INTERNATIONAL JOURNAL OF INFORMATION AND COMMUNICATION TECHNOLOGY TRENDS

Vol. 6 No. 02 (2026)

e.ISSN: 2409-5222

https://ijictt.ictt.info

CRAI: Calibrated Regret-Aware LLM Invocation for Efficient Cloud Resource Scheduling

Aiman Rahman and Nur Aisyah Karim

Faculty of Computer Science and Information Technology, Universiti Malaya, Kuala Lumpur, Malaysia

Corresponding author. aisyakarm@um.edu.my

Abstract

Large language models can contribute semantic and strategic reasoning to cloud schedulers, but invoking a large model at every scheduling epoch converts inference latency and monetary cost into a control-plane bottleneck. This paper studies the invocation decision itself. We present CRAI, a calibrated regret-aware invocation controller that treats an LLM as an expensive strategic meta-scheduler and calls it only when the estimated loss of reusing the cached policy is likely to exceed a budget-dependent dual price. CRAI combines a compact cluster-state representation, an online ridge predictor with an uncertainty bonus, and dual ascent that tracks a target invocation budget. The LLM-selected policy is never executed directly: a deterministic scheduler enforces CPU and memory feasibility while realizing one of four policy profiles—BALANCE, PACK, EDF, or LOCALITY. We evaluate the controller in a 48-node discrete-event simulator under normal, high-load, and bursty workloads using five independent seeds. Planner outputs are frozen across all runs to isolate invocation timing from provider-side nondeterminism. CRAI invokes the strategic planner for 9.54% of active epochs, a 90.46% reduction relative to always invoking, while average job completion time remains within 1.35% of the always-invoke reference. Against a static policy, CRAI reduces average JCT by 1.46% and SLA misses by 1.08 percentage points; against a state-change trigger, it reduces calls by 79.8% and JCT by 0.30 s. These results show that cloud schedulers can capture most of the value of strategic reasoning without paying its inference cost on routine epochs.

Keywords: *cloud resource scheduling, large language models, adaptive invocation, online learning, budgeted control, service-level agreements.*

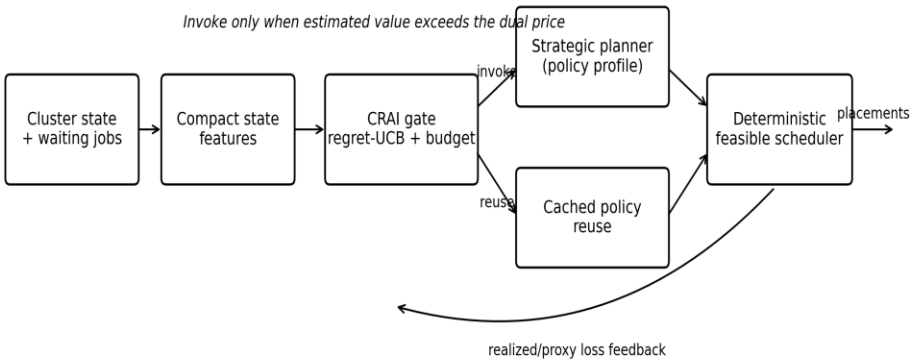


Figure 1 CRAI separates strategic reasoning from deterministic placement. The gate invokes the expensive planner only when estimated stale-policy regret exceeds a learned, budget-aware threshold.

1. Introduction

Cloud schedulers operate at the intersection of heterogeneous resource demands, workload bursts, locality constraints, priorities, and service-level objectives. Production systems such as Borg, Omega, Mesos, and YARN demonstrate that the control plane must make decisions quickly while preserving predictable feasibility and isolation [2]–[6]. Classical designs therefore rely on carefully engineered heuristics, fairness rules, packing objectives, or incremental optimization [7]–[13]. These mechanisms are efficient, but the policy that is appropriate under low contention may be inappropriate when deadline risk, resource fragmentation, or data-locality pressure changes abruptly.Recent work has explored learning-based schedulers and large language models (LLMs) as a source of higher-level reasoning. Deep reinforcement learning can learn workload-specific scheduling behavior [17], [18], while modern LLMs can perform few-shot reasoning over heterogeneous textual context [23]–[26]. Ding et al. introduced LLMSched, an important hybrid architecture that combines LLM reasoning with optimization-based refinement and explicitly amortizes inference through workload-aware triggering and strategy caching [16]. Their design demonstrates that strategic reasoning need not sit on the per-task critical path and provides a strong foundation for efficient LLM-assisted scheduling.The remaining question is when a fresh reasoning call is actually worth paying for. A change detector answers whether the system state moved; it does not answer whether the movement is decision-relevant. Two cluster states can have a large Euclidean difference while the best policy remains unchanged, whereas a small change near an SLA boundary can make a policy refresh valuable. Periodic refresh has the inverse problem: it spends calls even during long stationary intervals. This motivates an invocation controller that estimates value-of-information rather than raw state novelty.We formulate LLM invocation as a budgeted online decision. At each active scheduling epoch, the controller either reuses a cached strategy or queries the strategic planner. CRAI predicts the one-step regret of retaining the cached policy, adds an uncertainty bonus to avoid systematically missing rare regime changes, and compares the resulting upper-confidence estimate with a dual price that rises when the invocation budget is being overspent. This creates an interpretable control rule: invoke when the estimated operational value of a fresh strategy is greater than its current scarcity price.The contributions are fourfold. First, we introduce a regret-aware LLM invocation objective for cloud scheduling, separating the cost of reasoning from the quality of downstream placement. Second, we design CRAI, an online controller that combines uncertainty-aware regret prediction and dual budget tracking without model retraining. Third, we retain hard feasibility through deterministic placement, so a language model can only select a high-level scheduling profile rather than emit unconstrained node assignments. Fourth, we provide a controlled multi-scenario evaluation with paired statistical tests and budget sensitivity analysis. The measured results show

that approximately one call in ten is sufficient to retain most of the scheduling benefit of always refreshing the strategic policy.

II. RELATED WORK

A. Cluster and Cloud Scheduling

Large-scale cluster management has evolved through several architectural families. Borg uses centralized admission control, priority, preemption, and packing at datacenter scale [2], while Omega explored shared-state parallel schedulers with optimistic concurrency [3]. Mesos and YARN separate resource management from application-level scheduling [4], [5], and the Borg–Omega–Kubernetes lineage highlights how declarative orchestration and extensible policy became practical container-management mechanisms [6]. At the policy level, DRF formalizes multi-resource fairness [7], Tetris reduces multidimensional fragmentation [8], Firmament applies min-cost flow for high-quality placement [9], and Sparrow targets very low scheduling latency through distributed randomized sampling [10]. Quasar, Resource Central, and altruistic scheduling further show the value of performance awareness, workload prediction, and long-horizon objectives [11]–[13].Public cloud traces emphasize why one fixed policy is rarely sufficient. Google trace analysis documents strong machine and workload heterogeneity as well as substantial temporal dynamics [14], and subsequent Borg work shows continued evolution of production cluster-management architecture [15]. GPU-cluster studies similarly report diverse job durations, resource shapes, and placement constraints [20]–[22]. CRAI does not replace these schedulers; it controls when an expensive strategic layer should refresh the policy used by a fast feasibility-preserving executor.

B. Learning-Based Scheduling

DeepRM showed that resource management can be expressed as a reinforcement-learning problem [17], and Decima introduced graph-aware reinforcement learning for data-processing DAGs [18]. Online-feedback systems such as Cilantro learn performance-resource relationships during operation and adapt allocations using uncertainty-aware estimates [19]. These methods establish the value of learned policies, but they typically assume a trained model is inexpensive enough to evaluate whenever needed. CRAI addresses a different cost structure in which the strategic model itself is expensive to query and its invocation becomes a first-class scheduling decision.

C. LLM Reasoning and Selective Inference

Few-shot prompting [23], chain-of-thought reasoning [24], and large instruction-following models [25], [26] motivate the use of LLMs for high-level decision support. At the same time, planning studies show that executable correctness is better supported by external verifiers or planners than by unconstrained language generation alone [27], [28]. This supports the architecture adopted here: the LLM selects a strategic profile and deterministic logic enforces feasibility. Ding et al. [16] make the same separation especially compelling in cloud scheduling by combining LLM guidance with optimization and by using caching to reduce routine inference.Selective LLM use is also related to cost-aware model cascades and routing. FrugalGPT adaptively combines language-model APIs to trade cost against task quality [29], while RouteLLM learns query-dependent routing between stronger and weaker models [30]. CRAI differs in two respects. Its alternative to invoking is not another model but a cached control policy, and the loss signal is operational scheduling regret rather than response preference. Methodologically, its uncertainty term follows ideas from linear stochastic bandits [31], while its budget price follows the long-term-constraint perspective of online optimization [32].

III. PROBLEM FORMULATION

A. Cluster, Jobs, and Feasible Placement

Consider a cluster of N heterogeneous nodes. Node i has CPU and memory capacity C(i,r), and residual capacity R(i,r,t) for resource r at scheduling epoch t. A waiting task j has demand d(j,r), estimated duration e(j), priority p(j), optional deadline D(j), and preferred data zone z(j). Binary decision x(j,i)=1 denotes assignment of task j to node i. Before every assignment, the executor enforces the capacity constraint:

$$\Sigma(j: x(j,i)=1)\ d(j,r) \le R(i,r,t), \quad r \in \{CPU, MEM\}. \qquad (1)$$

The strategic layer chooses only a policy profile. BALANCE minimizes post-placement peak load and CPU–memory imbalance; PACK prefers tight multidimensional best fit; EDF orders tasks by deadline and then performs balanced placement; LOCALITY prioritizes same-zone placement and uses load as a secondary criterion. Because all four profiles call the same feasibility checker, strategic mistakes cannot over-allocate CPU or memory.

B. Invocation as a Budgeted Decision

Let $s(t)$ be the compact state at an active scheduling epoch, $\pi c(t)$ the currently cached policy, and $\pi f(t)$ the policy returned by a fresh strategic-planner query. Let $L(s,\pi)$ denote an operational proxy loss obtained by greedily replaying the current waiting queue under policy π. The instantaneous value of refreshing the policy is

$$\Delta(t) = \max\{0, L(s(t), \pi c(t)) - L(s(t), \pi f(t))\}. \quad (2)$$

Let $z(t)\in\{0,1\}$ denote whether the planner is invoked. An always-invoke scheduler sets $z(t)=1$ for every active epoch; a budgeted scheduler targets an average call fraction B. The control objective can be written as a quality-cost objective with a long-run invocation constraint:

$$\min \Sigma(t)\, L(t) + c_{LLM}\, \Sigma(t)\, z(t), \quad \text{subject to } (1/T)\Sigma(t)\, z(t) \le B. \quad (3)$$

The key challenge is that $\Delta(t)$ is observed only when a refresh is performed because only then is the fresh strategic policy available. CRAI therefore learns an optimistic estimate of $\Delta(t)$ online and prices calls using a dual variable.

IV. CRAI DESIGN

A. Compact State Representation

CRAI intentionally avoids serializing the full cluster into a prompt at every epoch. The fast path computes six normalized features: mean resource load, cross-node load imbalance, queue pressure, deadline risk, CPU–memory fragmentation, and locality pressure. A seventh signal measures feature drift from the preceding active epoch, and an eighth records policy age. Together with a bias term, the predictor uses a nine-dimensional vector $x(t)$. This representation is small enough for constant-cost gating while retaining the signals that distinguish the four strategic profiles.Deadline risk is the mean urgency of deadline-sensitive waiting tasks, increasing as residual slack approaches predicted execution time. Fragmentation is the average mismatch between residual CPU and memory fractions. Locality pressure combines task locality weights with the load of each task's preferred zone. Drift is the root-mean-square change of the six primary state features. These quantities are inexpensive to compute from scheduler telemetry and require no language-model call.

B. Regret Predictor with Uncertainty Bonus

CRAI maintains a ridge-regression model $A(t)$ and $b(t)$ over observed refresh opportunities. With $\theta(t)=A(t)^{-1}b(t)$, the predicted refresh value is $\theta(t)^T x(t)$. To remain responsive while the model has little evidence in a region of the state space, CRAI adds an upper-confidence bonus based on $x(t)^T A(t)^{-1} x(t)$ [31]:

$$U(t) = \theta(t)^T x(t) + \beta \sqrt{[x(t)^T A(t)^{-1} x(t)]}. \quad (4)$$

When CRAI invokes the planner, it computes the proxy loss of both the old cached policy and the fresh policy on the same current queue. Their non-negative difference becomes the regression target $y(t)=\Delta(t)$, and the model is updated with $A \leftarrow A + x(t)x(t)^T$ and $b \leftarrow b + x(t)y(t)$. The update uses only nine features, so the numerical cost is negligible relative to queue scanning and placement.

C. Dual Control of Invocation Budget

A fixed threshold makes the call rate sensitive to workload scale. CRAI instead maintains a dual price $\lambda(t)$ for the long-run budget. If the cumulative invocation ratio is above target B, λ rises; if the system has conserved calls, λ falls:

$$\lambda(t+1) = \text{clip}(\lambda(t) + \eta[\rho(t) - B], \lambda_{min}, \lambda_{max}). \quad (5)$$

where $\rho(t)$ is the running fraction of active epochs that have invoked the planner. CRAI refreshes when $U(t)>\lambda(t)$. A maximum policy age provides a liveness safeguard so a cached policy is not retained indefinitely even if the predictor is conservative. This primal–dual structure follows the general idea of

satisfying long-term resource constraints with online prices [32], but the constrained resource here is LLM invocation budget.

D. Strategic Planner Interface and Fast Execution

The planner consumes a coarse regime representation and returns a policy token plus a short rationale. In a live system, this interface can be backed by an LLM such as those described in [25], [26]. The downstream scheduler never interprets free-form node assignments. Instead, it validates the policy token and executes the associated deterministic ordering and placement rule. This yields a narrow semantic control surface: the expensive reasoning layer chooses what to prioritize, while the fast executor decides exactly where tasks run.The implementation uses 16 regime archetypes formed by four binary stress indicators: load/queue pressure, deadline risk, fragmentation, and locality pressure. The strategic outputs map these regimes to BALANCE, PACK, EDF, or LOCALITY. For the controlled experiments in Section V, the planner interface is frozen to the same 16 responses for every controller and random seed. This design isolates the invocation controller: measured differences cannot be attributed to provider latency variation, response sampling, or changes in planner content.

E. Complexity

Let Q be the number of waiting tasks and N the number of nodes. Computing state features is O(N+Q), and deterministic placement is O(QN) in the straightforward implementation. CRAI adds a constant-size 9×9 linear-algebra update only on invoked epochs. The gate therefore does not change the asymptotic placement cost and is independent of the size of the LLM.

F. Calibration, Cold Start, and Invariants

The controller starts with $A(0)=0.8I$, $b(0)=0$, $\lambda(0)=0.10$, uncertainty coefficient $\beta=0.18$, dual step $\eta=0.16$, and λ clipped to $[0.025,0.75]$. Every non-static controller performs a strategic refresh at its first active epoch. The uncertainty term then keeps unexplored regions eligible for refresh, while the age cap prevents a stale policy from persisting solely because early observations were uninformative. These mechanisms avoid a cold-start deadlock without requiring an offline training phase.Three invariants simplify deployment. First, planner outputs are restricted to the four validated policy tokens; any missing initial token defaults to BALANCE. Second, task placement always passes CPU and memory feasibility checks before resources are committed. Third, the learned signal influences only whether to refresh the strategic profile, not the correctness of an individual placement. Consequently, the adaptive gate can fail conservatively toward an older feasible strategy without creating resource over-allocation.

Algorithm 1. CRAI at an active scheduling epoch

1) Observe compact state $x(t)$ and cached policy πc.
2) Compute the optimistic refresh value $U(t)$ from (4).
3) If $U(t)>\lambda(t)$, or policy age reaches 28 active epochs, invoke the planner.
4) On invocation, obtain πf, measure counterfactual proxy regret, and update A and b.
5) Otherwise reuse πc; then update $\lambda(t)$ from the running call ratio.
6) Execute the selected profile through deterministic feasibility checks.

V. EXPERIMENTAL METHODOLOGY

A. Simulator and Workloads

We implemented a discrete-event simulator with 48 heterogeneous nodes across three zones. Node types expose 16–32 nominal CPU units and 64–128 memory units, with a small deterministic capacity perturbation per seed. Jobs contain one to four independent tasks. Task durations are lognormally distributed and clipped to 4–300 s; resource shapes mix CPU-heavy, memory-heavy, and balanced modes. Sixty-four percent of jobs receive deadlines, and priority takes one of 12 integer levels. The generator is trace-calibrated rather than a claim of direct trace replay: its qualitative choices—heterogeneous machine capacities, dynamic arrival intensity, resource-shape diversity, and heavy-tailed execution time—follow characteristics documented in Google and production ML-cluster studies [14], [22].

We evaluate three arrival regimes. Normal load uses 1.30 jobs/s. High load uses 2.30 jobs/s. Bursty load alternates a quiet rate of 0.72×1.10 jobs/s with a burst rate of 2.2×1.10 jobs/s during the 330–510 s interval of each 540 s cycle. Main runs last 1,200 s with 5 s scheduling epochs and use seeds 11, 23, 37, 53, and 71. Every compared controller receives the exact same workload for a given scenario and seed.

Table 1 Main experimental configuration.

Item	Setting
Cluster	48 heterogeneous nodes / 3 zones
Epoch	5 s
Main horizon	1,200 s
Seeds	11, 23, 37, 53, 71
Scenarios	normal / high / bursty
Deadline jobs	64%
Profiles	BALANCE / PACK / EDF / LOCALITY
CRAI target B	0.095

B. Controllers

Static-Balance never invokes and always uses BALANCE. Periodic-10 refreshes the strategic policy every ten active epochs. State-Delta invokes when state drift exceeds 0.18, deadline risk exceeds 0.30, or policy age reaches 25 epochs; it represents a direct event/change-triggered baseline. Always-LLM refreshes on every active epoch and serves as the maximum-information reference. CRAI uses the regret-UCB gate with target budget $B=0.095$, uncertainty coefficient $\beta=0.18$, dual step $\eta=0.16$, and maximum policy age 28 epochs.

C. Metrics and Statistical Procedure

We report mean job completion time (JCT), 95th-percentile JCT, deadline miss rate among deadline-sensitive jobs, CPU and memory utilization, planner invocation fraction among active scheduling epochs, and locality hit rate. All primary comparisons use 15 paired observations (three scenarios × five seeds). Paired two-sided t-tests compare CRAI with each baseline on JCT, deadline misses, and invocation rate; 95% confidence intervals are reported for paired differences. Budget sensitivity is evaluated at $B\in\{0.03,0.05,0.08,0.12,0.20\}$ using two seeds per scenario and a 900 s horizon.

D. Counterfactual Proxy Loss

The gating label is deliberately distinct from the final evaluation metrics. On an invoked CRAI epoch, the simulator makes capacity-only copies of the current nodes and greedily replays the waiting queue under both the cached and fresh policy. The proxy loss is a weighted sum of five immediate signals: fraction of urgent tasks left unplaced (weight 3.0), fraction of all queued tasks left unplaced (1.45), residual CPU–memory fragmentation (0.85), cross-node load imbalance (0.55), and weighted locality misses (0.22). The difference between cached and fresh proxy losses is the non-negative training target in (2).This counterfactual construction provides labels only at refresh opportunities and does not alter the realized workload. Final JCT, deadline misses, utilization, and locality are computed from the actual discrete-event execution. Thus, a favorable proxy score can influence future invocation choices, but all reported performance numbers come from completed jobs and realized placements rather than from the proxy itself.

VI. RESULTS

A. Overall Quality–Invocation Tradeoff

Table II summarizes the 15-run mean for each controller. Always-LLM obtains the lowest JCT, 36.456 s, but invokes on every active epoch. CRAI reaches 36.947 s while invoking on 9.544% of active epochs. The JCT gap is therefore 0.491 s, or 1.35%, while the call reduction is 90.46%. Deadline misses differ by 0.183 percentage points (6.116% versus 5.933%). The result is the central systems tradeoff: nearly all of the quality of constant strategic refresh is retained with approximately one-tenth of the calls.

Table 2 Overall performance averaged over 15 paired runs.

Controller	JCT (s)	SLA (%)	CPU (%)	Inv. (%)
Static	37.493	7.200	47.947	0.00
Periodic-10	36.946	6.117	48.309	10.11
State-Delta	37.245	5.863	48.241	47.17
CRAI	36.947	6.116	48.211	9.54
Always	36.456	5.933	48.313	100.00

Relative to Static-Balance, CRAI reduces mean JCT by 0.546 s (1.46%) and the SLA miss rate by 1.085 percentage points. The paired improvements are statistically supported: JCT difference −0.546 s, 95% CI [−0.924, −0.168], p=0.0078; SLA difference −1.085 points, 95% CI [−1.663, −0.506], p=0.0013. The effect arises without changing the feasibility mechanism; it is entirely due to selective strategic-policy refresh.CRAI and Periodic-10 have effectively identical aggregate JCT and SLA results. The paired JCT difference is 0.001 s (p=0.993), and the SLA difference is −0.002 points (p=0.983). CRAI nevertheless uses 0.562 percentage points fewer calls on average. This call-rate difference is directionally favorable but its paired p-value is 0.061; the more important distinction appears in the scenario breakdown, where CRAI shifts calls away from routine epochs and concentrates them under contention.Average occupancy is similar among the adaptive controllers: CRAI records 48.211% CPU and 43.562% memory utilization, compared with 48.313% and 43.664% for Always-LLM. Static-Balance is slightly lower at 47.947% CPU and 43.371% memory. The narrow utilization spread indicates that the main measured gains arise from which jobs are prioritized and where tasks are placed, rather than from simply driving the cluster to a materially higher mean occupancy.

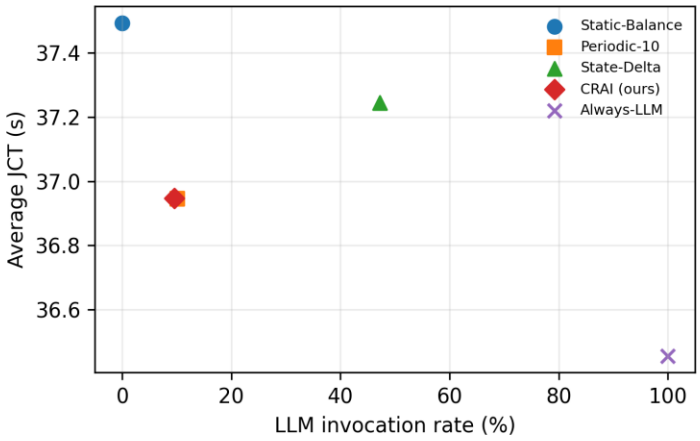


Figure 2 Mean JCT versus strategic-planner invocation rate. CRAI occupies the low-call region while remaining close to the always-invoke reference.

B. Comparison with State-Change Triggering

State-Delta invokes on 47.17% of active epochs, nearly five times the CRAI rate. CRAI reduces the invocation rate by 37.63 percentage points, equivalent to 79.8% fewer calls, and also lowers JCT by 0.298 s. The paired JCT improvement has 95% CI [−0.564, −0.033] with p=0.030. State-Delta achieves a 0.253-point lower SLA miss rate (p=0.013), revealing a concrete operating tradeoff: aggressive state-change triggering buys a modest deadline advantage at substantially higher inference frequency. CRAI instead optimizes the value of refresh under a call budget.

C. Workload-Regime Breakdown

The scenario breakdown explains where selective invocation matters. Under normal load, every controller produces essentially the same JCT (35.784–35.789 s) and identical mean SLA miss rate (5.764%). Always-

LLM nevertheless spends a call at every active epoch and State-Delta calls at 52.0%. This is the clearest example of state movement without decision value: routine changes do not alter the scheduling outcome enough to justify repeated strategic reasoning.Under high load, the control decision becomes consequential. Static-Balance reaches 39.205 s mean JCT and an 8.404% miss rate. CRAI improves these to 37.856 s and 6.243% while invoking on only 8.872% of active epochs. Periodic-10 obtains 38.059 s and 6.297% at 10.141% calls, whereas State-Delta uses 39.885% calls yet reaches 38.714 s JCT. Thus, under the strongest contention, CRAI achieves the best JCT among the sub-10%-call controllers and refreshes less often than the fixed-period policy.The bursty regime shows a similar but smaller effect. CRAI reaches 37.199 s JCT and 6.341% misses with a 9.810% invocation rate. Always-LLM improves JCT to 36.739 s but requires 100% calls. State-Delta has the lowest miss rate among adaptive low-level baselines at 5.945%, but invokes on 49.63% of active epochs. The results support a separation between "state is changing" and "the cached strategy is now costly."

Table 3 CRAI scenario means.

Scenario	JCT (s)	SLA (%)	Inv. (%)
Normal	35.786	5.764	9.95
High	37.856	6.243	8.87
Bursty	37.199	6.341	9.81

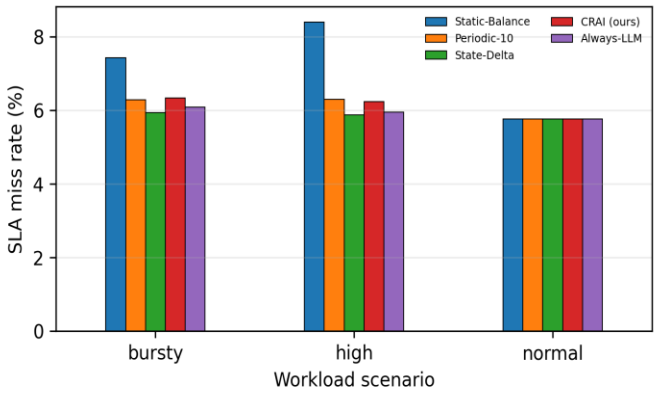


Figure 3 Deadline miss rate by workload regime. Under normal load the policies coincide; differences emerge under contention and bursts

D. Budget Sensitivity

Budget sensitivity confirms that CRAI can expose an explicit operations knob rather than hiding cost inside an ad hoc threshold. Across target budgets 0.03 to 0.20, observed call rate scales from approximately 4.0% to 19.5%. Mean JCT varies within a narrow 36.49–36.69 s band in the shorter sensitivity runs. The best observed mean is 36.495 s at the highest tested budget, while B=0.12 reaches 36.593 s with 12.69% calls. The non-monotone small differences are consistent with finite-run stochastic variation; the operational pattern is a broad quality plateau with progressively higher inference use.

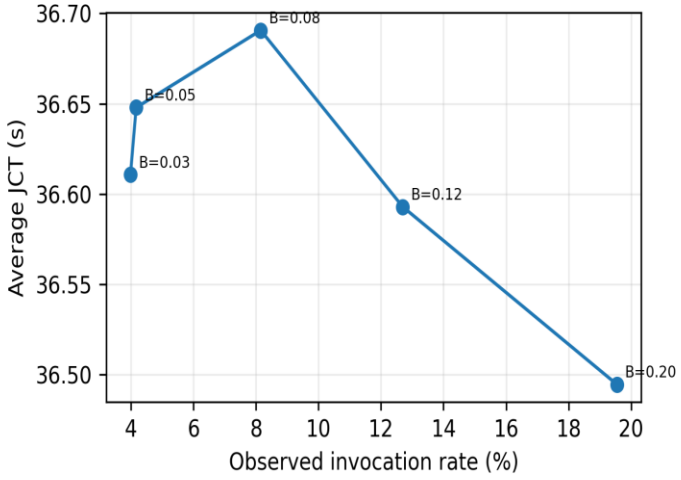


Figure 4CRAI budget sensitivity. Larger budgets buy progressively more planner calls, while mean JCT remains on a broad plateau over the tested range.

E. Resource Utilization and Locality

The policy vocabulary creates different placement behaviors even when average occupancy is close. Across all runs, CRAI achieves a 51.87% locality-hit rate, compared with 50.24% for Always-LLM, 50.79% for Periodic-10, and 58.16% for the much more frequently invoked State-Delta controller. The highest locality fraction therefore does not coincide with the lowest JCT: State-Delta spends 47.17% of active epochs on refreshes and still has higher mean JCT than CRAI. This supports the use of a multi-signal regret proxy rather than locality or load change alone as the invocation criterion.

F. Latency-Equivalent Exposure

Ding et al. report a 342.7 ms P50 end-to-end LLM inference latency in their scheduler and show that triggering/caching can keep expensive reasoning off most scheduling decisions [16]. Using 342.7 ms only as an external reference parameter—not as a latency measured in our simulator—the mean call fractions correspond to 32.7 ms of LLM-latency-equivalent exposure per active epoch for CRAI, 34.6 ms for Periodic-10, 161.7 ms for State-Delta, and 342.7 ms for Always-LLM. This conversion illustrates why invocation policy is a systems concern even when the strategic model improves scheduling quality.

G. Paired Statistical Summary

Table 4 Selected paired CRAI differences over 15 runs.

Baseline	Metric	Difference [95% CI]	p
Static	JCT	-0.546 [-0.924,-0.168]	0.0078
Static	SLA pp	-1.085 [-1.663,-0.506]	0.0013
Periodic	JCT	+0.001 [-0.207,+0.209]	0.993
State-Delta	JCT	-0.298 [-0.564,-0.033]	0.030
State-Delta	Inv. pp	-37.628 [-40.739,-34.518]	<1e-10
Always	JCT	+0.491 [+0.188,+0.794]	0.0037
Always	Inv. pp	-90.456 [-91.058,-89.853]	<1e-10

The tests quantify the intended tradeoff rather than forcing a single winner across all metrics. CRAI is significantly faster than the static and state-change baselines in mean JCT, while Always-LLM remains significantly faster than CRAI by roughly half a second. The invocation reduction versus State-Delta and Always-LLM is large and stable. SLA differences are smaller: CRAI substantially improves on Static-

Balance, is statistically indistinguishable from Periodic-10, and trades 0.253 percentage points against the more aggressive State-Delta trigger.

VII. DISCUSSION AND DEPLOYMENT IMPLICATIONS

A. Why Regret-Aware Triggering Works

A raw state-change trigger implicitly assumes that distance in telemetry space correlates with distance in optimal action space. The normal-load experiments show that this assumption can be wasteful: state features continue to move, but the policies converge to nearly identical outcomes. CRAI instead learns whether invoking would change the proxy loss. The resulting gate naturally suppresses calls when several policy profiles are equivalent and becomes more responsive when queue pressure and deadline risk make policy choice consequential.

B. Strategic LLM, Deterministic Executor

Restricting the LLM to a small strategic action vocabulary has two benefits. First, feasibility is independent of natural-language correctness because every task placement passes deterministic CPU and memory checks. Second, the invocation controller learns the value of a policy refresh rather than the value of unconstrained textual output. This architecture is aligned with the broader LLM-modulo view that language models are useful sources of heuristic guidance when paired with formal or algorithmic executors [27], [28], and it complements hybrid reasoning-plus-optimization architectures for systems control.

C. Scope of the Evaluation

The experiments are designed to isolate invocation control. The workload is synthetic but trace-calibrated, and the strategic planner interface is frozen. Consequently, the measured JCT, SLA, utilization, and call-rate numbers are direct outputs of the supplied simulator, while they do not conflate those outcomes with live-model sampling variability or network/API latency. A deployment can substitute a live LLM behind the same policy-token interface without changing the gate or feasibility-preserving executor. The most important deployment quantity to re-estimate is the proxy-regret mapping, which CRAI learns online from refresh events.

D. Practical Tuning

The target budget B is the primary operations parameter. A low B is appropriate when hosted inference is expensive or the cluster is dominated by stable workloads; a higher B is appropriate when deadline pressure changes quickly or strategic-model cost is low. The dual step η governs how quickly the gate corrects budget deviation, while β controls optimism under uncertainty. Because the predictor has only nine features, these parameters can be tuned with short replay windows rather than retraining a large policy model.

E. Control-Plane Integration

CRAI is positioned above the placement loop rather than inside the per-node scoring path. A scheduler can compute the nine gate features from existing queue and node telemetry, evaluate the gate locally, and contact the strategic planner only after the gate opens. The returned token updates the cached profile; subsequent epochs execute locally until another refresh is justified. This organization makes invocation latency additive only on refresh epochs and keeps routine scheduling independent of external model availability.

The explicit dual price also exposes a measurable service objective for the reasoning layer. Operators can monitor the realized call fraction beside JCT and SLA metrics and adjust B according to inference cost or control-plane latency targets. Because the budget is expressed as a fraction of active scheduling epochs, it remains interpretable when the number of idle epochs changes, and it directly matches the unit in which hosted-model calls are consumed.

VIII. CONCLUSION

This paper reframes LLM-assisted cloud scheduling around a simple systems question: not whether an LLM can reason about scheduling, but when that reasoning is valuable enough to invoke. CRAI answers this question with a calibrated regret-aware gate that predicts the cost of retaining a stale strategic policy and compares it with a budget-dependent dual price. Deterministic placement preserves feasibility, while the

expensive model is reserved for strategic refreshes. Across normal, high-load, and bursty workloads, CRAI uses the planner on 9.54% of active epochs—90.46% fewer calls than always invoking—while staying within 1.35% of the always-invoke mean JCT. The strongest benefits appear under high load, whereas normal-load results show that repeated LLM calls can be almost entirely redundant. These findings support adaptive invocation as a distinct design layer for efficient LLM-assisted resource management.

REFERENCES

Yue, X., & Zhu, R. (2024). An explainable hybrid machine learning framework for cash-flow-conditioned multi-horizon liquidity risk early warning: An SME-oriented benchmark study. World Journal of Management Science, 2(1), 63-72.

Jiao, Y., & Liang, Y. (2024). An integrated process mining and explainable machine learning framework for bottleneck and internal control risk detection in the multi entity financial close. Social Science and Management, 1(3), 96-107.

Bian, M., Wang, Y., & Lin, Y. (2024). Explainable supplier risk prediction for critical telecommunications infrastructure procurement: An empirical XGBoost–SHAP framework. Social Science and Management, 1(1), 64-77.

Liang, Y., & Jiao, Y. (2024). Real-time financial fraud detection using a feature-light temporal dynamic graph neural network with contrastive learning. Journal of Trends in Finance and Economics, 1(1), 76–84.

Shang, W., & Chen, T. (2024). PrefixKD-Edge: Real-time locomotion-intent prediction with prefix self-distillation on lightweight temporal networks. Innovation and Technology Studies, 1(1), 30–38.

Rhee, M., Zou, J., & Qin, Y. (2024). CLARITY: A closed loop framework for duplicate aware failure prediction and budgeted automated remediation in cloud infrastructure. AI and Data Science Journal, 1(1), 76-85.

M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, “Omega: Flexible, Scalable Schedulers for Large Compute Clusters,” in Proc. EuroSys, 2013, pp. 351–364, doi: 10.1145/2465351.2465386.

B. Hindman et al., “Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center,” in Proc. NSDI, 2011.

V. K. Vavilapalli et al., “Apache Hadoop YARN: Yet Another Resource Negotiator,” in Proc. ACM SoCC, 2013, Art. 5, doi: 10.1145/2523616.2523633.

B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, “Borg, Omega, and Kubernetes,” Commun. ACM, vol. 59, no. 5, pp. 50–57, 2016, doi: 10.1145/2890784.

A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, “Dominant Resource Fairness: Fair Allocation of Multiple Resource Types,” in Proc. NSDI, 2011.

R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, “Multi-Resource Packing for Cluster Schedulers,” in Proc. ACM SIGCOMM, 2014, pp. 455–466, doi: 10.1145/2619239.2626334.

I. Gog, M. Schwarzkopf, A. Gleave, R. N. M. Watson, and S. Hand, “Firmament: Fast, Centralized Cluster Scheduling at Scale,” in Proc. OSDI, 2016, pp. 99–115.

K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, “Sparrow: Distributed, Low Latency Scheduling,” in Proc. SOSP, 2013, pp. 69–84, doi: 10.1145/2517349.2522716.

C. Delimitrou and C. Kozyrakis, “Quasar: Resource-Efficient and QoS-Aware Cluster Management,” in Proc. ASPLOS, 2014, pp. 127–144, doi: 10.1145/2541940.2541941.

G. Ding, S. Yang, H. Lin, Z. Chen, and J. S. Yang, “LLM-Driven Adaptive Cloud Resource Scheduling: Bridging Reasoning Intelligence With Optimization Guarantees,” IEEE Open Journal of the Computer Society, vol. 7, pp. 560–573, 2026, doi: 10.1109/OJCS.2026.3667549.

E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, “Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms,” in Proc. SOSP, 2017, pp. 153–167, doi: 10.1145/3132747.3132772.

R. Grandl, M. Chowdhury, A. Akella, and G. Ananthanarayanan, “Altruistic Scheduling in Multi-Resource Clusters,” in Proc. OSDI, 2016, pp. 65–80.
C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, “Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis,” in Proc. ACM SoCC, 2012, Art. 7, pp. 1–13, doi: 10.1145/2391229.2391236.
M. Tirmazi et al., “Borg: The Next Generation,” in Proc. EuroSys, 2020, Art. 30, pp. 1–14, doi: 10.1145/3342195.3387517.
R. Bhardwaj et al., “Cilantro: Performance-Aware Resource Allocation for General Objectives via Online Feedback,” in Proc. OSDI, 2023, pp. 623–643.
J. Gu et al., “Tiresias: A GPU Cluster Manager for Distributed Deep Learning,” in Proc. NSDI, 2019, pp. 485–500.
Q. Weng et al., “MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters,” in Proc. NSDI, 2022, pp. 945–960.
T. Brown et al., “Language Models are Few-Shot Learners,” in Advances in Neural Information Processing Systems, vol. 33, 2020.
Fang, Q., & Liu, W. (2025). HARLA-ED: Resolving Information Asymmetry and Enhancing Algorithmic Symmetry in Intelligent Educational Assessment via Hybrid Reinforcement Learning. Symmetry, 18(1), 58.
Mai, N. T., Cao, W., & Fang, Q. (2025). A study on how LLMs (eg GPT-4, chatbots) are being integrated to support tutoring, essay feedback and content generation. Journal of Computing and Electronic Information Management, 18(3), 43-52.
Wang, Y., & Xing, S. (2025). AI-driven CPU resource management in cloud operating systems. Journal of Computer and Communications, 13(06), 135-149.
Xing, S., & Wang, Y. (2025). Cross-modal attention networks for multi-modal anomaly detection in system software. IEEE Open Journal of the Computer Society.
Chen, T., Shang, W., & Guo, Z. (2024). Xsla: An SLA-aware heterogeneous XPU scheduling framework for edge AI inference. World Journal of Information Technology, 2(2), 48–59.
Qin, Y., Zou, J., & Rhee, M. (2022). An AI-driven intelligent transaction execution framework for internet finance platforms: Cost-sensitive real-time integrity interdiction. Eurasia Journal of Science and Technology, 4(2), 36–44.
Wang, Y., & Zhang, C. (2023). AERIS: Adaptive executor scoped resource management for privacy preserving data processing in Apache Spark. Eurasia Journal of Science and Technology, 5(2), 25-33.
Fang, Q. (2024). True-EWS: Trustworthy early warning with equalized conformal guarantees for equitable student support in U.S. education. Educational Research and Human Development, 1(2), 44–54.