}	Primary (rps)	Share of ρ_{rel}=1.0	Co-tenant (rps)	Primary P99 (ms)
1.0	99.1	1.00	180.1	496
0.7	76.9	0.78	181.7	616
0.5	58.4	0.59	181.5	743
0.3	38.1	0.38	182.1	1211
0.1	14.0	0.14	184.1	2378

M System scale experiments

4-GPU deployment. The second node (cloudsys02) joined the k3s cluster as an agent. The Text Generation inference stage scaled across both machines behind one Service (kube-proxy distributes traffic cross-node). Fixed offered load: 240 rps Poisson, 180 s per phase, preprocessing 6 replicas, postprocessing 4 replicas pinned.

Table 11: Inference scaling curve at a fixed 240 rps offered load; one replica saturates.

Inf. replicas (placement)	Served (rps)	P50 (ms)	P99 (ms)
1 (node 1)	237.9	13,379	27,006
2 (1+1 cross-node)	240.1	72.0	120.2
4 (2+2 cross-node)	240.8	69.9	107.3

SAIR controlled the 4-GPU deployment (max inference replicas set to 4, 2+2 cross-node) in 3 runs: served 247.5/234.0/238.4 rps vs. 240 offered at $\geq$99.99% success, P99 589/476/571 ms. Each 300 s run completed 3–5 decisions because metric scrapes from the newly joined node timed out and delayed each observation; this lowers the decision rate and leaves the control logic unchanged from the single-node runs.

Control-plane concurrency. We ran N concurrent real controller instances (live metrics, real GPT-5.2 decisions, real validator; observers record decisions without actuating) at 15 s cadence, 240 s per phase, on NLP Analysis and Text Generation.

Table 12: Control-plane scale-out.

N	Decisions	Failures	Mean latency (s)	P95 (s)
2	32	0	1.03	4.13
8	128	0	0.59	0.69
16	256	0	0.64	0.71

Per-decision latency is flat in N; the N=2 P95 outlier (4.13 s) is a first-call LLM warm-up. We observed zero failures across 416 decisions; cost scales linearly ($N\times$ \$0.01–0.05 per pipeline-hour).

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The contributions stated in the introduction (autoscaler design, regret bound with constant Pareto separation margin, Bayesian-surprise experience retrieval, continuous GPU rate control) each map directly to a section of the paper (Sections 4–5) and to an empirical result in Section 6. Quantitative claims (best/tied-best P99 and effective cost, up to 50%/97% improvements, 86% bottleneck accuracy) are accompanied in the abstract and Section 7 with explicit qualifications about the GPU sharing assumptions under which the effective-cost numbers hold.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: The Scope paragraph of Section 3 lists the settings outside SAIR's scope: per-request scheduling, trace-rich deployments where offline-trained controllers can outperform a zero-shot controller, and sub-second control loops given the 1–2 s LLM decision latency; the same section states when the bandit abstraction becomes invalid. Section 5 states when the regret bound grows linearly in T and maps each term to a measured quantity. Section 7 covers cold-start behavior, three measured failure modes, the GPU-sharing condition behind the effective-cost savings, and the settings left to future work (cross-cluster scheduling, $\Delta t < 10$ s, and prefill/decode serving).

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.

- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

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

   Answer: [Yes]

   Justification: Assumptions 3.1–3.4 are stated explicitly in Section 3 with the conditions under which each holds. Each subsequent theoretical claim (Theorem 5.1, Proposition 5.2, Theorem A.1, Lemmas A.2–5.3, Corollary 5.4, Lemma B.1) cites the assumptions it depends on; main-text proof sketches are accompanied by full proofs in Appendix A and Appendix B.

   Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: The main paper specifies the agent algorithm (Algorithm 1), the reward computation (Algorithm 2, Section 4), and the experimental protocol (Section 6). Appendices document the action validator constraints (Table 3), the LLM prompt template and episode formatting (Appendix I), all hyperparameters including kernel bandwidth and LLM backend (Appendix F), and the system implementation surface (Kubernetes operator, metrics collection, action execution; Appendix C). Together these cover the components needed to reproduce the four serving pipelines and the autoscaling control loop.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.

- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

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

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

   Answer: [No]

   Justification: The code is not yet public. We will release the SAIR Kubernetes operator, the CUDA-interception rate-control library, prompt templates, and the workload generators under an open-source license. The paper itself fully specifies the algorithm (Algorithm 1), prompts (Appendix I), validator (Table 3), and hyperparameters (Appendix F) needed to re-implement SAIR independently.

   Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: Section 6 specifies hardware, the four pipeline workloads, the three workload patterns, and the per-baseline tuning protocol (HPA targets and Threshold parameters were swept and best-performing values selected). Per-decision token usage and LLM cost are reported in the Setup. SAIR has no gradient updates, so "training" reduces to the in-context

experience buffer described in Algorithm 1; all hyperparameters are listed in Appendix F and rationale for the reward weights is given in Section 4.

Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [No]

   Justification: Each experiment processes about 1M requests and is repeated with 3 independent seeds; the main figures show seed-averaged means. We report inter-seed variation explicitly in Section 6 (P99 latency varied by at most 5% on Image Classification, 8% on NLP and Text Generation, and 12% on Video Analysis), and we state that the cross-baseline gaps in the main results exceed this inter-seed variation. We do not overlay error bars on figure panels because they would compress the visual ranges; we will include 1-sigma intervals in a revision if requested.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: Section 6 specifies the cluster (2×NVIDIA RTX A6000, 64 CPU cores, 256 GB RAM, Kubernetes), the LLM backend used for the controller (GPT-5.1-mini), and the per-decision token budget (4–6K input, $\leq$300 output) and call rate ($\sim$120 calls/hour/pipeline). One full pipeline run ($\sim$1M requests) takes hours not days on this cluster; total compute across $4 \times 3 \times 3$ seed runs plus baselines fits within a single shared 2-GPU testbed.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The work involves only synthetic workload generation against open-source models on the authors' own hardware; there are no human subjects, no scraped or personally identifying data, and no deployed user-facing system.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: The primary intended impact is positive: more efficient autoscaling reduces the energy and resource cost of ML inference serving, which is a growing fraction of datacenter consumption. The principal indirect risk is that LLM-based control loops add a new dependency on a third-party model API for production-critical scheduling decisions; we mitigate this risk by making the system model-agnostic (Section 6 cross-checks the GPT-5.1-mini backend against Claude Haiku and Qwen-32B) and by enforcing safety bounds via the action validator (Table 3). The work is infrastructure-level and is not tied to a specific application or user-facing decision system.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.

- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The paper does not release pre-trained generative models, scraped datasets, or any other artifact with a meaningful misuse surface. The planned release is a Kubernetes operator and a CUDA rate-limiting library, neither of which poses a misuse risk distinct from existing open-source schedulers.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: All baselines, model architectures, and software components are cited at first use in Section 2 and Section 6. We use Kubernetes (Apache-2.0), DSPy (MIT), and standard PyTorch model implementations; the LLM backend (GPT-5.1-mini) is queried via its public API under the provider's terms of service. No proprietary or restricted-license dataset is used.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

Answer: [N/A]

Justification: The SAIR operator and CUDA rate-control library are planned for public release; the release will include README, prompt templates, hyperparameter files (Appendix F), and example PipelineAutoscaler resources documenting the action schema (Appendix I).

Guidelines:

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

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

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

Answer: [N/A]

Justification: The paper involves no crowdsourced annotations, no user studies, and no human subjects of any kind.

Guidelines:

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

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

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

Answer: [N/A]

Justification: No human subjects are involved in this work, so IRB approval is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

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

Answer: [Yes]

Justification: An LLM is the central component of the proposed method: it serves as the in-context reinforcement learning controller that selects scaling actions at each decision interval. Section 4 describes how the policy is defined as the LLM's conditional distribution over actions given retrieved experiences; Section 6 reports the specific backend (GPT-5.1-mini), the prompt structure, and the per-decision token usage. Appendix I provides the full prompt template, output schema, and absolute-to-delta mapping used in deployment.

Guidelines:

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