One State Variable Was the Whole Effect: A Controlled Study of Offline-to-Online Reinforcement Learning for Cluster Scheduling

Anonymous authors
Paper under double-blind review

Abstract

Pretraining a cluster scheduler on the production scheduler's placement log, fine-tuning online, and protecting the learned policy with a safety layer and an out-of-distribution fallback is a common recipe, yet its offline premise is rarely tested. We evaluate this pipeline on a Google production trace replayed in a capacity-constrained simulator. The offline premise fails: production placements show little predictable dependence on system load and cannot be reliably recovered from any shortlist we can construct. The deployed 64-slot shortlist recalls 0.43% of them, and a $1{,}187$-candidate shortlist containing every machine a job has ever used reaches only 13.92%, making behaviour cloning the poorest-performing model we train. Two implementation defects initially obscured this failure: using only one gradient step per update and a policy head degenerate at initialisation. The dispersion of candidate logits, rather than policy loss, reveals both defects. After correcting the pipeline, the learned policy outperforms the production scheduler. However, its apparent advantage over heuristic baselines disappears when those baselines are given the co-runner count, a state variable omitted from the original feature set: adding this variable shifts the performance frontier by 5.61 percentage points and reduces the learned policy's apparent wins from 9 of 20 operating points to 0 of 20, while denser sampling within the same heuristic family does not change the result. We therefore argue that baseline feature coverage should be established before attributing performance gains to learning. Replication on two additional traces, including one from a different operator, yields the same qualitative finding while showing that the magnitude of the gain depends strongly on simulator capacity: the gain is 0.29 percentage points at the trace's native cluster size but 43.3 percentage points in the capacity-shrunk setting. Thus, the observed improvement should not be interpreted as a general ranking of the production scheduler. Finally, a deployable takeover-and-fallback gate reduces catastrophic episodes from 14 of 50 to 1 of 50. The source code and the result artifacts behind every number reported here are provided in the supplementary material.

1 Introduction

Learning to schedule cluster workloads has an appealing recipe: collect the production scheduler's own placement decisions as an offline dataset, pretrain on them, fine-tune online, and guard the result with a safety layer and an out-of-distribution fallback. Its components are standard: imitation-initialised scheduling (Kang et al., 2024; Li et al., 2024; Venkataswamy et al., 2024), conservative offline RL with online fine-tuning (Kumar et al., 2020; Nakamoto et al., 2023), and runtime shielding (Alshiekh et al., 2018; Dalal et al., 2018). But its first step rests on an assumption rarely reported: that production placements are a useful imitation target. Two numbers decide it: whether the production action is what a competent rule would choose, and whether the action space can represent it. Imitation-based work trains against a *heuristic or solver* expert in a simulator, and the one work reporting an action-agreement metric reaches 95–99% with its heuristic expert (Venkataswamy et al., 2024): what happens when the expert is a real deployed scheduler?

We replay 7,087,379 real placement decisions from the clusterdata-2011-2 trace and two further traces through a capacity-constrained simulator. The learned policy's entire apparent advantage

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

over heuristic rules turns out to equal the value of one state variable those rules had been denied. After fixing two silent bugs it dominates the production scheduler and escapes the frontier of a parameterised greedy rule family on 9 of 20 operating points; giving that family the co-runner count n_{on} moves the frontier by 5.61 pp and drives the escapes to 0 of 20, while densifying its grid from 4 to 76 points moves it by exactly 0.00 pp. The bottleneck was not resolution but which state variables the baseline could see. Two results bound it: the comparison replicates on two more traces, where a single-parameter rule already suffices, and a like-for-like paired comparison shows the advantage is real in kind but set in magnitude by how far the cluster is shrunk.

Contributions.

- **A measurable admission test for learning from scheduling logs**: the chosen machine sits at load percentile 0.429 against 0.500 for uniform random (0.586 on measured usage), and a 1,187-candidate shortlist containing every machine the job has ever used recalls 13.92% of production actions against 0.43% for the deployed 64-slot one.
- **Two silent failure modes that manufacture false negatives**: a policy head degenerate at initialisation (logit standard deviation 10^{-4}, softmax exactly uniform) and one gradient step per update. Both masquerade as training-signal collapse; our own first diagnosis was wrong.
- **The baseline's feature coverage was the whole effect, and the fix is a reusable criterion**: parameter resolution is worth 0.00 pp over $4 \to 76$ points and feature coverage 5.61 pp for one variable, so we propose handing a baseline one more realised state variable and checking whether the frontier moves. The resolution half restates a known criticism; the coverage half is new. The one deployable result is a takeover-and-fallback gate cutting 11.28% SLA violations and 14 of 50 catastrophic episodes to 4.90% and 1 of 50, against a heuristic reference of 3.65%.
- **Cross-trace replication, and the scope of the domination claim**: the comparison reproduces on a second cell of the same operator and on a different operator's trace, and a paired one-step counterfactual shows the advantage is real in kind but runs from 0.29 pp at the trace's own cluster size to 43.3 pp in the shrunk world.

Section H states the boundaries and what would overturn each claim.

2 Related Work

Learned schedulers inside simulators. The dominant line learns a scheduling policy by interaction inside a simulator, using a trace as the generator of arrivals and job sizes rather than of policy supervision (Mao et al., 2019; 2016; Zhang et al., 2020); Gu et al. (2025) review its algorithmic variants, and recent work benchmarks on real traces (Jiang et al., 2026) or trains against several operators' traces (Dongare et al., 2025).

Imitation-initialised scheduling policies, and who the expert is. A second line warm-starts scheduling RL with behaviour cloning (Kang et al., 2024; Guo et al., 2021; Li et al., 2024; Venkataswamy et al., 2024); whether it bears on our setting turns on the provenance of the demonstrations. Where we could confirm it, the expert is a heuristic or solver: Li et al. (2024) use WFP/FCFS/UNI/SJF/F1 sequences, and Launchpad rolls out SJF/QoS/FCFS/HVF experts in its authors' simulator (Venkataswamy et al., 2024). To our knowledge no study imitates a production scheduler's own placement decisions while reporting their imitability on a real trace; this is a hedge, since our search was keyword-based. Launchpad bounds the claim in both directions: it already reports an action-agreement metric, so the instrument is not new, but its 95–99% agreement is with a *heuristic* expert; our finding therefore concerns *production* decisions rather than scheduling demonstrations in general. Reading a log is also not imitating it: Wang et al. (2025) mine runtime logs, source code and job history to *feature* jobs for an existing scheduler, leaving its placement rule untouched.

Offline datasets, coverage, and the case for an online stage. Offline RL theory identifies distributional shift as the core obstacle when the dataset does not cover the optimal policy's behaviour (Levine et al., 2020), motivating conservative methods and the pretrain-then-fine-tune recipe (Kumar et al., 2020; Kostrikov et al., 2021; Nakamoto et al., 2023). Recent scheduling work sharpens this

into a statement about data: van Remmerden et al. (2026) find scheduling quality depends more on coverage of the state–action space than on trajectory quality, and offline RL is by now applied across scheduling problem classes (Echeverria et al., 2025). This is the same axis as our recall bound, and it cuts against our framing, since the remedy for poor coverage may be broader offline data rather than an added online stage; we therefore treat the online-stage rationale as one of two hypotheses (Section 5.4).

Safe RL: shielding, safety layers, and novelty detection. Constraining a learned policy at runtime is well established: shields correct unsafe actions before execution (Alshiekh et al., 2018), the safety layer of Dalal et al. (2018) projects the action onto an analytically safe set (proposed for datacentre control), and García & Fernández (2015) survey the field. Shielding has since reached real-time scheduling (Shi & Chen, 2025), data-centre task selection and server assignment (Zhang et al., 2026), and production HPC resource management (Albaarini et al., 2026); the Mahalanobis score we use is a standard novelty rule (Lee et al., 2018). We found no *takeover-and-fallback* controller for cluster placement: the shields above restrict or project the action rather than hand control back to a rule, so Section 6 contributes the application and its quantification.

Negative results and baseline fairness. Our methodology draws on work about how machine-learning results fail: point estimates from few runs are unreliable (Agarwal et al., 2021; Henderson et al., 2018), advances shrink under a uniform protocol because baselines are under-tuned (Musgrave et al., 2020; Melis et al., 2017), and attributing a gain to the wrong cause is a named failure mode (Lipton & Steinhardt, 2018).

3 Setup: Traces, Simulator, and the Pipeline Under Test

3.1 Traces and simulator

We use three public traces, all replayed through the same simulator and code path, each recording *which machine* every task was placed on. The main trace is the Google clusterdata-2011-2 trace (Reiss et al., 2012), a Borg-managed cluster (Verma et al., 2015): 55.8 hours, 12,548 machines, 7,087,379 placement decisions, 172,568 tasks never scheduled. The second, the successor trace of Tirmazi et al. (2020) (Google 2019), is the same operator in a different year and cell: 10,001 machines of six types, with identifiers that join to the placement log, making capacity recoverable rather than inferred, though its shards are random row partitions, so a window needs every shard. The third is Alibaba v2018, a different operator: 4,034 machines, all 96-core (the one trace where the simulator's uniform-capacity assumption is exactly right), but with no submission time, so its SLA is not comparable in absolute terms.

Throughout, *load* means the sum of the concurrent resource *requests* of the tasks placed on a machine, because that is what a placement rule can observe; Section 4 recomputes our headline statistics under measured task_usage. *Capacity is a calibration dial on all three traces, not measured hardware*: the main trace is replayed at 0.22, while its observed per-machine peak concurrent request is 1.61, i.e. a cluster about $4.5\times$ smaller than production. Physical capacity leaves every trace uncongested, so what we vary is congestion, not a production workload; Section H lists the simulator's modelling choices.

3.2 The pipeline under test

We study the full recommended recipe (imitation-initialised offline RL, small-step PPO from that initialisation, action projection onto the safe set, and a Mahalanobis out-of-distribution fallback) as specified in Section B. Section 5.4 tests its online stage, whose stated rationale is that the log lacks *packing* behaviour; the competing remedy is broader offline data (Section 2).

3.3 The rule family and the protocol

Our baseline is a parameterised family, so the comparison is against a frontier rather than a point. Each member is a myopic greedy rule; members differ only in which state variables they may read and how those are thresholded, and the family can be given the same state variables as the learner.

Section 5 grants three of them in turn: *best_fit* on remaining capacity with a *headroom threshold* h_r, a *co-runner cap* $n_{\max}$ on n_{on}, and a *queue switch* q_{cur}. We also evaluate four fixed heuristics (*balanced*, *least_allocated*, *random*, *first_fit*) and the production policy itself, *borg*.

We compare operating points on three objectives: SLA violation rate, hotspot fraction (share of machines above a utilisation threshold), and tasks placed. A point *escapes* the family if no rule point dominates it on all three; escapes are counted over 20 learned points from four training runs crossed with five gate settings, relative to the family Section 5 names. Comparisons are paired across evaluation windows, separating three variance sources: the window (8, paired), the evaluation seed for one network (7 draws), and the training seed (4 runs); Table 4 in Section F lists the notation.

4 The Offline Premise Fails, and Two Bugs That Hid It

The first step of the recipe assumes that production placements are worth imitating; on this trace that is false, in two independent ways.

4.1 Two admission tests

For each of the 7,087,379 placement decisions we compute the load percentile of the machine the scheduler actually chose, where 0 is the emptiest machine at that instant (Figure 7B in Section F). The median is 0.429 against 0.500 for uniform-random placement, so the scheduler sits slightly below random and nowhere near optimal. This is not a narrow-load artefact: the chosen machine sits at 94.7% of the median cluster load although an idle machine is available at every decision, and measured task_usage moves the median to 0.586, to the *busier* side of random.

Imitability also depends on whether the action space contains the production action. The deployed 64-slot shortlist recalls 0.43% of production actions; a 1,187-candidate shortlist containing *every machine the job has ever used* raises this to 13.92% (Figure 7A in Section F), so even with hindsight knowledge of each job's history, 86% of production actions remain outside the action space.

4.2 The consequence, and the scope of this claim

Those measurements predict a performance signature, and it appears. In the evaluation window where the contest is decided every policy attains 0% SLA, so hotspot is the only discriminating quantity. Behaviour cloning, the component the recipe is built on, is the *worst* of the three models we train at 46.55%, worse than the production scheduler's 35.44% and *random*'s 36.80%, while a single-step regression model and CQL reach 31.03% and 31.11%. Offline value learning beats cloning and the production scheduler, so the log is not useless as a training signal, only as an imitation target; it is nonetheless only level with the plainest heuristics, *least_allocated* at 31.34% and *balanced* at 31.06%; Section 5 therefore compares against a family rather than a rule.

Two regularities dominate the log (Section B): 98.9% of decisions extend a job already running, and the chosen machine does *not* already host a task of the same job 68.1% of the time. The log shows a job's tasks being spread and kept apart, so a family that reads only load is not competing on the same objective. This is a result about a *production* scheduler's decisions and does not transfer to scheduling demonstrations in general: with a heuristic or solver expert, behaviour cloning agrees with the expert 95–99% of the time (Venkataswamy et al., 2024). The contribution is the measurement, not the recall instrument. Three explanations are consistent with our data and the log alone cannot separate them (Section B).

4.3 Two silent failure modes that manufacture a false negative

Before any comparison can be trusted, the pipeline has to be capable of learning at all. Ours was not, and it failed in a way that produced a confident, reproducible-looking negative result.

Mode 1: one gradient step per update. The update consolidated the whole rollout's loss into a single scalar and called backward() once, so eight "updates" performed eight gradient steps, folding all three epochs over $8{,}812$ transitions into each one; batching the same work into minibatches of 512

transitions raises that to roughly 52 gradient steps per update. *Check:* count the backward() calls per update, not the losses.

Mode 2: a policy head degenerate at initialisation. The scorer's logits ranged from 0.3105 to 0.3110 across the 64 candidates, a standard deviation of 10^{-4}, so softmax was exactly uniform and no action was ever preferred. The cause is structural rather than a matter of scale: a bilinear form was factorised for cost, and an action-independent bias shifts *all* logits equally, a direction to which softmax is invariant, so the gradient could not change the policy. *Check:* measure the standard deviation of the logits across candidates before and after training; if it is negligible, every gradient lies in that invariant direction. The policy loss is *expected* to be near zero here. Our first diagnosis, that the advantages had collapsed, was wrong (their measured standard deviation was 0.962). Both defects are unrelated to cluster scheduling. After fixing them the policy learns across two independent runs, with the logit spread growing from 0.061 to 0.863 (Table 7; Section B).

5 The Decisive Control, and How Far It Holds

With the pipeline fixed, the learned policy dominates the production scheduler and beats the heuristics we started with. Two questions decide how much that carries: whether the comparison is fair (Section 5.1, which replaces the end-to-end number with the strongest surviving form of the claim) and whether the advantage over heuristic *rules* is real (Section 5.2–Section 5.3).

5.1 The end-to-end comparison, and its like-for-like replacement

Our *borg* baseline is not *borg*. It replays the log, placing each task on the machine the production scheduler chose and falling back to *least_allocated* when that machine no longer fits: "the log plus a naive fallback". The smaller the cluster, the more often the fallback fires: at the main operating point (h37, capacity 0.22) only 36.4% of decisions still use the logged machine, against 99.8% at physical capacity (Figure 1 in Section F). The hotspot gap to the best fixed heuristic grows with the fallback fraction, from 0.07 pp at physical capacity to 1.26, 3.95 and 5.21 pp at 0.60, 0.40 and 0.22. At physical capacity all three traces put the production scheduler within 0.69 pp of the best rule, with SLA tied everywhere (0.07 / 0.69 / 0.00 pp for 2011 / 2019 / Alibaba), so the "domination" appears only after the cluster is deliberately shrunk, which is why we do not rest the claim on it.

A paired one-step counterfactual. The comparison cannot be repaired in place: when the logged machine does not fit, the release does not record the counterfactual. We therefore score, from the *same* pre-placement state, both the logged machine and each rule's choice, on four objectives (imbalance, hotspot, remaining free fraction, scarce-slot loss), restricted to decisions where the logged machine *was* feasible, so the gap cannot come from the fallback. That subset is $19{,}779$ of $54{,}328$ sampled decisions (36.4%). On it the log's own choice creates an immediate hotspot on 43.4% of decisions and a one-parameter rule (*greedy1*, $h_r = 0.10$) on 0.09%, a paired difference of -0.4328 ± 0.0069 (95% CI); *balanced* gives -0.4252 ± 0.0069, *least_allocated* -0.3478 ± 0.0076, and *best_fit* is *worse* than the log at $+0.5663 \pm 0.0069$. The family's advantage is a real scheduling-quality difference, not a straw man (Figure 1; Table 5 in Section F).

The scope restriction that travels with it. At the trace's own cluster size the same measurement gives only 0.29 pp, against 2.02 pp on Google 2019 and exactly 0.00 pp on Alibaba, where the production scheduler is optimal for this objective in its own world. The magnitude runs from 0.29 to 43.3 pp as a function of one dial: the logged choice was made for a machine about $4.5\times$ larger, so the same relative placement is far tighter in the shrunk world. *That confound lives in the world, not in the method*: in a deliberately shrunk world the replayed production actions are too tight; the scheduler is not bad (Section B).

5.2 A frontier, not a rule, and how the escape dissolves

To distinguish "learning helps" from "the baseline was weak" we sweep the frontier of the parameterised family of Section 3.3, granted the learner's state variables. The learned policy escapes it: at h37 and capacity 0.22 the best learned point (whose gate mixes a 57.5% learned policy with a

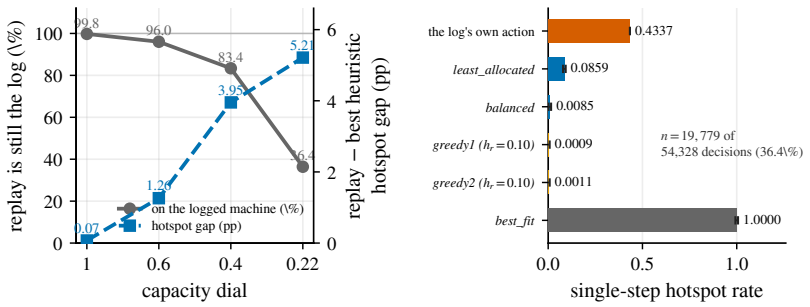

Figure 1: The advantage is real; the magnitude is manufactured (C8, Section 5.1). **(A)** Against the capacity dial, from the trace's physical cluster size to the shrunk world of the main result: the share of decisions still using the logged machine, and the hotspot gap to the best fixed heuristic. **(B)** The paired one-step comparison at capacity 0.22, restricted to the 19,779 decisions (36.4%) where the logged machine was feasible; bars are single-step hotspot rates.

42.5% fallback to *balanced*, Section B) reaches 19.24% SLA at a hotspot fraction of 31.00% while placing 62,097 tasks, against 20.40% for the best rule there: a lead of 1.16 percentage points, and the largest apparent win we observed anywhere. Add state variables one stage at a time and the escapes disappear (Figure 3A, Table 1): 19 of 20 against the fixed heuristics, 13 once h_r is granted, 9 with the queue switch, and **0** once the co-runner cap $n_{\max}$ lets the family read n_{on} (111 points). The frontier at a hotspot budget of 35% moves from 20.40% to 14.79% (5.61 pp), and the learned policy's best value there is 17.82%, above the frontier rather than on it (Figure 2).

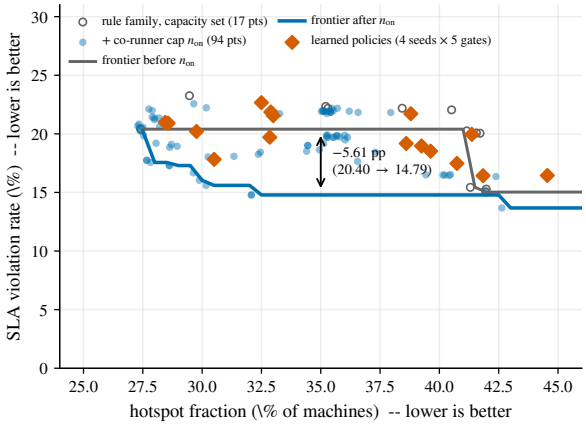

Figure 2: Rule-family frontier in the (SLA, hotspot) plane with the 20 learned operating points. Granting the family n_{on} moves the frontier below every one of them; lower-left is better. Window h37, capacity 0.22.

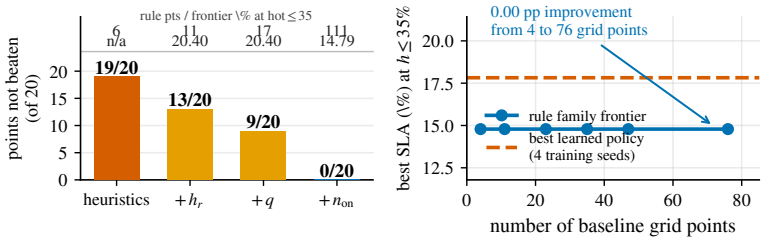

Figure 3: The baseline's feature coverage was the whole effect. **(A)** Learned points the family fails to beat as it is granted, in turn, h_r, q and n_{on}; the last removes every apparent win, moving the best SLA at a hotspot budget of 35% from 20.40% to 14.79%. **(B)** The same family on a nested grid from 4 to 76 points: the frontier does not move.

The three-objective criterion's placed$\geq$ clause is a knife edge: the same post-fix runs yield 4 of 20 escapes at no tolerance and 2 of 20 at a 0.5% throughput tolerance, and both remaining margins (+0.36 pp, +0.12 pp) vanish on windows the networks were not selected on (0 of 12; Section B). We report the two-objective count.

Table 1: The feature-coverage ladder (window h37, capacity 0.22): escapes as the family is granted one state variable after another, over the 20 learned operating points. Frontier is the lowest SLA at hotspot $\leq 35\%$; a point escapes if no rule point dominates it on (SLA, hotspot). The three-objective variant of the same runs gives 4 of 20 escapes, and a 0.5% throughput tolerance cuts the count to 2 of 20 (results/feature_fix_escape_check.txt).

rule family	added variable	points	escapes	frontier SLA%
baseline heuristics only	none	6	19/20	n/a
+ headroom threshold h_r	h_r	11	13/20	20.40
+ queue switch	queue	17	9/20	20.40
+ co-runner cap n_{on}	n_{on}	111	0/20	14.79

Table 2: Per-capacity comparison, window h37, $\kappa = 0$. An unplaced task counts as an SLA violation, so *placed* must be read with SLA.

	borg (production)			*balanced*		best rule point within *borg*'s hotspot budget		
cap.	SLA%	hot%	placed	SLA%	placed	rule	SLA%	ΔSLA
0.12	54.99	54.03	36,617	56.39	37,187	*least_allocated*	56.05	+1.06
0.15	47.47	54.88	42,298	47.47	40,089	*hr=0.100*	46.06	−1.41
0.22	22.05	40.51	54,328	22.17	55,105	*hr=0.100*	20.40	−1.65
0.30	0.00	19.22	62,430	0.00	62,430	*balanced*	0.00	+0.00
0.45	0.00	6.47	62,430	0.00	62,430	*balanced*	0.00	+0.00

5.3 Resolution was never the bottleneck: a reusable criterion

A natural objection is that 17 points is a coarse grid. It is not: a nested ladder over the two knobs that move the frontier, h_r and $n_{\max}$, holds the frontier at 14.79% from 4 to 76 points in six rungs, improving by at most +0.00 pp between rungs (Figure 3B). The frontier is saturated at four points, so the 5.61 pp came entirely from *which variable* the family could read: *parameter resolution* is worth 0.00 pp here, and *feature coverage*, meaning which realised state variables the baseline may read, 5.61 pp. The resolution half is a known criticism (Musgrave et al., 2020; Lipton & Steinhardt, 2018); the coverage half yields a criterion we have not seen stated: *before concluding that a learned policy beats a rule-based baseline, hand that baseline one more realised state variable, and check whether the frontier moves.*

5.4 The online stage's rationale, and the tension it would need

A negative result is only as useful as the boundaries drawn around it. The online stage's rationale (Section 3.2) is that the log lacks packing behaviour, so trial and error must supply it. It does not: across 24 half-hour windows, pure myopic *best_fit* attains the lowest SLA violation rate in 23 of the 24 (Section A), and when we add the tension explicitly, as a co-location cost $1 + \kappa\, c_o$ on each newly placed task, the learned family needs a high weight to become preferable: at $\kappa = 0$ no α up to 10 suffices, at $\kappa = 2.0$ the crossover arrives only at $\alpha \in [3.5, 5.5]$ and buys 0.181 pp ($\approx 0.3\%$ relative), and for $\kappa \geq 0.5$ a two-parameter rule absorbs the cost with no learning (Table 10). The direction matches the hypothesis; the magnitude does not.

5.5 Where the result holds, and what the feature-path correction changed

Inside 0.15–0.45 the rule family 3-objective-dominates the production scheduler at every capacity: by −1.41 pp SLA at 0.15 and −1.65 pp at 0.22, while at 0.30 and 0.45 both reach 0% SLA and full throughput, so dominance reduces to hotspot and those capacities are uninformative rather than favourable. At 0.12 the family loses, its best rule reaching 56.05% against the production scheduler's 54.99% at comparable hotspot because saturation leaves 51–70% of machines over-utilised. The boundary is empirical, found by sweeping rather than derived (Table 2).

Across 8 paired evaluation windows the family dominates in all 8, by -10.52 ± 8.72 pp SLA and -5.95 ± 3.26 pp hotspot, both 95% intervals excluding zero. The evaluation seed for one network moves SLA by ±0.2–0.6 pp, and the training seed over four runs and five gate settings ($n = 20$) gives $18.34 \pm 1.04\%$ SLA and $38.60 \pm 2.99\%$ hotspot (Agarwal et al., 2021; Henderson et al., 2018) (Table 9 in Section F).

Those bands are post-fix: a feature-path defect wrote several channels as though machine capacity were 1.0, and fixing it moves them by -0.30 pp and -0.35 pp, inside the ± 2.7 pp seed spread (Section B).

5.6 Does the result hold on other traces?

External validity was our largest remaining weakness, so we replicated the protocol on two further public traces through the same code path: **Google 2019**, the successor trace of Tirmazi et al. (2020), and **Alibaba 2018**, a different operator whose congestion is a cliff rather than a slope (Section 3.1). The negative result reproduces: the family 3-objectively dominates *borg* on the main trace at 0.15, 0.22, 0.30 and 0.45, on Google 2019 at 0.20, 0.25 and 0.30, and on Alibaba at 0.30, where 17 rule points dominate simultaneously. On Google 2019 at 0.20 a single rule cuts SLA from 51.63% to 9.47% and hotspot from 27.86% to 9.04% with throughput unchanged, though 76.5% of decisions there still use the logged machine (70.5% on Alibaba; Figure 4, Table 6 in Section F). Two cells saturate: the main trace at 0.12 and Alibaba at 0.25. On the new traces the best dominators are typically *single-parameter* rules (*greedy1*, $h_r = 0.10$) or fixed heuristics, so the result there does not even require handing the family n_{on}. Heterogeneity is unmodelled even on Google 2019, where the identifiers to model it exist.

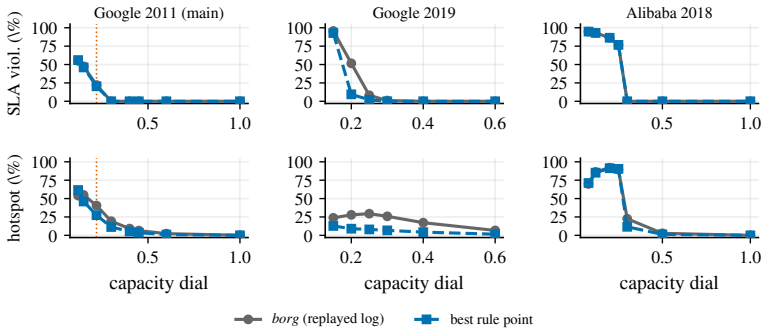

Figure 4: The same protocol on three traces (C7, Section 5.6): one code path, one rule family, one objective triple, against the capacity dial. The dotted line marks the main operating point; a rule marker above a *borg* marker means the family cannot beat the replay within the replay's own budget (the two saturated cells). Alibaba's SLA is not comparable.

6 What Survives: Safe Takeover and Fallback

The one mechanism that holds up is the safety gate; it is the only positive claim in this paper.

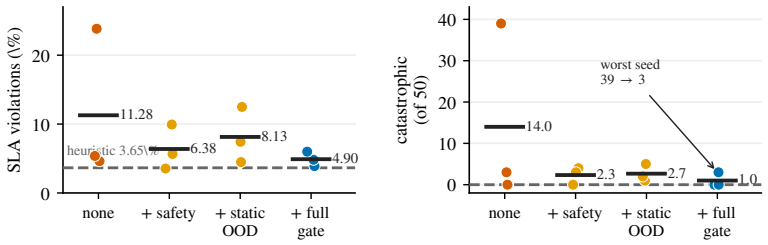

Figure 5: What a safe-takeover gate buys. All four rungs are the same policy, evaluated in the training regime; dots are seeds ($n = 3$), rules are seed means. The rungs add, in order, the safety layer, a static OOD fallback, and the full adaptive gate. **(A)** SLA violations against the heuristic reference. **(B)** Catastrophic episodes per 50-episode run. The bare policy's mean of 14.0 comes from one seed at 39, so the gate's contribution is capping the worst seed at 3.

6.1 The failure mode the gate addresses

Online fine-tuning is where the pipeline can do real damage. Starting PPO from scratch on a shifted workload, with neither offline initialisation nor protection, degrades the policy catastrophically: SLA

violations rise from the 3.65% heuristic reference to 11.28% (standard deviation 10.89) and 14 of 50 episodes are catastrophic. We add protections one at a time to that same policy (Figure 5, Table 8 in Section F).

Adding protection one rung at a time gives 6.38% and 2.33 catastrophic episodes for the safety layer, 8.13% and 2.67 for a static Mahalanobis cutoff, and 4.90% (standard deviation 1.05) with 1.00 catastrophic episodes at a takeover rate of 65.4% for the full adaptive gate that tightens when the observed safety metric exceeds budget. The static fallback fails for an instructive reason: a threshold set on the offline score distribution still lets the degraded policy take over 88% of decisions.

A fallback controller keyed only on state novelty will not protect a policy that degrades while remaining in distribution; it has to key on the safety metric itself.

6.2 Reading the ladder honestly, the detector, and the residual gap

Every rung is $n = 3$ seeds, and with $n = 3$ the mean hides what matters: the bare policy's mean of 14.0 catastrophic episodes comes from a single seed at 39, so the informative summary is that the *worst* seed falls from 39 to 3. The four rungs aggregate to 6.01 ± 2.55 and 1.8 over $n = 12$, so the $n = 3$ figure is not taken for the whole picture.

The Mahalanobis score driving the gate separates the regimes well but not perfectly: AUROC 0.925, true-positive rate 0.78 at a 5% false-positive rate (Figure 6 in Section F). It cannot resolve states novel in queue depth but ordinary in utilisation, where the policy is both degraded and unflagged.

On the real trace the gate moves the same policy along its frontier without retraining: at a hotspot budget of 35%, hotspot falls from 35.6% to 30.6% at a cost in SLA of $18.9\% \rightarrow 21.1\%$, at a fallback rate that is a property of the network (4.1–9.1% and 34.0–42.5% for two policies). The gate compresses the failure mode but does not remove it, leaving a 1.25 pp SLA gap to the heuristic (4.90% against 3.65%). The mechanism is established (Section 2); the contribution is that a controller can hold online RL's failure mode to at worst falling back to the heuristic, and that the bound is quantified.

7 Conclusion

We put the recommended offline-to-online RL recipe for cluster scheduling through a controlled study on a real production trace, and found a sharp asymmetry between its two halves. The offline premise fails before any training begins: production placements sit near random on load, and a shortlist containing every machine a job has ever used excludes 86% of its actions. The learned policy, once two silent training defects were fixed, does dominate that scheduler, but its whole apparent advantage over heuristic rules was the value of one state variable the rules had been denied. Handing the family the co-runner count moved the frontier by 5.61 percentage points and eliminated every apparent win, while sampling the same family from 4 to 76 points changed nothing. The comparison also reproduces on a second cell of the same operator and on a different operator's trace.

How far that travels is itself a result: a like-for-like paired comparison settles that the advantage is a real scheduling-quality difference, from identical states where the logged machine was feasible, but one whose magnitude is a function of the capacity dial, from 0.29 percentage points at the trace's own cluster size to 43.3 in the shrunk world. The criterion we would most want a practitioner to carry away is this: before concluding that a learned policy beats a rule-based baseline, give the baseline one more realised state variable and check whether the frontier moves. What survives is narrow but usable: a gate that decides when the learned policy may take over and tightens when the safety metric degrades turns 14 of 50 catastrophic episodes into one, so the downside is quantified rather than unbounded.

AI use statement

In this work, we used generative AI tools for designing feedback on research experiments, implementing methods, cleaning and reformatting datasets, supporting qualitative and thematic data analysis and interpreting results. We have not used generative AI tools for helping develop theoretical models or conceptual frameworks, formulating mathematical claims, and providing critical ingredients for

proving mathematical claims. Generating synthetic data sets, assisting in the writing of proofs, refining hypotheses and assisting with translation are not applicable to this work. Additionally, we used generative AI tools for creating and modifying scientific figures, creating and editing software code, drafting parts of a research paper. We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness both by another AI agent with a different context and by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Ethics statement

This work uses a public, de-identified production cluster trace released by Google for research use; no human subjects, personal data, or private systems are involved. The paper reports a negative result about learning-based scheduling. Its central practical recommendation (that a learned scheduler be deployed only behind a takeover-and-fallback gate, and that the gate react to the safety metric rather than only to state novelty) is a safety-motivated one. We are not aware of a specific harmful application arising from this work that is not already present in the general use of learning-based resource management.

Reproducibility statement

Every quantitative result in this paper is regenerated from raw artifacts in the project repository. Section A maps each claim to the command that produces it and the file it is read from. The simulator, the offline and online training code, the rule-family sweep, the interference and capacity sweeps, and the figure and table generators are all included as source. Hyperparameters, the reward definition and the state-feature set are given in Section B; the notation is defined in Table 4; and the trace window, capacity range and variance axes are stated in Section 3 and Section 5.5. The audit scripts that reconcile the manuscript against the result files are included as well, since we regard that check as part of the method rather than as a one-off.

References

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Mohammad Nour Albaarini, Nouh Sabri Elmitwally, Khalid Ismail, Adel Aneiba, and Sagheer Abbas. Safety-aware reinforcement learning for adaptive resource rightsizing in Slurm-based high-performance computing systems. In *2026 IEEE 15th International Conference on Communication Systems and Network Technologies (CSNT)*, pp. 435–440, 2026. doi: 10.1109/CSNT69054.2026.11502301.

Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. doi: 10.1609/aaai.v32i1.11797.

Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.

Shruti Dongare, Redwan Ibne Seraj Khan, Hadeel Albahar, Nannan Zhao, Diego Melendez Maita, and Ali R. Butt. Hybrid learning and optimization-based dynamic scheduling for DL workloads on heterogeneous GPU clusters. *arXiv preprint arXiv:2512.10271*, 2025.

Imanol Echeverria, Maialen Murua, and Roberto Santana. Offline reinforcement learning for job-shop scheduling problems. *Applied Soft Computing*, 184:113736, 2025. doi: 10.1016/j.asoc.2025.113736.

Javier García and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16:1437–1480, 2015.

Yan Gu, Zhaoze Liu, Shuhong Dai, Cong Liu, Ying Wang, Shen Wang, Georgios Theodoropoulos, and Long Cheng. Deep reinforcement learning for job scheduling and resource management in cloud computing: An algorithm-level review. *arXiv preprint arXiv:2501.01007*, 2025.

Wenxia Guo, Wenhong Tian, Yufei Ye, Lingxiao Xu, and Kui Wu. Cloud resource scheduling with deep reinforcement learning and imitation learning. *IEEE Internet of Things Journal*, 8(5): 3576–3586, 2021. doi: 10.1109/JIOT.2020.3025015.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. doi: 10.1609/aaai.v32i1.11694.

Zhiyun Jiang, Tianming Zhao, Chunqiu Xia, and Albert Zomaya. Atlas: Alibaba dataset and benchmark for learning-augmented scheduling. In *International Conference on Learning Representations (ICLR)*, 2026.

KaiXuan Kang, Ding Ding, HuaMao Xie, LiHong Zhao, YiNong Li, and YiXuan Xie. Imitation learning enabled fast and adaptive task scheduling in cloud. *Future Generation Computer Systems*, 154:160–172, 2024. doi: 10.1016/j.future.2024.01.002.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Shihao Li, Wei Dai, Yongyan Chen, and Bo Liang. Optimization of high-performance computing job scheduling based on offline reinforcement learning. *Applied Sciences*, 14(23):11220, 2024. doi: 10.3390/app142311220.

Zachary C. Lipton and Jacob Steinhardt. Troubling trends in machine learning scholarship. *arXiv preprint arXiv:1807.03341*, 2018. Presented at ICML 2018: The Debates.

Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets '16)*, pp. 50–56, 2016. doi: 10.1145/3005745.3005750.

Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19)*, pp. 270–288, 2019. doi: 10.1145/3341302.3342080.

Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

Kevin Musgrave, Serge Belongie, and Ser-Nam Lim. A metric learning reality check. In *Computer Vision – ECCV 2020*, pp. 681–699, 2020. doi: 10.1007/978-3-030-58595-2_41.

Mitsuhiko Nakamoto, Yuexiang Zhai, Anikait Singh, Max Sobol Mark, Yi Ma, Chelsea Finn, Aviral Kumar, and Sergey Levine. Cal-ql: Calibrated offline rl pre-training for efficient online fine-tuning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing (SoCC '12)*, pp. 1–13, 2012. doi: 10.1145/2391229.2391236.

Junjie Shi and Kuan-Hsun Chen. Shielded reinforcement learning for fault-tolerant scheduling in real-time systems. *Real-Time Systems*, 61(2):306–310, 2025. doi: 10.1007/s11241-025-09441-z.

Muhammad Tirmazi, Adam Barker, Nan Deng, Md E. Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. Borg: the next generation. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys '20)*, pp. 1–14, 2020. doi: 10.1145/3342195.3387517.

Jesse van Remmerden, Zaharah Bukhsh, and Yingqian Zhang. Generalizing beyond suboptimality: Offline reinforcement learning learns effective scheduling through random solutions. *Transactions on Machine Learning Research*, 2026.

Vanamala Venkataswamy, Jake Grigsby, Andrew Grimshaw, and Yanjun Qi. Launchpad: Learning to schedule using offline and online rl methods. In *Job Scheduling Strategies for Parallel Processing (JSSPP 2024)*, Lecture Notes in Computer Science, pp. 60–83, 2024. doi: 10.1007/978-3-031-74430-3_4.

Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems (EuroSys '15)*, pp. 1–17, 2015. doi: 10.1145/2741948.2741964.

Zerui Wang, Qinghao Hu, Ana Klimovic, Tianwei Zhang, Yonggang Wen, Peng Sun, and Dahua Lin. Semantic-aware scheduling for GPU clusters with large language models. *arXiv preprint arXiv:2510.03334*, 2025.

Di Zhang, Dong Dai, Youbiao He, Forrest Sheng Bao, and Bing Xie. RLScheduler: An automated HPC batch job scheduler using reinforcement learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15, 2020. doi: 10.1109/SC41405.2020.00035.

Yiling Zhang, Yujian Ye, Jianxiong Hu, Heng Hu, Xi Zhang, and Dezhi Xu. Constrained semi-MDP formulation and perception-enhanced safe policy learning for efficient dynamic task scheduling of data centers. *IEEE Transactions on Smart Grid*, 17(2):1209–1224, 2026. doi: 10.1109/tsg.2025.3630667.

A Reproduction Map

Every quantitative claim in the main body is regenerated from the artifacts listed below. Paths are relative to the repository root. All commands assume the project's Python environment.

Table 3: Where each result comes from.

result	command → artifact
Trace characterization, recall bounds (Section 4.1, Section 4.2)	gct_shortlist, gct_replay → gct_characterization.md, gct_replay_summary.json, gct_shortlist_recall.json
Offline comparison (Section 4.2)	gct_offline → gct_offline_metrics.json
Failure modes (Section 4.3)	gct_online –updates 2; read the init logit spread line → gct_online.py
Coverage ladder (Section 5.2)	gct_capacity_audit, gct_key_figure → gct_saturation_audit.txt, gct_key_figure.json
Saturation ladder (Section 5.3)	gct_seed_analysis → gct_seed_report.txt §5
Interference crossover (Section 5.4)	run_interference.sh, gct_interf_analysis → gct_interf_report.txt
Window scan (Section 5.4)	gct_frontier –mode scan → gct_window_scan.json
Capacity boundary (Section 5.5)	gct_capacity_analysis → gct_capacity_report.txt
Variance axes (Section 5.5)	gct_seed_analysis; gct_seedfix_S*.json for the post-fix bands → gct_seed_report.txt, feature_normalisation_compare.txt
Feature-path correction (Section 5.5)	post-fix gct_online runs → results/feature_*.txt (normalisation comparison, escape-tolerance sensitivity, cross-window check)
Cross-trace replication (Section 5.6)	gct2019_replay, ali2018_replay, gct_frontier –trace, multi_trace_audit → cross_trace_audit.txt, gct2019_*, ali2018_*
Like-for-like paired comparison (Section 5.1)	gct_fair_compare → gct_fair_*.json; gct_frontier –families base at $1.0, 0.6, 0.4$ → gct2011_physicalcap.json
Gate ladder and OOD (Section 6)	run_all.sh, aggregate → summary_gated_scratch.csv, all_runs.csv, offline/ood.npz
Figures and tables	figures/gen_fig*.py, figures/gen_tables.py → figures/*.pdf, figures/TABLE_*.tex

B Implementation Details

Offline pipeline. Transitions of $(\text{task features}, \text{candidate-machine features}, \text{action}, \text{reward})$ are exported by replaying the trace, giving $27{,}896$ transitions over a 0.3-hour window. The offline model is a task-conditioned scorer $Q(s, a) = f(\text{task features}, \text{machine features})$, and we evaluate three variants: behaviour cloning, single-step regression without a conservatism term, and CQL. All three are evaluated in the same replay simulator as the heuristics, at capacity 0.22 with a 0.3-hour evaluation window.

Online pipeline. Training uses window h34.5 (0.5 hours, approximately $51{,}500$ decisions per episode) with evaluation on h37, so that train and evaluation windows differ and both are congested. The reward is $r = -\mathbb{1}[\text{delay} > 60\,\text{s}] - 0.2\,\mathbb{1}[\text{utilisation after placement} > 0.9] - \text{stall}$, with $\gamma = 0.999$ and $\lambda = 0.99$, an effective horizon near 91 steps. Fifteen updates at three epochs over minibatches of 512 transitions. λ-returns are standardised; unstandardised, value loss reaches 137–732 and the advantages are effectively noise.

The OOD detector. A Mahalanobis score over the 27-dimensional state, with mean and inverse covariance estimated from offline states; thresholds are set at percentiles of the offline score distribution ($p90$, $p95$, $p99$, $p99.5$). The adaptive gate re-derives the percentile when the recent SLA rate exceeds budget.

What the surviving escape was. That point's gate does not merely filter actions: with a projected headroom of 0.10 it executes a *mixture*: 57.5% of decisions follow the learned policy's ranking and 42.5% fall back to *balanced*. Single-threshold greedy rules cannot express a state-dependent mixture however their thresholds are set, so the 1.16 pp residual is real *for this family*, an artefact of its shape rather than of where its thresholds sit, which is why the criterion is to grant a variable rather than to re-tune. It is not evidence about the learner: granting the family n_{on} removes the escape (Section 5.2).

The escape criterion's knife edge, and the honest counterexample. A weakness in the criterion bounds how such a count may be read. Those counts use the operating points trained before the feature-path correction of Section 5.5; re-running the same four seeds afterwards leaves the conclusion unchanged, but against the fine 138-point family 4 of 20 points now escape and the clause separating them is *throughput*: two of the four match or beat the learner on SLA and fail only by placing 68 fewer tasks, 0.1% of the window. Relaxing to a 0.5% throughput tolerance, the right comparison

when SLA and hotspot are equal, cuts the count to 2 of 20; both remaining margins ($+0.36$ pp and $+0.12$ pp) are an order of magnitude below the ± 2.4 pp seed spread and vanish on three windows the networks were not selected on (0 of 12).

One objective does not separate the families: on the anticipatory scarce-slot loss there is no consistent winner, since *balanced* wins on hotspot but loses on slot loss ($+0.0352$) whereas *greedy1* is better on both (-0.4328 and -0.0424).

Why the production scheduler might look near-random (Section 4). Three explanations are consistent with our data and the event log alone cannot separate them: the scheduler may score *predicted* rather than logged usage; the logged requests are coarse and the true admission constraint may permit overcommitment; or job-level constraints may force spread regardless of load.

What the production scheduler is doing instead (Section 4.2). Two regularities dominate the log: 98.9% of placement decisions extend a job already running, and such jobs already occupy a median of $1{,}782$ distinct machines; and the chosen machine does *not* already host a task of the same job 68.1% of the time.

A reward-scaling defect worth recording. Using the increment of a *global* imbalance statistic as a dense reward does not work at this scale: a single placement changes the standard deviation of occupancy across 12,548 machines by roughly 10^{-6}, lost in the cancellation of two $\mathcal{O}(1)$ statistics in float32, so the measured reward standard deviation printed as 0.00000. Dense rewards must be $\mathcal{O}(1)$ local signals. The reward above is written accordingly.

The feature-path correction, and what it changed (Section 5.5). The seed bands are from re-running the same four seeds after fixing a defect in the learner's feature path: several channels were written as though machine capacity were 1.0 (e.g., utilisation as $u = 1 - free_cpu$ rather than $1 - free_cpu/cap$), while capacity is a calibration dial (at the main result, 0.22). The consequences were not cosmetic: the utilisation channel's dynamic range was 0.22 instead of 1.0, and the channel *named* hotspot actually thresholded at 54.5% utilisation, reporting 0.772 where the metric of the same name reads 0.494. Every capacity-scaled channel now divides by cap, and summary's cap argument is required, so a missed call site raises. The fix was verified *definitionally* (each channel must equal its physical quantity divided by the dial) rather than by the tempting but false invariance check "state and dial scaled together leave features unchanged", which fails because a task's request is data, not part of the dial. After the fix the seed bands move by -0.30 pp and -0.35 pp, well inside the ± 2.7 pp seed spread, and every conclusion in Section 5.2–Section 5.3 stands. One further observation: the per-seed SLA on the *training* window is bimodal at roughly 10% and 28%, and the fix swapped which seed landed where ($9.91/28.88/11.22/26.08\%$ became $25.84/10.97/28.49/11.47\%$) while the mean barely moved (19.02% versus 19.19%), so which seed learns well is decided by the seed rather than by the method at this scale.

C The Two Failure Modes in Detail

The mechanics, in full.

Mode 1: one gradient step per update. The update consolidated the whole rollout's loss into a single scalar by summation and called backward() once, so eight "updates" performed eight stochastic gradient steps, folding all three epochs over $8{,}812$ transitions into each one; minibatch updates at 512 transitions moved a single update from 1 step to roughly 52. *Check:* count the backward() calls per update, not the losses: a rollout accumulated into one scalar is one gradient step.

Mode 2: a policy head degenerate at initialisation. The scorer used a bilinear form $f(s, a) = w^\top (t \odot m(a)) + b(t)$, chosen so that 12,548 machines could be scored cheaply. Instrumenting the shortlist gave logits ranging over 0.3105 to 0.3110 across the 64 candidates, a standard deviation of 10^{-4}. Softmax is therefore uniform, so no action is ever preferred; and the bias $b(t)$ shifts *all* logits equally, a direction softmax is invariant to, so the gradient it receives cannot change the policy. Every update is a no-op, and the three symptoms follow exactly. The fix was a concatenation MLP (with 64 candidates there is no reason to factorise), with the action-independent bias removed and inputs

standardised (their scales had differed by two orders of magnitude, free_cpu in $[0, 0.22]$ against a percentile in $[0, 1]$). *Check:* measure the standard deviation of the logits across candidates before training and again after; if it is negligible, every gradient lies in the softmax-invariant direction. The diagnostics that answer the question are KL, entropy, and logit spread, not the policy loss.

The symptom, and why the obvious explanation is wrong. Across every update the policy entropy stayed pinned at $4.159 = \ln 64$ (uniform over the 64-slot shortlist), the KL divergence at exactly 0.00000, and the policy loss at ± 0.000. Our first diagnosis, that the advantages had collapsed after batch normalisation, was wrong: instrumenting them showed standard deviation 0.962. The inference that motivated it was wrong too, since the policy loss is *expected* to be near zero for a softmax policy with zero-mean-standardised advantages, because $\min(r_t a_t, \mathrm{clip}(r_t) a_t) \approx a_t$ and $\mathbb{E}[a_t] = 0$.

After the fix. With minibatch updates, a concatenation head, and input standardisation, and the same rewards otherwise, the policy learns across two independent runs: entropy falls monotonically, KL becomes nonzero, and the logit spread across candidates grows from 0.061 to 0.863 (Table 7). Nothing about the reward function or the environment changed. Both defects are implementation-level and independent of cluster scheduling, so we expect them to transfer to other settings where a scorer is factorised for cost or a rollout loss is accumulated for convenience.

D The Capacity Sweep in Full

Each capacity is evaluated independently at window h37 with $\kappa = 0$. The rule family composition is held fixed across capacities (six baseline heuristics, five headroom settings, two queue switches, four queue-with-cap variants), so that differences between capacities are not differences in the family. The production scheduler is 3-objectively dominated at capacities 0.15, 0.22, 0.30 and 0.45, and not dominated at 0.12. The learned operating points that survive 2-objective domination at the coarse family are at capacity 0.12 (headroom 0.10, SLA $= 58.26\%$) and capacity 0.22 (headroom 0.05, SLA $= 20.26\%$); both are dominated once the co-runner cap is added to the family, which is the effect Section 5.2 quantifies.

E Training-Seed Detail

Four independent training runs were evaluated under five gate settings each, giving 20 operating points. Per-run SLA rates range from 14.91% to 22.66%; the bare policy without a gate ranges from 14.91% to 19.01% across the four seeds ($16.83 \pm 1.70\%$, $n = 4$, 95% CI of the mean). These are the post-fix runs of Section 5.5; the pre-fix runs gave 13.32–21.72% for the bare policy ($18.54 \pm 3.57\%$), so the correction moved the learner's own objective by less than its seed spread. Against the family extended with the co-runner cap, all 20 points are dominated. This is the training-seed evidence behind Section 5.2 and Section 5.5.

F Additional Figures and Tables

The figures and tables below are referenced from the main body; they are placed here to keep the main text within the venue page budget.

Table 4: Notation. Rates are percentages; the three objectives are minimised except placed.

sym.	meaning	sym.	meaning
SLA	SLA violation rate	n_{on}	co-runner count on a candidate
hot	share of machines over a utilisation threshold	h_r	headroom threshold
placed	tasks placed	$n_{\max}$	co-runner cap
q_{cur}	pending queue length	κ, c_o	interference: $1 + \kappa\, c_o$

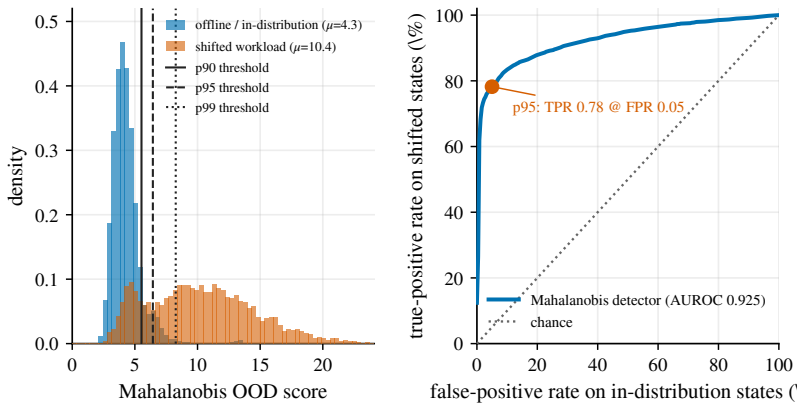

Figure 6: The out-of-distribution detector. **(A)** Mahalanobis score density for offline (in-distribution) and shifted-workload states, with the offline score percentiles used as thresholds. **(B)** ROC of the detector, with the AUROC computed from the same states. Detection is good but not clean: the regimes' utilisation distributions overlap, and most of the separability comes from queue length. This is why a static novelty threshold cannot protect a policy that degrades in distribution.

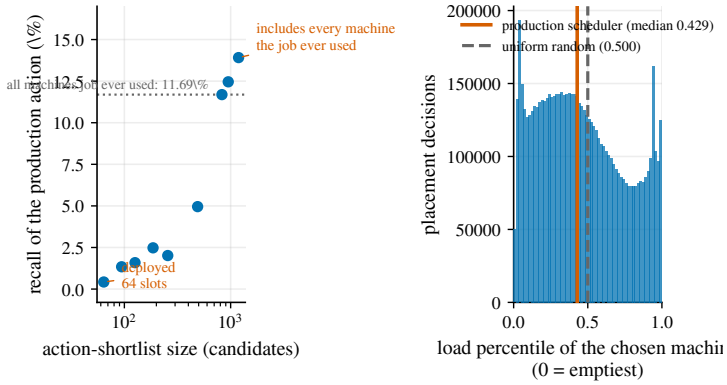

Figure 7: The production scheduler is a poor imitation target (Section 4). **(A)** Recall of the production action against action-shortlist size. The deployed 64-slot shortlist recalls 0.43%; a 1,187-candidate shortlist that includes every machine the job has ever used reaches only 13.92%. **(B)** Load percentile of the machine the production scheduler actually chose (0 = emptiest), over all 7,087,379 placement decisions. The median is 0.429 against 0.500 for uniform-random placement; recomputing with measured task_usage moves it to 0.586.

Table 5: Paired one-step comparison (C8), window h37, capacity 0.22, restricted to the 19,779 of 54,328 sampled decisions (36.4%) where the logged machine was feasible. "Paired" is the difference against the logged action. Source: results/gct_fair_gct2011_c0.22.json (absolute level from REPORT.md A.27.2).

action	hotspot	paired Δhot	remaining free	slot loss
log's own action	0.4337	–	0.1621	0.1261
least_allocated	0.0859	-0.3478 ± 0.0076	0.3837	0.1369
balanced	0.0085	-0.4252 ± 0.0069	0.4504	0.1613
greedy1, h_r=0.10	0.0009	-0.4328 ± 0.0069	0.1100	0.0837
greedy2, h_r=0.10, $n \leq 4$	0.0011	-0.4326 ± 0.0069	0.1156	0.0855
best_fit	1.0000	$+0.5663 \pm 0.0069$	0.0015	0.0694

G Falsification Conditions

H Limitations, and what would change our conclusion

Three traces, one simulator, one calibration dial. The external-validity gap is partly closed: the scheduler-versus-rules comparison replicates on three traces, two operators and three cells or years, through one code path and protocol (Section 5.6). Two gaps remain: the copyability measurement of Section 4 was made only on the main trace, so we cannot say whether production decisions are

Table 6: The same protocol on three traces (C7), one code path. "Dominators" counts rule points no worse than *borg* on all three objectives; a zero marks a saturated cell. Parsed from results/cross_trace_audit.txt.

trace	cap.	*borg* SLA%	*borg* hot%	dominators	best dominator	SLA% / hot%
Google 2011	0.12	54.99	54.03	0	– (saturated)	–
Google 2011	0.15	47.47	54.88	1	*greedy1/hr=0.100*	46.06 / 46.03
Google 2011	0.22	22.05	40.51	1	*greedy1/hr=0.100*	20.40 / 27.42
Google 2011	0.30	0.00	19.22	11	*baseline/balanced*	0.00 / 12.82
Google 2011	0.45	0.00	6.47	11	*baseline/balanced*	0.00 / 3.59
Google 2019	0.20	51.63	27.86	2	*greedy1/hr=0.100*	9.47 / 9.04
Google 2019	0.25	8.18	29.45	12	*queue/pack_if_q>1*	1.95 / 29.40
Google 2019	0.30	0.98	25.87	3	*greedy1/hr=0.100*	0.00 / 6.78
Alibaba 2018	0.25	76.60	89.49	0	– (saturated)	–
Alibaba 2018	0.30	0.00	22.54	17	*baseline/balanced*	0.00 / 12.31

Table 7: Two implementation defects and their signatures, on the 64-candidate shortlist. Source: REPORT.md appendix A.13.

mode	quantity	before	after
one step per update	gradient steps per update	1	~ 52
	transitions folded into it	8,812	512
degenerate head	logit std across 64 candidates	10^{-4}	$0.061 \rightarrow 0.863$
	policy entropy	4.159 (const.)	$4.154 \rightarrow 3.895$
	KL(old $\|$ new)	0.00000	0.002–0.010
	clip fraction	0.000	0.012–0.107
advantage std (never collapsed)		0.962	

Table 8: The protection ladder, $n = 3$ seeds per rung, all rungs the same policy evaluated in the training regime. "Worst" is the largest per-seed catastrophic count out of 50 episodes; the four rungs aggregate to 6.01 ± 2.55 and 1.8 over $n = 12$.

protection	SLA% (n=3)	cat. mean	worst	takeover%
none (bare PPO)	11.28 ± 10.89	14.00	39	–
+ safety layer	6.38 ± 3.25	2.33	4	100.0
+ static OOD fallback	8.13 ± 4.05	2.67	5	88.0
+ full adaptive gate	4.90 ± 1.05	1.00	3	65.4
heuristic (reference)	3.65	0.00	0	0.0

near-unimitable in general; and the new traces carry no learned policies, so what replicates is the comparison, not the pipeline. The simulator is also open: it reserves requested rather than measured resources, has no preemption or migration, and counts an unplaced task as an SLA violation. That is a real failure, but folding throughput into SLA makes our rates incomparable to a deployment measuring violations only among placed tasks (hence the placed column of Table 2). Two further limits: capacity is a calibration dial, not measured hardware (Section 3.1), and Alibaba's SLA is not comparable in absolute terms because that trace has no submission time. The production baseline itself degrades with the dial: beyond roughly physical capacity the replayed log is a log-plus-fallback, which is why Section 5.1 rests the claim on the paired comparison.

The interference model is ours. The co-location cost $1 + \kappa\, c_o$ penalises only the newly placed task and does not retroactively slow running tasks, so Section 5.4 reports a crossover *weight* under a model we chose: our evidence supports only that much state-dependent tension is needed before learning becomes preferable *under this cost model*.

Online scale, and the pipeline we did not re-run. The online stage goes no further than a 64-slot shortlist scorer trained with PPO for 15 rounds, and we fixed the two failure modes of Section 4.3 *after* those runs, so CQL was never re-run on the corrected pipeline, and the validity region of Section 5.5 was swept rather than derived. The feature-path correction is part of the same record: an observation defect that moved the learner's numbers by less than the seed spread.

An operational criterion, and a hedged prior-work claim. The test in Section 5.3 can falsify coverage but cannot exhaust it: we show the family saturated with respect to the variables we tried, not that no further one would move it, and the escape criterion needs its stated throughput tolerance

Table 9: Three variance axes for the headline comparisons; ± is a 95% confidence interval of the mean. Source: the post-fix runs results/gct_seedfix_S*.json.

axis	quantity	mean ± 95% CI	n
window (8, paired)	ΔSLA vs. *borg* (pp)	-10.52 ± 8.72	8
	Δhotspot vs. *borg* (pp)	-5.95 ± 3.26	8
eval seed (1 net)	SLA (pp)	±0.2–0.6	7
training seed	SLA (%)	18.34 ± 1.04	20
	hotspot (%)	38.60 ± 2.99	20
gate rung (bare)	SLA (%)	11.28 ± 10.89	3
gate rung (full)	SLA (%)	4.90 ± 1.05	3

Table 10: The crossover weight α (pp of SLA per unit slowdown) at which the learned family becomes preferable, under the co-location cost $\text{cost} = 1 + \kappa\, c_o$. Source: results/gct_interf_report.txt.

κ	α at crossover (pp SLA / unit slowdown)	payoff at crossover
0	never (up to $\alpha = 10$)	–
0.5	27.5–45	~ 0
1.0	21.5–33.5	+0.139 pp
2.0	3.5–5.5	+0.181 pp

Table 11: Falsification conditions for this paper's claims.

if this appears	then	what it would take
A state variable not expressible via n_{on} or q_{cur} that, once granted to the rule family, leaves the frontier unmoved	The coverage criterion is incomplete and the escape was not fully explained	Add a state-dependent cost to the environment (preemption risk, heterogeneous interference) and re-measure
A non-threshold simple policy (soft penalty, two-window scheduling) that clearly beats the family	The family's *shape* was the limitation, not its features	Enlarge the family's functional form, not its grid, and recount escapes
A copyability ceiling on the other traces materially above the main trace's 13.92% recall	Section 4 is a property of this recording, not of production schedulers	Re-run the recall and load-percentile measurements on 2019 and Alibaba (Section 4 reports them for the main trace only)
A fair design that is insensitive to the capacity dial still finds the rule family far ahead of the logged action	The scope restriction of Section 5.1 falls away and the claim becomes stronger	At physical capacity the paired difference is 0.29–2.02 pp, so this direction favours the paper rather than defeating it
The production scheduler's own choices clearly worse than the rules in *its* world (e.g., Alibaba's 0.00% becoming 20%)	The scope restriction narrows to some traces rather than all	Re-run the paired measurement per trace
Hotspot saturation also at capacity ≥ 0.15 on another trace	The validity region generalises, which strengthens the negative result	Reproduce the capacity sweep

(Section 5.2). Our prior-work position is similarly bounded, since the search was keyword-based. Section G records what would defeat each headline claim.

I Acknowledgements of Scope

We do not claim that reinforcement learning is unsuited to cluster scheduling. We show that on the one trace where the whole pipeline could be run, the recommended offline-to-online pipeline's performance advantage over heuristic rules is explained by a baseline feature-coverage artefact; that the scheduler-versus-rules comparison, and the paired one-step counterfactual, replicate across three production traces under one simulator and one self-authored interference model; that the

pipeline's advantage over the *replayed* production scheduler is a real scheduling-quality difference whose magnitude is set by the capacity dial rather than by the scheduler's quality; and that its safety mechanism is real and quantifiable. Section H lists what would change that reading.