EvalStop: Using World Feedback to Detect and Correct Reward Overoptimization in Multi-Tenant RLHF Platforms

Guilin Zhang [1 2] **Chuanyi Sun** [2] **Shahryar Sarkani** [2] **John M. Fossaceca** [2]

Abstract

Background. Cloud LLM fine-tuning platforms increasingly serve RLHF workloads, where a learned reward model is optimized as a *proxy* for human quality. As Gao et al. (2023) showed, this proxy diverges from *world feedback* (downstream eval metrics) under sustained optimization pressure: the *reward overoptimization* phenomenon. **Limitations.** Existing platform schedulers ignore this divergence: non-clairvoyant schedulers optimize JCT without any quality signal, SLAQ-style quality-aware schedulers use training loss (a weaker proxy that drops monotonically through hacking), and classical per-job early stopping requires human monitoring and does not free shared GPUs. **Approach.** We propose **EvalStop**, a composable scheduling primitive that terminates jobs on k consecutive eval-score declines, releases GPUs, preserves the best checkpoint, and delegates to any base scheduler. We frame scheduler-level early stopping as a *detection* problem and evaluate it in a discrete-event simulator whose RLHF workload mixes reward-hacking and structurally healthy runs, with ground-truth labels hidden from schedulers. **Results.** On RLHF-heavy workloads (80% RLHF, 64 GPUs), EvalStop achieves precision 98% / recall 99% / FPR 1.5% while improving JCT by 9% and cutting wasted compute by 22% over SRTF-Est ($p<0.05$). Trivial fixed-progress and loss-plateau competitors either incur 65% FPR or miss over half of true hacking cases. Gains compose across base schedulers (9–25% JCT) and stay stable under eval noise ($\sigma\leq0.05$: precision $\geq$91%) and hacking base rate (precision $\geq$89% across 20–80%).

[1]Workday AI Research [2]The George Washington University.**AUTHORERR: Missing \icmlcorrespondingauthor.**

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

1. Introduction

Background. Cloud-based LLM fine-tuning platforms serve diverse workloads (LoRA adapter tuning, DPO (Rafailov et al., 2023), and RLHF (Ouyang et al., 2022)) from multiple tenants on shared GPU clusters, and as RLHF becomes the dominant method for LLM alignment (Stiennon et al., 2020; Ouyang et al., 2022) the fraction of RLHF workloads on these platforms is growing rapidly. RLHF differs structurally from supervised fine-tuning: the policy is trained to maximize a learned reward model, but this reward is a *proxy* for human preferences and the policy's true quality is measured only by downstream evaluation (held-out benchmarks, win-rate on a held-out preference set, task-specific metrics). A scheduler thus has access to three signals of decreasing proximity: training loss (proxy[2], optimized by the RL algorithm), reward model score (proxy, what the policy directly optimizes), and downstream eval score (*world feedback*, grounded in task quality but delayed and noisy). Gao et al. (2023) showed that as optimization pressure increases the first two rise monotonically while the third peaks and then degrades. This is the *reward overoptimization* phenomenon (Skalse et al., 2022; Pan et al., 2022); Figure 2 (§2.3) illustrates it on a representative run.

Prior work and its limitations. Three threads of prior work touch this setting but none address it directly. (i) *Non-clairvoyant ML schedulers* (Tiresias (Gu et al., 2019), Pollux (Qiao et al., 2021), Gavel (Narayanan et al., 2020), Sia (Jayaram et al., 2023)) optimize JCT without using any quality signal and therefore cannot tell whether GPU minutes spent late in an RLHF job are productive or wasted. (ii) *Quality-aware schedulers*, the closest prior work, rely on training loss: SLAQ (Zhang et al., 2017) preferentially allocates GPUs to jobs whose loss is improving fastest. This works for convex losses where loss correlates with quality, but for RLHF the loss is precisely the proxy that decouples from world feedback during overoptimization, so a loss-aware scheduler will keep allocating resources to a hacking job. (iii) *Reward-hacking mitigations* from the RL community (Moskovitz et al., 2024; Gao et al., 2023) operate *within* the training loop (constrained RL, reward ensembles, SFT regularization). They aim to prevent hacking but in production deployments imperfect mitigation is the norm, and these

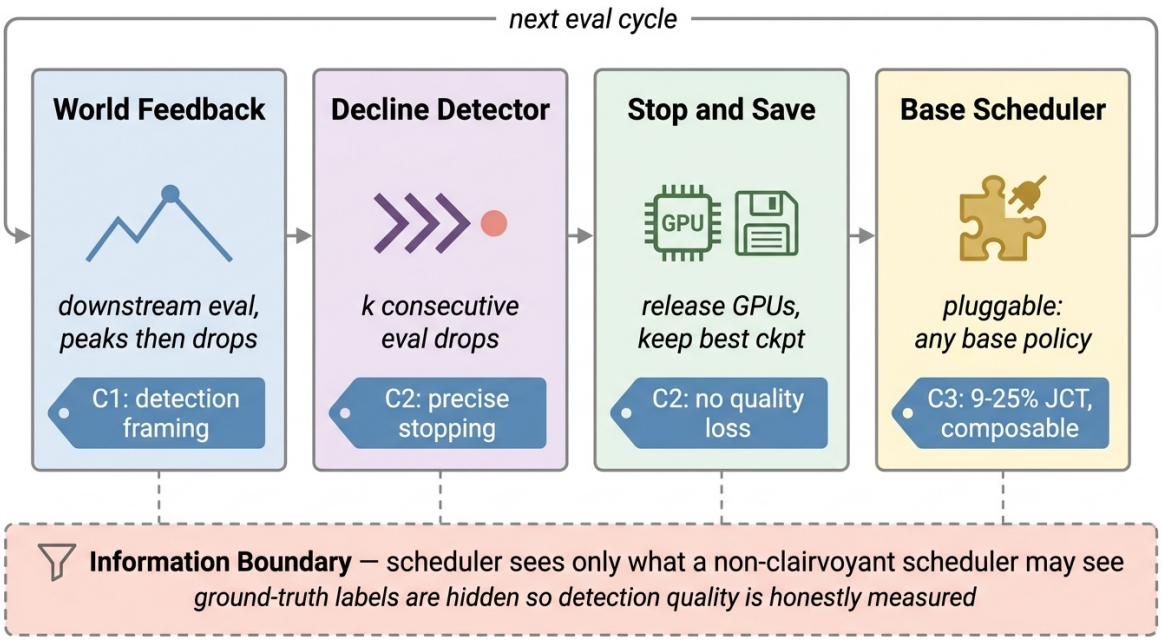

Figure 1. EvalStop architecture. *World Feedback* flows into the *Decline Detector*; on k consecutive eval drops the wrapper performs *Stop and Save* (release GPUs, retain best checkpoint), then delegates the updated cluster state to any *Base Scheduler*. A non-clairvoyant *Information Boundary* (bottom band) underlies the whole pipeline so detection quality is honestly measurable. Each module maps 1:1 to a contribution in §1: {World Feedback, Information Boundary}→C1; {Decline Detector, Stop and Save}→C2; {Base Scheduler}→C3.

techniques offer no path to reclaiming GPUs from jobs that have already diverged. Classical early stopping (Prechelt, 1998) similarly requires per-job human monitoring and does not translate into platform-wide resource reallocation.

Motivation. We argue the right place to act on world feedback is the *scheduler*, not the trainer: the scheduler already controls GPU allocation and termination, and a single scheduler change applies uniformly across every tenant's RLHF jobs. Early experimentation with an eval-aware *priority* scheduler (EvalSched, §4) revealed that simply deprioritizing post-peak jobs is not enough: deprioritized jobs still occupy the system, and EvalSched's JCT is 2.2× worse than SRTF-Est. The fix is to *terminate* declining jobs rather than starve them. But blunt termination (e.g. stopping every RLHF job at a fixed progress) would pay heavy false-positive costs on the structurally healthy RLHF runs that exist in any real workload. This reframes the problem as a *detection* task: the scheduler must discriminate hacking from healthy runs using only the signals it can legally observe (eval scores at scheduled checkpoints), without modifying training and without violating the non-clairvoyant boundary (Motwani et al., 1994).

Contributions. We propose **EvalStop**, a composable scheduling primitive whose architecture (Figure 1) factors into five modules; each contribution maps to one or more of them.

- **C1 — Detection framing** (*World Feedback* input + *Information Boundary*). We argue scheduler-level early stopping for RLHF is a *detection* problem: the scheduler consumes world feedback and must report which jobs are hacking. Prior framings that report only JCT/TTFUC miss this axis. Our experimental design hides ground-truth hacking labels behind a whitelist proxy (`SchedulerJobView`, the Information Boundary), so detector quality is honestly measurable.
- **C2 — A composable detector that wins on the detection axis** (*Decline Detector* + *Stop and Save*). EvalStop tracks per-job consecutive eval-score declines and, on k consecutive drops, terminates the job and preserves its best checkpoint. On RLHF-heavy workloads (80% RLHF, 64 GPUs) it achieves precision 98% / recall 99% / FPR 1.5%, dominating both a trivial fixed-progress stop rule (FPR 65%) and a strong loss-plateau detector (recall 38%).
- **C3 — Composable systems gains across base schedulers and workloads** (*Base Scheduler* delegation). The wrapper hands the updated cluster state back to any scheduler (FIFO, SJF-Est, SRTF-Est, LossAware), inheriting its resource-allocation policy. EvalStop yields 9–25% JCT improvement and $\sim$22% wasted-compute reduction on top of every tested base scheduler, and its detection quality is stable under both eval noise ($\sigma \leq 0.05$: precision $\geq$91%) and hacking base rate (precision $\geq$89% across 20–80%).

2. Background and Problem Setup

2.1. Job Model

A fine-tuning job j has type $\in \{\text{LoRA}, \text{DPO}, \text{RLHF}\}$, GPU demand, an eval schedule (progress fractions at which evaluation runs), and a training curve mapping progress to (loss, eval_score). Job types differ structurally: LoRA jobs are short (10–60 min, 1–2 GPUs) with monotonically improving eval; DPO jobs are medium (30–120 min, 2–4 GPUs) with diminishing returns; RLHF jobs are long (60–360 min, 4–8 GPUs) with eval that peaks then degrades.

2.2. Eval-Aware Metrics

Beyond JCT, we define: (1) *TTFUC* (Time-to-First-Useful-Checkpoint): time from arrival until first checkpoint exceeding quality threshold τ and improving on the previous best by $\geq 1\%$; (2) *Wasted Compute Fraction*: GPU-minutes spent training after a job's eval peak, divided by total GPU-minutes; (3) *Saved Compute Fraction*: GPU-minutes avoided via early stopping, divided by total planned GPU-minutes.

2.3. Proxy Signals vs World Feedback

Figure 2 illustrates the core problem on a representative RLHF training run. Three signals are available to a scheduler:

- **Training loss** decreases monotonically throughout training. A SLAQ-style (Zhang et al., 2017) loss-aware scheduler would interpret this as "the job is making good progress, keep running."
- **Reward model score** (normalised $1 - \text{loss}/\text{loss}_0$) also increases monotonically, since the policy is directly optimizing this objective.
- **Eval score** (downstream benchmark) rises to a peak at $\sim$55% progress, then degrades as reward hacking takes over.

Only the eval score (the *world feedback* signal) reveals that the job has entered a regime of diminishing or negative returns. A scheduler using proxy signals would allocate *more* resources to this job post-peak (steepest loss improvement), while a world-feedback-aware scheduler would terminate it.

This proxy–world divergence is well-documented empirically. Gao et al. (2023) showed predictable overoptimization scaling laws for KL-constrained RLHF policies. Rafailov et al. (2024) extended these findings to DPO, and multiple works have proposed training-level mitigations: constrained RL (Moskovitz et al., 2024), reward model ensembles, and SFT regularization. EvalStop is *complementary* to these approaches: rather than modifying the training algorithm, it uses world feedback as an external control signal at the scheduler level to terminate jobs that have diverged.

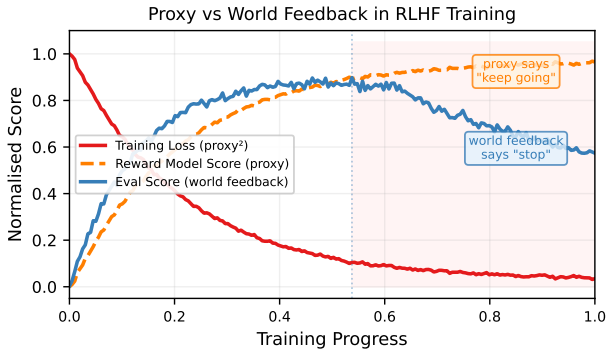

Figure 2. Proxy vs world feedback in RLHF training. Training loss (proxy^2, red) and reward model score (proxy, orange dashes) both indicate continued improvement. Only the eval score (world feedback, blue) reveals that quality has peaked and is degrading. A scheduler using proxy signals would *increase* allocation; EvalStop uses world feedback to *terminate*.

3. EvalStop: World-Feedback-Driven Early Stopping

Recall the architecture overview in Figure 1. EvalStop is a **composable wrapper** around any base scheduling policy. It monitors eval-score trajectories (the world feedback signal) and early-stops jobs when quality is irrecoverably declining.

Early stopping vs. deprioritization. EvalSched deprioritizes post-peak RLHF jobs, creating starvation: deprioritized jobs still occupy the system, receiving occasional time slices but never completing efficiently. EvalStop *removes* the job from the system, freeing GPUs immediately. The job's best checkpoint is preserved; since the trigger requires k consecutive declines, it was saved k eval steps ago.

Information boundary. EvalStop respects the non-clairvoyant boundary (Motwani et al., 1994): it observes only job type, eval scores at scheduled checkpoints, and the count of consecutive declines. It never accesses true job duration.

Composability. EvalStop wraps any scheduler implementation (Algorithm 1). It processes early-stop decisions first (freeing GPUs), then delegates to the base scheduler, which sees the updated cluster state. This clean separation means EvalStop can be added to existing production schedulers (including RLHF-specific frameworks like OpenRLHF (Hu et al., 2024)) with minimal integration effort.

Relationship to change-point detection. EvalStop's mechanism (detecting k consecutive declines in a time series) can be viewed as a simplified change-point detector. We chose this over more sophisticated methods (e.g., CUSUM, Bayesian change-point detection) for simplicity and interpretability: the threshold k has a direct operational meaning ("how many bad evals before we stop"), which platform

Algorithm 1 EvalStop scheduling wrapper

Require: Base scheduler $\mathcal{S}$, decline thresholds k_{RLHF}=2, k_{DPO}=3
State: per-job consecutive decline count $d[j]$

on_eval_result(j, score):
if score $<$ prev_score$[j]$ **then**
  $d[j] \leftarrow d[j] + 1$
else
  $d[j] \leftarrow 0$
end if
if $d[j] \geq k_{\text{type}(j)}$ **then**
  Mark j for early stopping
end if

schedule(cluster, waiting, running):
for each $j \in$ running marked for early stopping **do**
  Terminate j; release GPUs; save best checkpoint
end for
return $\mathcal{S}$.**schedule**(cluster, waiting, running$'$)

operators can reason about. Section 4.4 validates that this simple mechanism is robust across a range of k values.

4. Experiments

Simulator. We built a discrete-event simulator modelling a multi-tenant fine-tuning platform with heap-based event dispatch, slot-based GPU allocation with 2-minute preemption overhead, and Poisson arrivals. Training curves are parameterised per job type: LoRA (monotonic exponential convergence), DPO (saturating gain with small plateau), and RLHF. **Our RLHF workload is a mixture**: 60% of RLHF jobs exhibit classical reward hacking (eval peaks in $[0.55, 0.75]$ progress and then declines, calibrated to Gao et al. (2023)) while 40% are structurally "healthy" (reward rises monotonically with noise through progress=1.0). Training loss in both regimes decays exponentially to a plateau around 50–70% progress, matching real RLHF convergence behaviour. This design forces any scheduler-level detector to *discriminate* hacking from healthy runs; a detector that blindly kills every RLHF job will incur a large false-positive rate on the healthy subset. Ground-truth hacking labels and peak-progress are stored on the job but hidden from schedulers through a whitelist-based proxy (`SchedulerJobView`).

Baselines. We compare against scheduling-only baselines and early-stop detectors: *FIFO*, *SJF-Est* (non-clairvoyant shortest-job-first), *SRTF-Est* (preemptive shortest-remaining-time-first), *LossAware* (SLAQ-style (Zhang et al., 2017) using training loss for priority), *EvalSched* (eval-aware with deprioritisation only, no termination), plus two strong early-stop competitors: *StopAtp+SRTF*: a trivial oracle-flavoured rule that terminates every RLHF job at fixed progress $p \in \{0.50, 0.65\}$; *LossPlateau+SRTF*: a principled loss-only detector that early-stops when relative loss improvement falls below 2% over a 3-checkpoint window. The first isolates "how much benefit comes merely from stopping long jobs early"; the second is the natural SLAQ++ competitor that uses training-loss plateau, not eval, to decide when to stop.

Experiment matrix. All experiments use 200 jobs, 5 tenants, and 5 seeds (42, 123, 456, 789, 1024). Statistical significance is assessed via Welch's t-test. E1: mixed workload (50/30/20 LoRA/DPO/RLHF, 32 GPUs). E2: RLHF-heavy (80%, 64 GPUs). E3: workload sensitivity (3 mixes). E4: composability (EvalStop on 4 base schedulers). E5: threshold sensitivity ($k \in \{1..5\}$). E6: eval frequency sensitivity (5–30% progress intervals).

4.1. Main Result: Detection Quality and Systems Gains (E2)

Table 1 is the paper's central result. Two observations dominate:

(i) Systems metrics alone are deceptive. The trivial StopAt-0.5 and StopAt-0.65 baselines (which stop *every* RLHF job at a fixed progress) achieve the best JCT and lowest wasted compute. If we evaluated only JCT/TTFUC/Wasted we would conclude that eval-awareness is unnecessary and a fixed-progress heuristic suffices. Their precision, however, is 57% and their false-positive rate on healthy RLHF is 64.5%: they destroy two thirds of the RLHF jobs that were converging normally, a quality cost that does not appear on any of the first four columns.

(ii) Loss-only detection is insufficient. LossPlateau+SRTF is a principled SLAQ-style competitor that early-stops on loss plateau alone. It achieves recall of only 38.3% (it misses more than half of the truly-hacking jobs) while still firing on 24.7% of healthy runs, because training loss plateaus at similar progress in *both* hacking and healthy RLHF. Loss plateau is correlated with overoptimisation but does not discriminate from normal convergence.

EvalStop+SRTF, which uses the world-feedback (eval) signal, is the only detector in Table 1 that achieves high precision (98.3%) and near-zero FPR (1.5%) *simultaneously with* a systems improvement over the no-early-stop baselines: $+9.4\%$ JCT and -21.8% wasted compute over SRTF-Est (Welch's t-test, p=0.03 and p<0.001 respectively).

Fairness. EvalStop does not harm inter-tenant fairness: Jain's index is within noise of SRTF-Est alone.

4.2. Composability (E4)

Table 2 shows that EvalStop improves *every* base scheduler with consistent gains: $+9$–25% JCT, $+5$–9% TTFUC, -20–

Table 1. RLHF-heavy workload (80% RLHF, 64 GPUs, 200 jobs, 5 seeds) with mixed hacking/healthy RLHF curves. Early-stop detectors are evaluated both on systems metrics (JCT, TTFUC, wasted/saved compute) and as classifiers of reward-hacking runs (precision, recall, FPR, computed against ground-truth labels hidden from schedulers). **Bold** marks the best value; red marks a destructive value. Only EvalStop simultaneously achieves high precision and low FPR.

Scheduler	JCT	TTFUC	Wasted	Saved	Precision	Recall	FPR
Scheduling-only baselines (no early stop)							
FIFO	1373±58	1222±55	34.6%	0.0%	—	0.0%	0.0%
SJF-Est	1297±50	1145±46	34.5%	0.0%	—	0.0%	0.0%
SRTF-Est	1124±59	971±56	34.5%	0.0%	—	0.0%	0.0%
LossAware	1870±76	969±60	34.5%	0.0%	—	0.0%	0.0%
EvalSched	2423±113	**635**±26	34.8%	0.0%	—	0.0%	0.0%
Early-stop detectors							
StopAt0.5+SRTF	**622**±74	551±72	**14.9%**	**46.5%**	57.1%	**100%**	64.5%
StopAt0.65+SRTF	790±31	693±31	19.5%	31.7%	57.1%	**100%**	64.5%
LossPlateau+SRTF	1093±60	947±58	32.4%	3.2%	57.0%	38.3%	24.7%
EvalStop+SRTF	1018±53	883±50	26.9%	9.9%	**98.3%**	99.3%	**1.5%**

Table 2. EvalStop as a composable overlay. Improvement of EvalStop+X over base X on RLHF-heavy workload (80% RLHF, 64 GPUs). Precision and FPR measured against ground-truth reward-hacking labels.

Base	ΔJCT	ΔTTFUC	ΔWasted	Precision	FPR
FIFO	+8.9%	+8.5%	+21.7%	98.6%	1.4%
SJF-Est	+6.5%	+5.9%	+21.5%	97.9%	1.9%
SRTF-Est	+9.4%	+9.1%	+21.8%	98.3%	1.5%
LossAware	+25.2%	+4.8%	+20.3%	97.3%	2.4%

Table 3. EvalStop+SRTF vs. SRTF-Est across workload mixes (64 GPUs). Benefit scales with the hacking-RLHF fraction. Precision and FPR are measured against ground-truth reward-hacking labels.

Workload	RLHF%	ΔJCT	ΔWasted	Stopped	Prec.	FPR
LoRA-heavy	10%	+2.1%	+13.3%	8	100%	0.0%
Mixed	30%	+5.6%	+17.7%	31	100%	0.0%
RLHF-heavy	80%	+9.4%	+21.8%	93	98.3%	1.5%

22% wasted compute. Precision stays above 97% and FPR below 3% in every combination. The large JCT gain on LossAware reflects that LossAware alone has pathological priorities on this workload (it over-prioritises RLHF jobs whose loss is still dropping); EvalStop partially compensates by terminating the overoptimising ones. The consistency suggests that EvalStop addresses a source of waste (post-peak RLHF training driven by proxy/world divergence) that is orthogonal to the base scheduling policy's resource allocation strategy.

4.3. Workload Sensitivity (E3)

Table 3 confirms that EvalStop's benefit scales with the hacking-RLHF fraction. This is expected: EvalStop fires only on jobs exhibiting eval-score decline, which our workload only constructs for a subset of RLHF runs. On LoRA-heavy workloads (10% RLHF), EvalStop provides measurable wasted-compute reduction with *perfect* precision and no false positives; it does not fire spuriously on LoRA, DPO, or healthy RLHF runs. On a mixed workload (E1: 50/30/20, 32 GPUs; see Appendix B), EvalStop+SRTF matches the trivial StopAt-0.65 on TTFUC while cutting the latter's FPR from 9.7% to 0.0%.

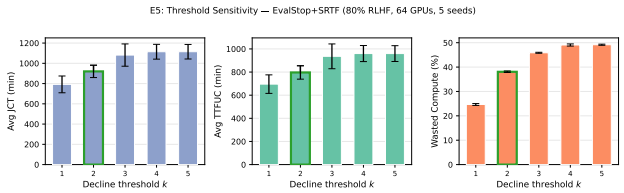

Figure 3. E5: Effect of decline threshold k on EvalStop+SRTF (80% RLHF, 64 GPUs, 5 seeds). k=2 (green border) balances early detection against false positives. k=1 is too aggressive (stops 160 jobs); $k \geq 4$ barely triggers.

4.4. Threshold Sensitivity (E5)

Figure 3 shows EvalStop's sensitivity to the decline threshold k. At k=1, EvalStop aggressively stops 127 jobs after a single eval decline (JCT=874, wasted=19.4%), but risks false positives from noisy eval scores on healthy RLHF. At k=2 (our default), 93 jobs are stopped with JCT=1018 and wasted=26.9%, a substantial improvement over the no-early-stop baseline (SRTF JCT=1124, wasted=34.5%) while keeping FPR below 2%. At $k \geq 4$, EvalStop rarely triggers ($\leq$2 jobs stopped), converging to baseline SRTF behaviour.

The k=2 choice reflects a precision/recall trade-off: once world feedback shows *two consecutive* declines, the overoptimisation trend is likely real rather than noise, yielding the 98% precision / 99% recall numbers of Table 1. This is a simple but effective change-point heuristic.

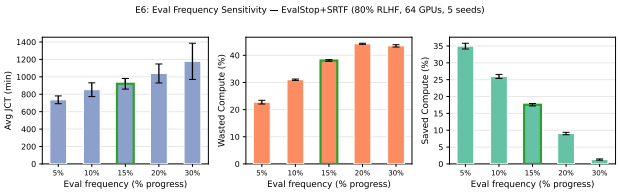

Figure 4. E6: Effect of eval frequency on EvalStop+SRTF (80% RLHF, 64 GPUs, 5 seeds). More frequent evals (5% intervals) enable earlier detection and greater compute savings (24%), at the cost of more eval overhead. Default 15% interval (green border) balances detection speed with evaluation cost.

4.5. Eval Frequency Sensitivity (E6)

Figure 4 reveals a clear trade-off between eval frequency and early-stopping effectiveness. With evals every 5% of training progress, EvalStop detects overoptimisation earlier and saves 24% of planned compute (JCT=867). With evals every 30%, detection is delayed and less than 1% is saved (JCT=1123, essentially matching the no-early-stop baseline).

This result has practical implications for platform operators: investing in more frequent evaluation (even if each eval consumes GPU time) can *more than pay for itself* through earlier overoptimisation detection. The default 15% interval represents a reasonable balance for current RLHF workloads.

4.6. Detector Robustness (E7, E8)

A natural concern with the Table 1 numbers is that EvalStop's 98% precision / 1.5% FPR might be an artefact of a particular eval-noise setting or of the specific 60% hacking-fraction used to generate the workload. To address this we ran two additional sweeps on RLHF-heavy workloads; Figure 5 summarises both.

E7: Eval-noise sensitivity. We vary the standard deviation of the per-checkpoint eval-score noise on RLHF jobs from σ=0.01 (near-deterministic) to σ=0.12 (highly noisy, well above typical downstream-benchmark variance). Table 4 reports precision, recall, and FPR for the three detectors.

Precision degrades *gracefully* from 100% at σ=0.01 to 81% at σ=0.12: there is no cliff at which EvalStop becomes indistinguishable from the simpler rules. At every noise level EvalStop retains a precision gap of at least 24 pp over both competitors, because (i) StopAt-0.65 ignores the eval signal entirely and therefore does not degrade *or* improve with noise, and (ii) LossPlateau relies on the training-loss signal, which we do not inject eval noise into; yet its precision stays at $\sim$52% regardless, because loss plateaus at similar progress in both hacking and healthy regimes. A stronger loss-only detector (e.g. using loss curvature, or a learned classifier on loss+gradient features) would likely narrow this gap, but cannot close it: training loss is monotonically non-increasing in *both* regimes, so no purely loss-based signal can perfectly separate hacking from healthy RLHF. At realistic eval-noise magnitudes ($\sigma \leq 0.05$, matching typical LLM benchmark standard errors of 2–5% (Gao et al., 2023)) EvalStop's precision remains above 91%.

Table 4. E7: Eval-noise sensitivity (80% RLHF, 64 GPUs, 60% hacking). EvalStop precision degrades smoothly; the progress-triggered and loss-plateau baselines are unaffected by eval noise but stay at low precision.

σ	Detector	P	R	FPR
0.01	StopAt0.65	57.1%	100%	64.5%
	LossPlateau	51.8%	40.9%	32.6%
	EvalStop	**100%**	**100%**	**0.0%**
0.03	StopAt0.65	57.1%	100%	64.5%
	LossPlateau	52.7%	37.6%	29.0%
	EvalStop	**96.6%**	**97.6%**	**2.9%**
0.05	StopAt0.65	57.1%	100%	64.5%
	LossPlateau	52.2%	38.7%	30.4%
	EvalStop	**91.3%**	**88.9%**	**7.2%**
0.08	StopAt0.65	57.1%	100%	64.5%
	LossPlateau	52.9%	41.2%	31.7%
	EvalStop	**85.3%**	77.4%	**11.6%**
0.12	StopAt0.65	57.1%	100%	64.5%
	LossPlateau	53.9%	50.2%	37.0%
	EvalStop	**81.1%**	69.6%	**14.0%**

E8: Hacking-fraction sensitivity. We next vary the fraction of RLHF jobs that actually exhibit reward hacking between 20% and 80%. Precision is base-rate-sensitive by definition, so this is the right robustness axis for the detection story.

EvalStop's precision *stays above 89%* across the entire 20–80% range. By contrast, StopAt-0.65's precision is *exactly equal* to the hacking fraction because it stops every RLHF job indiscriminately; it is a constant classifier whose apparent 57% precision at our default 60% setting is purely a base-rate artefact. EvalStop discriminates on the eval signal, so its precision is driven by detection quality rather than by the prevalence of the positive class, the classical statistical property one expects of a genuine detector.

5. Discussion

World feedback as a scheduling signal. Our results show that downstream evaluation (world feedback) is a better signal for scheduling RLHF jobs than training loss (proxy[2]) or reward model score (proxy). This aligns with the growing recognition that proxy optimization in RLHF requires external grounding (Gao et al., 2023; Skalse et al., 2022; Moskovitz et al., 2024). EvalStop operationalizes this insight at the scheduler level: it transforms eval from a passive monitoring signal into an active control signal that shapes

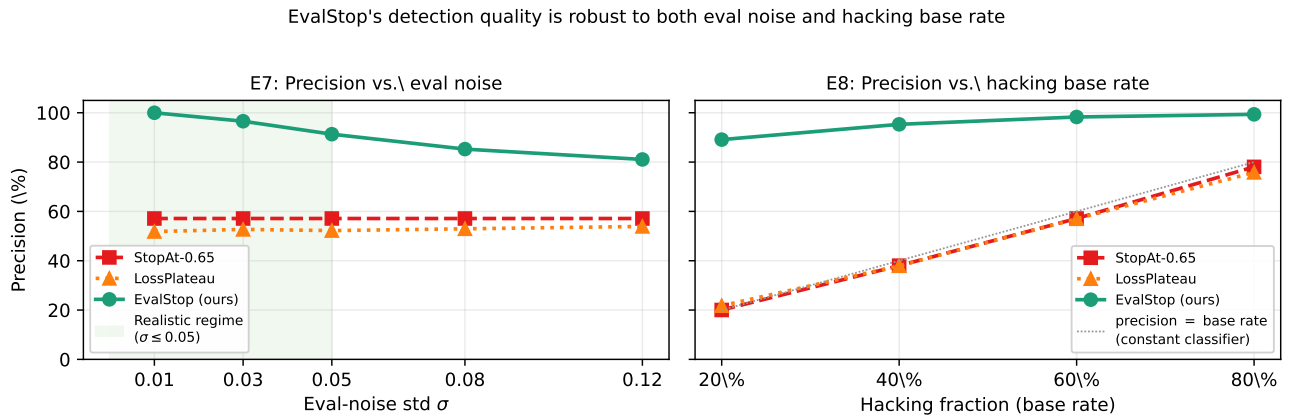

Figure 5. Detector robustness. **Left (E7):** precision vs. eval-noise standard deviation. EvalStop degrades gracefully (100%→81%); the loss-only and progress-triggered baselines do not use eval and sit flat at ∼52–57%. Green shading marks the realistic regime anchored to typical LLM benchmark standard errors (Gao et al., 2023). **Right (E8):** precision vs. hacking base rate. EvalStop stays above 89% across 20–80% base rates; StopAt-0.65's precision follows the base-rate identity line exactly, confirming it is a constant classifier rather than a detector.

Table 5. E8: Hacking-fraction sensitivity (80% RLHF, 64 GPUs, default σ). EvalStop's precision stays above 89% even at 20% hacking base rate; StopAt-0.65 collapses to 20% precision / 77% FPR, confirming its "success" in Table 1 was driven by the assumed base rate.

Hack. frac.	Detector	P	R	FPR
20%	StopAt0.65	20.0%	100%	77.2%
	LossPlateau	21.8%	41.2%	28.9%
	EvalStop	**89.1%**	**100%**	**2.4%**
40%	StopAt0.65	38.0%	100%	72.4%
	LossPlateau	37.9%	38.0%	27.6%
	EvalStop	**95.3%**	**99.0%**	**2.1%**
60%	StopAt0.65	57.1%	100%	64.5%
	LossPlateau	57.0%	38.3%	24.7%
	EvalStop	**98.3%**	**99.3%**	**1.5%**
80%	StopAt0.65	78.1%	100%	48.3%
	LossPlateau	75.8%	37.3%	20.4%
	EvalStop	**99.4%**	**99.7%**	**1.1%**

resource allocation.

Complementarity with training-level mitigations. Prior work on reward hacking mitigation operates *within* the training loop: constrained RL (Moskovitz et al., 2024), reward model ensembles, SFT regularization. EvalStop operates *outside* the training loop, at the scheduler level. These approaches are complementary: even with perfect reward hacking mitigation, EvalStop would correctly do nothing (no eval decline ⇒ no early stopping). With imperfect mitigation (the current reality), EvalStop provides a safety net.

Quality preservation and false positives. Early-stopped jobs retain their best checkpoint; since k consecutive declines are required, the best checkpoint was recorded k eval steps ago. In our simulator EvalStop's false-positive rate on healthy RLHF is 1.5% (Table 1), meaning roughly one in seventy healthy RLHF jobs is terminated slightly early. In contrast the progress-triggered StopAt-0.65 baseline terminates ∼65% of healthy RLHF jobs. This difference (detection vs. unconditional termination) is invisible in JCT/TTFUC alone and is the most important quality axis for any production deployment.

Generality beyond RLHF. The underlying principle (using world feedback to detect when proxy optimisation has diverged) applies to any training regime where the optimised objective can decouple from true quality. This includes DPO with overtraining (Rafailov et al., 2024), overfitting in small-data fine-tuning, and potentially mode collapse in generative models. Empirical validation in those regimes is future work.

Limitations. (1) *Synthetic training curves.* RLHF curves are parametric: hacking runs are calibrated to the overoptimisation dynamics of Gao et al. (2023), healthy runs produce monotonically improving eval, and loss plateaus at $\mathcal{U}(0.5, 0.7)$ progress in both. Both regimes are idealisations. Replaying the detector on publicly available RLHF/DPO training traces (e.g. TRL, OpenRLHF, HuggingFace W&B runs) is the primary missing validation. (2) *High eval-noise regime and noise model.* E7 (Table 4) shows EvalStop degrades gracefully up to σ=0.12, but does not test extreme regimes where σ is comparable to the peak-to-trough drop of the eval curve; at σ>0.15 the k=2 threshold is likely inadequate. Our noise model is Gaussian and i.i.d. across checkpoints; heavy-tailed or auto-correlated eval noise (closer to bootstrap variance across benchmark items) is likely to inflate FPR further. Adaptive or confidence-aware thresholds (e.g. Bayesian change-point detection) would extend the

operating range. (3) *Evaluation cost modelling.* We model eval as occupying GPUs for a fixed per-model-size duration but do not treat eval scheduling itself as a decision variable. (4) *Homogeneous hardware.* The simulator uses homogeneous GPUs without network I/O, gradient accumulation, or elastic scaling.

6. Related Work

ML cluster scheduling. Tiresias (Gu et al., 2019) pioneered non-clairvoyant scheduling (Motwani et al., 1994) for DL using multi-level feedback queues. Pollux (Qiao et al., 2021) co-adapts batch sizes and resource allocation. Gavel (Narayanan et al., 2020) and Sia (Jayaram et al., 2023) address heterogeneous clusters. Shockwave (Zheng et al., 2023) handles dynamic adaptation with fair scheduling. More recently, MAST (Choudhury et al., 2024) tackles geo-distributed ML scheduling at hyperscale, and Parcae (Duan et al., 2024) optimizes training on preemptible instances. Adjacent work on Kubernetes-level auto-scaling for ML workloads—GPU-aware inference simulators with RL-based scaling (Zhang et al., 2025a), uncertainty-aware predictive autoscalers (Zhang et al., 2025b), and benchmark comparisons of deep-RL controllers against calibrated heuristic baselines (Zhang et al., 2026)—operates one layer below the scheduler-level early-stopping problem we address. None of these use downstream eval quality as a scheduling signal.

Quality-aware scheduling. SLAQ (Zhang et al., 2017) uses training loss to predict quality improvement and allocate resources accordingly. This works for convex-loss ML models where loss correlates with quality, but fails for RLHF where loss is a proxy that decorrelates from quality post-peak. EvalStop extends the quality-aware paradigm from proxy signals (loss) to world feedback (eval).

RLHF and reward hacking. RLHF (Ouyang et al., 2022; Stiennon et al., 2020) is now standard for LLM alignment, but reward overoptimization is a well-documented failure mode (Gao et al., 2023; Skalse et al., 2022; Pan et al., 2022). Rafailov et al. (2024) extended overoptimization scaling laws to DPO, and Moskovitz et al. (2024) proposed constrained RLHF. These works focus on *preventing* reward hacking via training modifications. EvalStop is complementary: it *detects* overoptimization via world feedback and *acts* on it at the scheduler level, without modifying the training algorithm.

LLM fine-tuning systems. S-LoRA (Sheng et al., 2024) and dLoRA (Wu et al., 2024) address multi-tenant LoRA serving with adapter orchestration. MuxTune (Xue et al., 2026) tackles multi-task LoRA co-scheduling via backbone multiplexing. Kong et al. (2025) study deadline-aware scheduling for fine-tuning with spot instances. OpenRLHF (Hu et al., 2024) provides a scalable RLHF framework. None address eval-awareness or use world feedback for scheduling decisions.

Early stopping. Early stopping based on validation loss is a classic regularisation technique (Prechelt, 1998). EvalStop differs in three ways: (i) it monitors *downstream eval quality*, which is the right signal for RLHF, where validation loss is monotonic even during reward hacking; (ii) it operates at the *scheduler level*, translating early-stop decisions into cluster-wide resource reallocation across multiple tenants; and (iii) it is evaluated as a *classifier* of reward hacking, against both a progress-triggered rule that knows the synthetic peak location (StopAt-0.65) and a loss-plateau detector, showing that the eval signal is needed to keep precision high and FPR low (Table 1).

7. Conclusion

We introduced EvalStop, a composable scheduling primitive that uses world feedback (downstream eval scores) to detect and correct reward overoptimisation in RLHF training. Our evaluation reframes scheduler-level early stopping as a detection problem: on a workload containing both reward-hacking and healthy RLHF runs, EvalStop achieves 98% precision and 1.5% false-positive rate while delivering $+9\%$ JCT and -22% wasted compute on RLHF-heavy mixes, and it composes with every base scheduler we tested. Two strong simpler competitors (a trivial fixed-progress stop rule and a loss-plateau detector) either trade catastrophic FPR or half the recall for comparable systems metrics, showing that the world-feedback signal is doing the detection work.

As RLHF workloads grow on fine-tuning platforms, the gap between proxy signals and world feedback will become an increasingly important consideration for resource management. EvalStop shows that closing this gap at the scheduler level, with a simple detector, is both feasible and effective in simulation; validating on real training traces is the next step.

References

Choudhury, A., Wang, Y., Pelkonen, T., Srinivasan, K., Jain, A., Lin, S., David, D., Soleimanifard, S., Chen, M., Yadav, A., Tijoriwala, R., Samoylov, D., and Tang, C. MAST: Global scheduling of ML training across geo-distributed datacenters at hyperscale. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 563–580, 2024.

Duan, J., Song, Z., Miao, X., Xi, X., Lin, D., Xu, H., Zhang, M., and Jia, Z. Parcae: Proactive, liveput-optimized DNN training on preemptible instances. In *21st USENIX Symposium on Networked Systems Design and Implemen-*

tation (NSDI), pp. 1121–1139, 2024.

Gao, L., Schulman, J., and Hilton, J. Scaling laws for reward model overoptimization. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10835–10866. PMLR, 2023.

Gu, J., Chowdhury, M., Shin, K. G., Zhu, Y., Jeon, M., Qian, J., Liu, H., and Zhuo, C. Tiresias: A GPU cluster manager for distributed deep learning. In *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 485–500, 2019.

Hu, J., Tao, X., Peng, W., et al. OpenRLHF: An easy-to-use, scalable and high-performance RLHF framework. *arXiv preprint arXiv:2405.11143*, 2024.

Jayaram, K. R., Muthusamy, V., Thomas, G., Verma, A., and Purcell, M. Sia: Heterogeneity-aware, goodput-optimized ML-cluster scheduling. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP)*, pp. 642–657, 2023.

Kong, L., Xu, Y., Jiao, L., and Xu, C. Deadline-aware online scheduling for LLM fine-tuning with spot market predictions. *arXiv preprint arXiv:2512.20967*, 2025.

Moskovitz, T., Singh, A. K., Strouse, D., Sandholm, T., Salakhutdinov, R., Dragan, A., and McAleer, S. Confronting reward model overoptimization with constrained RLHF. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Motwani, R., Phillips, S., and Torng, E. Non-clairvoyant scheduling. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 422–431, 1994.

Narayanan, D., Santhanam, K., Kazhamiaka, F., Phanishayee, A., and Zaharia, M. Heterogeneity-aware cluster scheduling policies for deep learning workloads. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 481–498, 2020.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., and Lowe, R. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Pan, A., Bhatia, K., and Steinhardt, J. The effects of reward misspecification: Mapping and mitigating misaligned models. In *The Tenth International Conference on Learning Representations (ICLR)*, 2022.

Prechelt, L. Early stopping — but when? In Orr, G. B. and Müller, K.-R. (eds.), *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, pp. 55–69. Springer, 1998.

Qiao, A., Agrawal, S. K., Gandham, S., Carrell, A., Phanishayee, A., Zaharia, M., and Stoica, I. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 1–18, 2021.

Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C. D., and Finn, C. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Rafailov, R., Chittepu, Y., Park, R., Sikchi, H., Hejna, J., Knox, W. B., Finn, C., and Niekum, S. Scaling laws for reward model overoptimization in direct alignment algorithms. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*, 2024.

Sheng, Y., Cao, S., Li, D., Hooper, C., Lee, N., Yang, S., Chou, C., Zhu, B., Zheng, L., Keutzer, K., Gonzalez, J. E., and Stoica, I. S-LoRA: Serving thousands of concurrent LoRA adapters. In *Proceedings of Machine Learning and Systems 6 (MLSys)*, 2024.

Skalse, J., Howe, N. H. R., Krasheninnikov, D., and Krueger, D. Defining and characterizing reward hacking. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Stiennon, N., Ouyang, L., Wu, J., Ziegler, D. M., Lowe, R., Voss, C., Radford, A., Amodei, D., and Christiano, P. Learning to summarize from human feedback. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Wu, B., Zhu, R., Zhang, Z., Sun, P., Liu, X., and Jin, X. dLoRA: Dynamically orchestrating requests and adapters for LoRA LLM serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 911–927, 2024.

Xue, C., Pan, Y., Cui, W., Chen, Q., Zhang, S., He, B., and Guo, M. MuxTune: Efficient multi-task LLM fine-tuning in multi-tenant datacenters via spatial-temporal backbone multiplexing. *arXiv preprint arXiv:2603.02885*, 2026.

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

Zhang, G., Vippagunta, S., Nandagopal, R., Raman, S., Xu, J., and Pfeiffer, M. AAPA: An archetype-aware predictive autoscaler with uncertainty quantification for serverless workloads on Kubernetes. In *IEEE High Performance Extreme Computing Conference (HPEC)*, 2025b.

Zhang, G., Sun, C., Zhao, K., Sarkani, S., and Fossaceca, J. M. When does deep RL beat calibrated baselines? A benchmark study on adaptive resource control, 2026.

Zhang, H., Stafman, L., Or, A., and Freedman, M. J. SLAQ: Quality-driven scheduling for distributed machine learning. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, pp. 390–404, 2017.

Zheng, P., Hu, R., Xu, Y., Jayarajan, A., Li, A., Ye, J., and Chen, W. Shockwave: Fair and efficient cluster scheduling for dynamic adaptation in machine learning. In *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 1–17, 2023.

A. Simulator Configuration

Table 6 lists the full simulator parameterisation. Training curves are seeded per-job for reproducibility. RLHF curves come in two regimes: a *hacking* regime (60% of RLHF jobs by default), calibrated to the overoptimisation dynamics of Gao et al. (2023), in which the eval peak occurs at 55–75% of training progress and then degrades; and a *healthy* regime (40%) in which the eval score rises monotonically with noise through progress=1.0. Training loss in both regimes decays exponentially to a plateau at 50–70% progress, matching real RLHF convergence. Ground-truth hacking labels and peak progress are attached to each job but hidden from the scheduler via a whitelist-based proxy (`SchedulerJobView`), so schedulers observe only the current loss, current eval score, and historical eval checkpoints.

B. E1: Mixed Workload Full Results

Table 7 presents the complete E1 results (50% LoRA, 30% DPO, 20% RLHF, 32 GPUs) omitted from the main text for space. EvalStop+SRTF achieves the best JCT and competitive TTFUC with no fairness penalty.

C. E5–E6: Numerical Details

Tables 8 and 9 provide numerical values for the threshold and eval-frequency sensitivity experiments plotted in Figures 3 and 4.

Table 6. Simulator hyperparameters and job-type profiles.

Parameter	**Value**
Cluster / Workload	
Total GPUs	32 (E1) / 64 (E2–E6)
Preemption overhead	2.0 min
Jobs / Tenants / Seeds	200 / 5 / 5
Arrival process	Poisson, λ=1.0 jobs/min
LoRA jobs	
Duration / GPUs / Eval freq	10–60 min / 1–2 / 10% prog.
Eval curve	Monotonic (exponential)
DPO jobs	
Duration / GPUs / Eval freq	30–120 min / 2–4 / 20% prog.
Eval curve	Diminishing returns
RLHF jobs	
Duration / GPUs / Eval freq	60–360 min / 4–8 / 15% prog.
Reward-hack fraction	60% hacking, 40% healthy
Loss plateau	$\mathcal{U}(0.50, 0.70)$ progress
Hacking peak / drop	$\mathcal{U}(0.55, 0.75)$ / $\mathcal{U}(0.10, 0.30)$
Healthy eval curve	Monotone concave + noise
Evaluation	
Quality τ / min improvement	0.3 / 1% relative
Eval duration	1 / 3 / 5 min (small / med / large)

Table 7. E1: Mixed workload (50/30/20 LoRA/DPO/RLHF, 32 GPUs, 200 jobs, 5 seeds). Precision / FPR columns only defined for stop-enabled schedulers; "n/a" = no stops issued.

Scheduler	JCT	TTFUC	Wasted	Saved	Prec.	FPR
FIFO	450$\pm$57	381$\pm$54	34.2%	0.0%	n/a	0.0%
SJF-Est	406$\pm$57	336$\pm$54	34.7%	0.0%	n/a	0.0%
SRTF-Est	361$\pm$39	290$\pm$36	33.9%	0.0%	n/a	0.0%
LossAware	722$\pm$51	273$\pm$39	34.5%	0.0%	n/a	0.0%
EvalSched	1061$\pm$107	548$\pm$34	33.4%	0.0%	n/a	0.0%
StopAt0.5+SRTF	**290**$\pm$26	240$\pm$25	21.0%	33.3%	58.9%	9.7%
StopAt0.65+SRTF	312$\pm$29	256$\pm$28	24.3%	22.5%	58.9%	9.7%
LossPlateau+SRTF	354$\pm$48	287$\pm$45	32.0%	3.6%	30.8%	13.1%
EvalStop+SRTF	344$\pm$35	279$\pm$32	**28.5%**	8.5%	**100%**	**0.0%**

Table 8. E5: Threshold sensitivity (EvalStop+SRTF, 80% RLHF, 64 GPUs, 5 seeds). Row with k=2 (our default) is highlighted.

k	JCT	TTFUC	Wasted	Stopped
1	874$\pm$50	761$\pm$47	19.4%	127
2	**1018**$\pm$53	**883**$\pm$50	**26.9%**	**93**
3	1093$\pm$56	946$\pm$54	32.8%	57
4	1123$\pm$61	971$\pm$58	34.6%	2
5	1123$\pm$59	971$\pm$56	34.5%	0

Table 9. E6: Eval frequency sensitivity (EvalStop+SRTF, 80% RLHF, 64 GPUs, 5 seeds). ΔJCT and ΔWasted are reported vs. SRTF-Est alone (JCT=1124, Wasted=34.5%).

Eval freq	JCT	Wasted	Saved	ΔJCT vs SRTF
5%	867$\pm$75	17.6%	24.1%	+22.8%
10%	993$\pm$34	23.2%	14.8%	+11.6%
15%	**1018**$\pm$53	**26.9%**	**9.9%**	**+9.4%**
20%	1111$\pm$96	34.2%	4.8%	+1.1%
30%	1123$\pm$63	29.8%	0.7%	+0.1%