Recovering Temporal Slack on GPUs

Serving, batch, training and evaluation share GPUs for up to 1.5066× the useful tokens per GPU-hour and 1.65× the improvement rounds

Pranav Varshney
Nodus Compute

Viswa Kotra
Nodus Compute

Abstract

GPU utilization is usually reported as device occupancy, but operators pay for useful outcomes: interactive requests that meet their SLO, batch items that finish, training steps that preserve correctness, and complete model-improvement rounds. We unify four real-GPU campaigns that recover temporal slack across these workload classes. Reactive batch harvesting behind interactive serving adds 2.03 million batch tokens per GPU-hour at a 26–30% increase in interactive TTFT p95; an oracle with traffic foresight raises utilization from 51.5% to 93.1% at a 7–9% TTFT cost. A combined test on one GPU preempts and resumes 64/64 batch items with 0 failures while serving an interactive request in 0.501 s. Co-locating RL rollout and training on eight GPUs produces 882 versus 505 steps per GPU-hour and raises utilization from 48% to 96%, though the original loss-match gate is below the measured run-to-run noise floor. A separate shared serving engine on one GPU produces 49.54 million useful tokens per GPU-hour from live and offline traffic against 32.88 million for a dedicated two-GPU split, a 1.5066× gain at 0.9982 live SLO attainment. Finally, one GPU runs a five-round sample-train-hot-load-evaluate loop beside live traffic at 1.65× the best SLO-qualified baseline under the preregistered accounting and 3.31× on total GPU-seconds. The synthesis shows where temporal sharing helps, what latency it costs, and why busy time alone is not the result.

1 Introduction

A GPU-hour is purchased whether the device is producing a user-visible result or waiting for the next phase of a pipeline. Interactive serving idles between bursts. Offline generation waits on separately reserved capacity. Reinforcement-learning pipelines alternate between rollout and training. Model-improvement loops serialize sampling, fine-tuning, deployment, and evaluation while a second GPU keeps production traffic alive. These gaps are different manifestations of temporal slack.

The conventional utilization percentage does not decide whether filling that slack is useful. A batch kernel can make the GPU look busy while pushing live requests past its latency objective. A co-located training run can finish more steps while changing the result. We therefore account for useful outcomes per paid GPU-hour and retain the guard metric for each workload: live SLO attainment and TTFT, item completion after preemption, loss or adapter equality, evaluation acceptance, and errors.

This paper combines the harvesting, live-plus-batch, rollout-plus-training, and train-serve-evaluate reports into one evaluation, and it absorbs reports 010 and 011, Train, Serve and Evaluate on One GPU and Live Traffic and Batch on One GPU, whose full results are the

sections on the train-serve-evaluate loop and on live and batch generation. The experiments do not share one universal unit, so we do not average their ratios. Instead, we ask the same systems question in each campaign: how much otherwise idle paid time becomes valid work, and what measurable service or correctness cost accompanies it?

2 Metric and accounting rules

For a campaign with useful output U, G occupied GPUs, and elapsed time T, useful work per paid GPU-hour is U divided by G×T. U is campaign-specific but always completion-qualified. Batch tokens count only when their item eventually completes. Live output counts only when the request meets its SLO. RL output is completed steps. A model-improvement output is a complete sample, train, deploy, and evaluate round whose adapter passes the acceptance rule.

Every occupied GPU is charged through the end-of-window drain. Work discarded by preemption, failed requests, and late live output contributes zero. We separately report device utilization because it helps explain slack, but a utilization increase is not converted into a useful-work gain.

Table 1: Campaigns and primary useful-work units.

Campaign	Useful output	Guard metric	Hardware
Serving harvest	Completed batch tokens	Interactive TTFT p95	Current flagship GPU
Live plus batch	SLO live + completed batch	Live SLO	1× flagship GPU
RL co-location	Completed RL steps	Loss noise control	8× previous-generation training GPU
Improvement loop	Accepted complete rounds	Live SLO $\geq$0.95	1–2× flagship GPU

3 Reactive batch harvesting behind serving

The serving-harvest campaign replays the same interactive arrivals with and without a latency-tolerant batch backlog. The deployable harvester reacts to live demand without future knowledge. A separate oracle sees future arrivals and provides an upper bound on recoverable slack. Both regimes report additional completed batch tokens, GPU utilization, and the change in interactive TTFT p95 relative to interactive-only service.

Table 2: Measured harvesting outcomes. The oracle is an upper bound, not a deployable result.

Regime	Recovered work	GPU utilization	Interactive cost
Reactive	+2.03M tokens/GPU-h	Measured separately	+26–30% TTFT p95
Perfect-foresight oracle	Upper bound	51.5% → 93.1%	+7–9% TTFT p95

The reactive system converts slack into 2.03 million completed batch tokens per GPU-hour without adding a GPU. That work is not free: live TTFT p95 rises by 26–30%. Whether the exchange is acceptable depends on SLO headroom. The oracle shows that most idle capacity can be recovered at a smaller 7–9% latency cost if arrivals are known, but it does not establish that a production predictor reaches that bound.

This result illustrates why occupancy cannot be the objective. The system must optimize a constrained pair: batch completions and live latency. Reporting only the recovered tokens would hide the user-facing cost; reporting only utilization would hide both.

4 Preemption and recovery under a combined system

A second test on the flagship GPU enables harvesting together with memory isolation and idle release on a Qwen3-8B base model and one LoRA tenant. It records cold readiness, memory residency, reactivation latency, deterministic output equality, batch completions,

failures, preemptions, resumptions, and an interactive request issued while batch work is active.

Table 3: Combined-system completion and latency checks on one flagship GPU.

Measurement	Result
Active batch items	64
Preemptions / resumptions	64 / 64
Completed items / failures	64 / 0
Completed batch output	16,384 tokens
Concurrent interactive request	HTTP 200 in 0.501 s

All preempted items resume and finish, so their tokens satisfy the completion-qualified numerator. The interactive request also succeeds while the backlog is active. This experiment establishes the end-to-end state transition and completion invariant when three mechanisms run together.

5 Co-locating RL rollout and training

RL pipelines commonly allocate separate GPU pools to rollout generation and training. One pool waits while the other phase runs. We compare split placement with co-location on 8 previous-generation GPUs over 3 seeds and 8 RL steps per run. The same pipeline is executed in both arms and errors are counted.

Table 4: Preliminary RL placement result, 8 previous-generation GPUs, 3 seeds, 8 steps.

Placement	RL steps/GPU-h	GPU utilization	Errors
Split rollout and training	505	48%	0
Co-located	882	96%	0

Co-location yields 1.74–1.75× the RL steps per GPU-hour and approximately doubles measured utilization. The direction matches the pipeline structure: the same GPUs stay active through rollout and optimization instead of alternating between two paid pools.

5.1 Correctness noise floor

The preregistered 0.1% relative loss-match criterion was too strict for observed run-to-run variability. Two nominally identical split runs already diverge at step 1. Split-versus-co-located differences over 8 steps are of the same order as split-versus-split and co-located-versus-co-located variation. This is not an equivalence result. Correctness requires a noise-calibrated distribution over longer, multi-seed runs before the throughput gain can support a quality-preserving RL claim.

6 Live and batch generation on one serving engine

The live-plus-batch campaign uses one flagship GPU and a single FP8/EAGLE3 serving engine. A bursty live trace runs beside a continuously refilled offline backlog. The strongest dedicated reference uses one live GPU and one batch GPU and may use the same fast configuration, so the comparison credits co-location rather than precision or speculation. Seeds 17, 29, and 43 are pooled after charging each occupied GPU through drain. Useful tokens are the output tokens of completed batch requests plus those of live requests that met their SLO, so live tokens that missed the SLO and failed requests earn nothing. The same live arrivals and batch prompts were replayed for every arm, backfill never looked at future live arrivals, and for each seed the dedicated reference took the best of the pre-registered stock candidates. All settings were pre-registered before any GPU time, in a preregistration with SHA-256 9a804ecee4c393a191ae3e2b3885baa0cfbba0d7b98cf59b90987b876b1b1929.

Table 5: Useful tokens per GPU-hour pooled over 3 seeds.

Arm	Useful tokens/GPU-h	Ratio	Live outcome
Shared GPU, priority	49.54M	1.5066×	0.9982 SLO; 0 errors
Shared GPU, FCFS	49.52M	n/a	Same workload
Dedicated live + batch	32.88M	baseline	Separate GPUs

The shared engine clears the preregistered 1.5× gate by a thin margin. FCFS reaches 49.52 million useful tokens per GPU-hour, essentially matching priority at 49.54 million. The gain therefore comes from placing both queues on one engine, not from the priority policy.

Live TTFT p95 rises from 0.027 to 0.063 s, while SLO attainment remains 0.9982 with 0 errors. As in harvesting, co-location exchanges latency headroom for completed background work.

7 Closing the train-serve-evaluate loop on one GPU

The improvement-loop campaign runs live serving, rejection sampling, LoRA training, adapter hot-loading, and evaluation for Qwen3-1.7B on GSM8K. All arms execute 5 rounds and evaluate on the same deterministic 500-question subset. An adapter is accepted only if it does not regress evaluation accuracy. The gate requires at least 1.5× the best SLO-qualified baseline and live SLO attainment of at least 0.95.

Table 6: Five-round improvement loop. The standard baseline is reported under gate accounting and total GPU-seconds.

Arm	Rounds/GPU-h	GPU-seconds	Live SLO
Nodus, one shared GPU	28.7704	625.643	0.96190
Standard loop, gate field	17.4046	2,068.419	1.0
Standard loop, all GPUs	8.70	2,068.419	1.0
Naive shared GPU	16.68	n/a	0.4047

Nodus completes 1.65× more rounds per GPU-hour than the best qualified baseline under the preregistered field and 3.31× more on total GPU-seconds. It serves 2,171 of 2,257 requests within SLO with 0 errors. Naive sharing is disqualified: 1,595 of 3,941 requests meet SLO, attainment falls to 0.4047, and 2,346 errors occur. The guard and hot-load path are part of the result.

All 5 Nodus rounds are accepted and accuracy rises from 0.196 to 0.346; the standard loop moves from 0.196 to 0.26. These are one-seed within-run trends without confidence intervals. A product-path repeat passes all 5 gates at 48.2919 rounds per GPU-hour, 0.99683 live attainment, 0 loop errors, and 0 errors across four adapter swaps. Its baseline is reused from the harness, so it validates the product path rather than forming a new matched headline.

The subsections below carry report 010's full measurements of this loop: its setup and accounting, the per-arm throughput and live-serving tables, live latency, per-round accuracy, the ablation and the product-path validation.

7.1 Setup and accounting of the loop

Model and task. All arms run Qwen/Qwen3-1.7B [1] at a pinned revision with seed 0 for 5 rounds on GSM8K [2]. Every arm is evaluated on the same deterministic 500-question subset. A round's adapter is accepted only if it does not regress evaluation accuracy.

Serving. The serving engine is vLLM v0.30.0. Each arm carries a live request stream alongside the loop. A live request meets its SLO when it meets both its time-to-first-token (TTFT) and time-per-output-token (TPOT) targets, and SLO attainment is the share of offered live requests that do so. Errors count as misses.

Arms. Nodus runs live serving, rejection-sampling sampling, LoRA training and evaluation together on one GPU, with priority scheduling, the SLO guard and adapter hot-loading. Naive shared GPU (b1) runs the same work on one GPU without the guard. Standard loop (b2) trains, saves, restarts the serving engine and evaluates on one GPU, while live serving runs on a second GPU with first-come first-served scheduling.

Accounting. GPU-seconds count every GPU an arm occupies. The harness field for rounds per GPU-hour divides by the loop GPU's wall time only, which for b2 leaves out its live-serving GPU. We report b2 both ways, and the pre-registered gate uses the conservative field. All numbers come from one measured run whose frozen summary has SHA-256 1bd64c47287764ad831140473e38982a9a54ac1529bee6f4e7de1f01612c9228.

7.2 Loop throughput under a live SLO

Table 7: Measured loop throughput for 5 rounds of Qwen3-1.7B on GSM8K. b2 is shown under the gate's conservative field and on all of its GPU-seconds.

Arm	GPUs	GPU-seconds	Rounds per GPU-hour
Nodus	1	625.6429666420008	28.770402545418175
b1: naive shared GPU	1	1079.13	16.68
b2: standard loop, gate field	2	2068.418653431996	17.404600340587475
b2: standard loop, all GPU-seconds	2	2068.418653431996	8.70

Table 8: Live serving quality while the loop runs.

Arm	Live SLO attainment	SLO-met of offered	Live errors
Nodus	0.9618963225520603	2171 of 2257	0
b1: naive shared GPU	0.4047	1595 of 3941	2346
b2: standard loop	1.0	4033 of 4033	0

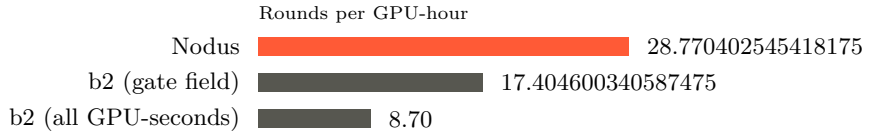


Figure 1: Rounds per GPU-hour. b2 is shown both under the gate's conservative field and on its total GPU-seconds.

Nodus completes all 5 rounds in 625.6429666420008 GPU-seconds on one GPU, or 28.770402545418175 rounds per GPU-hour. Its live stream meets the SLO on 2171 of 2257 offered requests, an attainment of 0.9618963225520603, with 0 errors. The standard loop b2 keeps perfect live attainment of 1.0 but needs 2068.418653431996 GPU-seconds on two GPUs. Against b2, Nodus delivers 1.65× more rounds per GPU-hour under the gate's conservative field and 3.31× on total GPU-seconds. The verbatim gate result names b2 as the best qualified baseline and records a pass with no failure reasons.

Sharing the GPU without the guard does not work. The naive shared arm b1 reaches 16.68 rounds per GPU-hour, but its live stream meets the SLO on only 1595 of 3941 requests, an attainment of 0.4047, with 2346 errors. It falls below the 0.95 SLO requirement and is disqualified.

7.3 Live serving latency

With the loop running on the same GPU, Nodus serves live requests at a TTFT of 0.0507 s at p50 and 0.1319 s at p95, and a TPOT of 5.91 ms at p50 and 9.12 ms at p95. The trainer paused for a total of 0.0106 s across the run to protect live traffic. For comparison, b2's dedicated live GPU reaches a TTFT p95 of 0.0432 s.

7.4 Accuracy per round

Table 9: GSM8K accuracy on the same 500-question subset after each round. Rounds marked "not accepted" regressed and were not deployed.

Arm	Round 1	Round 2	Round 3	Round 4	Round 5
Nodus	0.196	0.246	0.266	0.284	0.346
b1: naive shared GPU	0.18	0.208	0.264	0.264 (not accepted)	0.264 (not accepted)
b2: standard loop	0.196	0.246	0.246 (not accepted)	0.258	0.26

Nodus accepted all 5 rounds and its accuracy rose from 0.196 to 0.346. The standard loop moved from 0.196 to 0.26. These are within-run trends from one seed on 500 questions without confidence intervals, so we report them as measured and do not claim a statistically significant accuracy difference between arms.

7.5 Ablation

An earlier run of the same design without priority scheduling or the trainer guard reached live SLO attainment of 0.9315 with a TTFT p95 of 2.21 s, and the pre-registered gate failed on SLO. That run used a smaller evaluation set, so its accuracies are not directly comparable with the results above. The comparison shows that sharing one GPU across serving and training only meets the SLO once the guard is in place. The earlier run's frozen summary has SHA-256 e7b0807342e54187eb36515a4585c876c17c04fdab6e4cd634134046787f00fc.

7.6 Product-path validation

We repeated the loop through the Nodus product serving path rather than the experiment harness. The run used a real GPU, Qwen3-1.7B at the same pinned revision, 5 rounds of GSM8K and the same pre-registered gate. The standard-loop baseline was not re-run alongside it. The throughput gate instead uses the harness run's 8.70 rounds per GPU-hour, so the product path must reach at least 13.05 rounds per GPU-hour, which is 1.5× that baseline. The headline numbers above come from the harness run and are unchanged.

Table 10: Pre-registered gates for the product-path run. All 5 gates pass.

Gate	Requirement	Measured	Result
Rounds complete	5	5	PASS
Accepted rounds	$\geq$1	4	PASS
Live SLO attainment	$\geq$0.95	0.99683	PASS
Live errors, including swaps	0	0	PASS
Rounds per GPU-hour	$\geq$13.05	48.2919	PASS

The live stream offered 1,261 requests, of which 1,257 met the SLO, an attainment of 0.99683 with 0 errors. All 4 misses were TTFT misses on requests that started during sampling. No request that started during evaluation, training or an adapter swap missed. The loop completed 48.2919 rounds per GPU-hour with 0 loop errors, and all four adapter swaps completed with 0 errors.

Table 11: Live SLO misses by the loop phase in which each request started.

Phase at request start	Misses	Requests
Evaluation	0	254
Sampling	4	437
Training	0	558
Adapter swap	0	12

Accuracy rose from 0.258 to 0.298 across the run, a smaller gain than in the harness run. Like the harness run, this validation covers one seed, one model and one task. The artifact manifest has SHA-256 d7e55e979afc54f6d6ee7e4c14f9f49ed756fc0dc966204358915f5f5b41aaef.

Table 12: Per-round results on the product path, using the run's own round numbering.

Round	Kept samples	GSM8K accuracy	Accepted
0	226	0.258	yes
1	294	0.240	no
2	272	0.282	yes
3	317	0.292	yes
4	332	0.298	yes

7.7 Earlier product-path attempts

Four earlier product-path attempts did not pass. R1 failed during setup. R2 failed on a missing dependency. R3 failed when an adapter request returned 404. R4 completed but reached only 89.0% live SLO attainment because of a stale-alias bug. The run reported above followed these attempts.

8 Cross-campaign synthesis

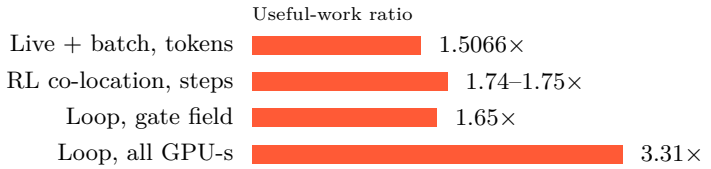


Figure 2: Measured useful-work gains over each campaign's matched baseline. Ratios use different workload-specific units and are shown together only as a cross-campaign index. Each bar is a result reported in its own section above.

Slack exists at several time scales: milliseconds between live requests, seconds across batch preemption, minutes across rollout and training, and hundreds of seconds across model-improvement stages. The mechanism changes with the time scale, but the accounting rule does not: count completed, accepted work; charge every occupied GPU; retain the foreground guard.

Table 13: Measured result and evidence boundary for each campaign.

Campaign	Established	Boundary
Reactive harvest	+2.03M tokens/GPU-h	TTFT +26–30%
Live + batch	1.5066× useful tokens/GPU-h	One model/GPU type
RL co-location	1.74–1.75× steps/GPU-h	Correctness unresolved
Improvement loop	1.65× gate; 3.31× total GPU-s	One seed/model/task

8.1 Evidence boundary

The results span different hardware, models, traces, seeds, and useful-work units. They cannot be averaged into one global utilization multiplier. The serving campaigns do not prove arbitrary foreground isolation; the RL campaign is preliminary; the improvement loop does not establish general accuracy uplift. Traffic knowledge from an oracle is never treated as deployable.

9 Reproducibility

Each campaign freezes arrivals, prompts, output limits, seeds, and arm definitions before the measured run. Useful-work denominators include occupied GPU count and drain time. Error counts and guard outcomes remain adjacent to throughput. The source evidence retains

preregistration digests where available, frozen summaries, per-seed results, and product-path gate output.

A reproduction should repeat each matched campaign independently before adding longer RL runs with a split-versus-split noise control, practical forecast policies between reactive and oracle harvesting, and additional models and GPU types. Those extensions must preserve completion-qualified accounting rather than substituting raw utilization.

10 Conclusion

Temporal slack is recoverable only when foreground service and correctness remain part of the objective. Across four real-GPU campaigns, Nodus converts idle paid time into completed batch tokens, RL steps, live-plus-batch output, and full model-improvement rounds. The gains range from 1.5066× useful live-plus-batch tokens to 3.31× improvement rounds on total GPU-seconds, with explicit latency and evidence costs. A busier GPU matters when the additional work finishes, passes its acceptance rule, and preserves the service commitment that justified sharing it.

References

[1] Qwen Team. Qwen3 Technical Report. arXiv preprint, 2025.

[2] K. Cobbe et al. Training Verifiers to Solve Math Word Problems. arXiv preprint, 2021.