Harvesting Idle GPU Cycles: Preemptible Batch Work Behind Interactive LLM Serving, and Co-located RL Rollout and Training

Pranav Varshney, Viswa Kotra
Nodus Compute

Abstract

GPUs are paid for by the hour, but much of a paid GPU-hour produces no useful AI output: interactive LLM serving is provisioned for bursts and idles between them, and reinforcement-learning (RL) pipelines that place rollout generation and training on disjoint GPU pools leave each pool waiting on the other. We argue that the right figure of merit for such systems is *useful AI output per paid GPU-hour*, and we report measurements of two ways to raise it. First, we ran preemptible batch inference in the idle capacity behind interactive serving. A reactive harvester added +2.03M batch tokens per GPU-hour at a cost of +26–30% interactive time-to-first-token (TTFT) p95. With perfect traffic foresight (an oracle upper bound), GPU utilization rose from 51.5% to 93.1% at +7–9% TTFT p95. In a combined-system test on one GPU, 64 active batch items were preempted 64 times and resumed 64 times while an interactive request returned in 0.501 s, with all 64 items completed and 0 failures. Second, as a *preliminary* result, co-locating RL rollout and training on the same GPUs yielded 1.74–1.75$\times$ RL steps per GPU-hour versus split placement (GPU utilization $\sim$48% $\rightarrow$ $\sim$96%) on eight GPUs over 3 seeds and 8 steps. We also report a negative methodological finding: a strict 0.1% loss-match correctness gate is below run-to-run noise, since two identical split runs already diverge at step 1. We describe the problem setting, evaluation methodology, and results. System internals are out of scope.

1 Introduction

Accelerator time is a major cost of deploying and improving large language models (LLMs). Yet the unit that is purchased, a GPU-hour, is rarely the unit that matters. What an operator actually wants is *useful AI output*: interactive responses delivered within their latency targets, batch tokens that complete, and RL training steps that make correct progress. Between these two units sits idle time, and idle time is common in two settings that dominate modern LLM fleets.

Interactive serving idles between bursts. Interactive LLM endpoints must absorb bursty arrivals with low time-to-first-token (TTFT). Provisioning for the burst means that, on average, a substantial fraction of the GPU is unused. Meanwhile, many workloads (offline evaluation, synthetic data generation, document processing, embedding and summarization jobs) are latency-tolerant and could absorb that slack if they could be pushed out of the way the moment interactive demand returns.

Split RL pipelines idle by construction. RL post-training of LLMs [Ouyang et al., 2022, Schulman et al., 2017] alternates between generating rollouts and updating the policy. When the two phases are placed on disjoint GPU pools, each pool is idle while the other phase runs. Frameworks such as OpenRLHF [Hu et al., 2024], HybridFlow/veRL [Sheng et al., 2025] and RLHFuse [Zhong et al.,

Preprint.

2024] have explored placement choices for these phases. The measured cost of leaving them split is the starting point for our second study.

Harvesting idle resources is an old idea in cluster management [Verma et al., 2015, Lo et al., 2015, Chen et al., 2019, Zhang et al., 2014, Guo et al., 2019] and in deep-learning cluster scheduling [Xiao et al., 2018, 2020, Li et al., 2023]. What is new in the LLM setting is the combination of (i) very large, stateful GPU-resident engines whose cold start is measured in minutes rather than milliseconds, (ii) a latency-critical tenant whose SLO is defined on token-level metrics such as TTFT, and (iii) training workloads whose *correctness* must be checked, not just their speed.

Contributions. This paper is a measurement study. We make the following contributions:

- We frame GPU harvesting around a single metric, *useful AI output per paid GPU-hour*, and define what counts as useful for interactive, batch and RL work (§2).
- We measure preemptible batch harvesting behind interactive serving in a reactive regime and under an oracle with perfect traffic foresight, reporting both the gain and the interactive TTFT cost (§4).
- We report a combined-system test on a single GPU in which harvesting, memory isolation and idle release ran together, including 64 preemptions and 64 resumptions with 0 failures (§5).
- We report preliminary measurements of co-locating RL rollout and training on the same GPUs, together with the finding that a strict loss-match correctness gate is below run-to-run noise and therefore cannot serve as a correctness criterion (§6).

Scope of disclosure. This paper describes *what* the research prototype did, the problem setting, the evaluation methodology and the measured outcomes. The scheduling and preemption policies, their parameters and the system architecture are proprietary and intentionally not described. We have tried to make the evaluation protocol precise enough that the reported quantities are unambiguous, and we are explicit about where the evidence is preliminary.

2 Problem Setting and Metric

2.1 Useful AI output per paid GPU-hour

Let a deployment pay for G GPUs for a wall-clock interval of length T, so that it pays for $G \cdot T$ GPU-hours regardless of how they are used. Let U denote the useful output produced in that interval. We report

$$\text{Useful output per paid GPU-hour} \;=\; \frac{U}{G \cdot T}, \tag{1}$$

where U is workload-specific:

- **Interactive serving.** Responses delivered without error. The *cost* side of harvesting is measured as the change in interactive TTFT at the 95th percentile (p95) relative to running the same interactive traffic alone.
- **Batch inference.** Output tokens of batch items that complete. Work on an item that is preempted is useful only if the item eventually completes. Failed items contribute nothing.
- **RL training.** Completed RL steps (rollout followed by policy update). A step is useful only if the co-located pipeline is as correct as the split one, which raises the question of how correctness should be tested (§6.2).

Because the denominator is fixed by what is paid for, any output produced in otherwise idle capacity raises the metric directly. For scale, the on-demand price of the data-center GPUs used elsewhere in our measurements is $3.99/GPU-h ($31.92/h per eight-GPU node). Every idle fraction of such an hour is paid for in full.

We also report *GPU utilization* as a secondary, mechanism-agnostic indicator of how much of the paid capacity is busy. Utilization is not itself useful output: a GPU can be busy doing work that is later discarded. We therefore always pair utilization with an output or completion measure.

Table 1: Preemptible batch harvesting behind interactive LLM serving. Gains and costs are relative to the same interactive traffic served alone. The oracle regime has perfect traffic foresight and is an upper bound, not a deployable configuration. "–" marks a quantity not reported for that regime.

Regime	Added batch output	GPU util.	Interactive TTFT p95
Interactive only	–	51.5%	reference
Reactive harvesting	+2.03M tokens/GPU-h	–	+26–30%
Oracle (perfect foresight)	–	93.1%	+7–9%

2.2 Workload model

Interactive tenant. A latency-critical LLM endpoint receives requests whose arrivals are not under the operator's control. Its requests must never be failed in order to make room for harvested work, and its TTFT should degrade as little as possible.

Preemptible batch tenant. A latency-tolerant tenant submits batch items. Batch items may be paused at any time and resumed later. The harvester's obligation is that every admitted item eventually completes and that no item fails because it was preempted.

RL pipeline. An RL job alternates between rollout generation and training. In the *split* placement, the two phases run on disjoint GPU subsets of the same node. In the *co-located* placement, both phases run on the same GPUs. Both arms use the same model, data and hyperparameters. Only placement differs.

3 Evaluation Methodology

Harvesting regimes. We evaluate preemptible batch harvesting in two regimes. In the *reactive* regime, the harvester observes only current and past interactive load. It has no knowledge of future arrivals. In the *oracle* regime, the harvester is given perfect foresight of the interactive traffic. The oracle is not deployable. It is an upper bound that indicates how much of the idle capacity is in principle recoverable and how much of the reactive TTFT cost is attributable to not knowing the future.

Harvesting metrics. For each regime we report (i) additional batch output tokens per GPU-hour relative to interactive-only operation, (ii) GPU utilization, and (iii) the change in interactive TTFT p95 relative to interactive-only operation. Ranges are reported as measured.

Combined-system test. To check that harvesting composes with other mechanisms required in a real deployment, we ran a single server on one GPU with memory isolation, idle release and harvesting all enabled. The server hosted a Qwen3-8B base model and one LoRA tenant. We record cold-start readiness, GPU memory before and during idle release and after reactivation, reactivation latency, output equality before and after reactivation under greedy decoding, batch completions and failures, preemption and resumption counts, the latency and HTTP status of an interactive request issued while batch work is active, and errors.

RL co-location. We run the split and co-located arms on eight GPUs with 3 seeds and 8 RL steps per run. We report RL steps per GPU-hour, GPU utilization and errors. For correctness, we compare per-step training loss trajectories, and, crucially, we also compare two *identical* runs of the same arm, which establishes the run-to-run noise floor against which any split-vs-co-located difference must be judged.

Reporting discipline. All numbers in this paper are measured on real GPUs. We do not extrapolate.

4 Results: Preemptible Batch Work Behind Interactive Serving

Table 1 summarizes the harvesting results.

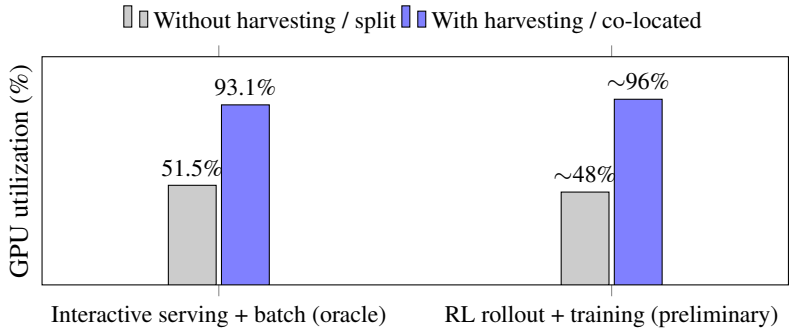


Figure 1: GPU utilization reported in this paper. Left: interactive-only serving vs. batch harvesting with perfect traffic foresight (oracle upper bound). Right: split vs. co-located RL rollout and training (preliminary, approximate values as reported). All bar heights are the reported numbers, no other values are plotted.

The gain. Reactive harvesting added +2.03M batch tokens per GPU-hour. Since the GPU-hours are paid for whether or not batch work ran, this output was obtained without any additional GPU spend. In terms of our metric, every one of these tokens increased useful output per paid GPU-hour.

The headroom. With perfect traffic foresight, GPU utilization rose from 51.5% to 93.1%. In other words, in this setting nearly half of the paid capacity sat idle under interactive-only operation, and the oracle showed that most of it can, in principle, be filled.

The cost. Harvesting is not free for the interactive tenant. In the reactive regime, interactive TTFT p95 increased by +26–30%. With perfect foresight, the cost fell to +7–9% TTFT p95. The gap between the two regimes is therefore largely a *prediction* problem: a harvester that knew when interactive requests would arrive could step aside before they arrived instead of after. We regard closing this gap with deployable forecasts as the main open problem for this part of the work.

Interpreting the trade-off. Whether +26–30% TTFT p95 is acceptable depends on how much slack the interactive SLO has. For deployments whose TTFT target leaves headroom, the reactive regime could turn idle capacity into batch output at no extra GPU cost. For deployments operating close to their TTFT target, the oracle result suggests most of the utilization gain could be achieved at a much smaller latency cost, but only to the extent that traffic can be anticipated.

5 Results: Combined-System Test

Component measurements can hide interactions. We therefore ran harvesting together with memory isolation and idle release on a single one GPU server hosting a Qwen3-8B base model and one LoRA tenant. Table 2 reports the outcome.

Preempted items complete. With 64 batch requests active, an interactive request arrived and returned HTTP 200 in 0.501 s. Over this phase, batch work was preempted 64 times and resumed 64 times. All 64 items completed with 0 failures and 16,384 output tokens. This is the property the useful-output metric requires: preempted items still complete, so their output tokens count toward useful output.

Harvesting composes with idle release. In the same server, idle release reduced GPU memory from 66,527 MiB to 4,529 MiB while idle, freeing 61,998 MiB (~60.5 GiB) that other work could use, and reactivation took 0.528 s for the base engine and 0.646 s for the LoRA engine, compared with 170.0 s for a cold start. Greedy outputs after reactivation matched the pre-idle outputs exactly in 2/2 checks. No errors were observed over the test.

What this test does and does not show. The test establishes that the mechanisms operated together on one server without failures or output changes. It is a functional, single-server test: it is not

Table 2: Combined-system test on one GPU (Qwen3-8B base + one LoRA tenant) with memory isolation, idle release and harvesting enabled together.

Measurement	Result
Service ready (cold)	170.0 s
GPU memory, active → idle	66,527 → 4,529 MiB (61,998 MiB, ∼60.5 GiB freed)
GPU memory after reactivation	65,381 MiB
Reactivation latency	base engine 0.528 s, LoRA engine 0.646 s
Reactivation vs. cold start	∼260–320× faster than cold start
Greedy outputs after reactivation	matched pre-idle outputs exactly (2/2)
Harvesting (12-item run)	12 batch items completed, 0 failed, 6,144 output tokens
Harvesting under interactive load	64 batch requests active, interactive HTTP 200 in 0.501 s
Preemptions / resumptions	64 / 64
Batch completion	all 64 items completed, 0 failures, 16,384 output tokens
Errors	0

Table 3: RL rollout + training, split vs. co-located on the same GPUs (eight GPUs, 3 seeds, 8 RL steps per run). **Preliminary.**

Placement	RL steps per GPU-hour	GPU utilization	Errors
Split (disjoint GPU pools)	505	∼48%	0
Co-located (same GPUs)	882	∼96%	0
Ratio, co-located vs. split	1.74–1.75× RL steps per GPU-hour		

a throughput benchmark, and it involves one interactive probe under load rather than a latency distribution.

6 Preliminary Results: Co-locating RL Rollout and Training

6.1 Throughput and utilization

Table 3 compares split and co-located placement of RL rollout and training on eight GPUs over 3 seeds.

Co-location gave 1.74–1.75× RL steps per GPU-hour (882 vs. 505), and GPU utilization rose from ∼48% to ∼96%, with 0 errors. The utilization figures are consistent with the motivation for the experiment: in the split arm each GPU pool is idle while the other phase runs, and co-location keeps the same GPUs busy across both phases.

6.2 Correctness: the loss-match gate is below the noise floor

A throughput gain for RL is only useful if training is unaffected. Our original correctness criterion was a strict loss-match gate: the co-located loss trajectory should match the split trajectory to within 0.1% relative loss error. The co-located arm did not match the split arm under this gate. Before attributing the difference to co-location, we measured the noise floor by comparing identical runs of each arm (Table 4).

Two identical split runs already diverge at step 1, with a maximum relative loss error of 0.7116 and a maximum absolute error of 0.0286 over 8 steps. Two identical co-located runs also diverge at step 1, with a maximum absolute error of 0.0693. A 0.1% loss-match gate is therefore below run-to-run noise: it would reject the split arm compared against itself. It cannot distinguish a correct co-located pipeline from an incorrect one.

We therefore make no correctness claim beyond the following: *over 8 steps, split-vs-co-located differences are of the same order as run-to-run variation.* This is not evidence of equivalence. Establishing it requires a gate calibrated against a split-vs-split control (for example, comparing distributions of loss or reward over many seeds and longer runs), which we have not yet run.

Table 4: Run-to-run noise in per-step training loss over 8 RL steps (eight GPUs). Both comparisons are between two *identical* runs of the same arm.

Comparison	Diverges at	Max rel. loss err.	Max abs. loss err.
Split vs. split (identical)	step 1	0.7116	0.0286
Co-located vs. co-located (identical)	step 1	–	0.0693

A methodological note. We believe this negative finding is of independent interest. Per-step exact-match or tight-tolerance loss checks are a natural first test when changing where RL phases run, but RL loss trajectories are sensitive to sources of nondeterminism even between identical configurations. Any placement study that reports a correctness pass or fail should report the corresponding identical-run control alongside it.

7 Related Work

Harvesting and co-location in datacenters. Co-locating latency-critical and best-effort work to raise utilization has a long history. Borg [Verma et al., 2015] runs production and batch jobs in the same cells and reclaims unused reservations for lower-priority work. Heracles [Lo et al., 2015] co-locates best-effort tasks with latency-critical services while protecting their tail latency through feedback-based isolation of shared resources. PARTIES [Chen et al., 2019] extends QoS-aware partitioning to multiple interactive services on one server. Fuxi [Zhang et al., 2014] manages resources and jobs at internet scale, and analyses of co-located datacenter traces study where resource efficiency is lost when online services and batch jobs share machines. Our setting differs in the resource (GPU memory and compute held by large, stateful model engines) and in the latency metric (token-level TTFT).

GPU cluster scheduling for deep learning. Gandiva [Xiao et al., 2018] introduced introspective scheduling with time-slicing, suspend-resume and migration for deep-learning jobs. AntMan [Xiao et al., 2020] dynamically scales the memory and compute of co-located deep-learning jobs to use idle GPU resources while protecting resource-guaranteed jobs. Lyra [Li et al., 2023] loans idle inference-cluster GPUs to training jobs and uses elastic scaling to reclaim them. These systems target training jobs. We study harvesting behind interactive LLM serving, where the latency-critical tenant is itself a GPU-resident engine.

LLM serving. vLLM [Kwon et al., 2023] introduced PagedAttention for memory-efficient KV-cache management. Sarathi-Serve [Agrawal et al., 2024] addresses the throughput–latency trade-off of LLM inference through chunked prefills and stall-free batching. Llumnix [Sun et al., 2024] reschedules requests across model instances at runtime. These works optimize serving of a given workload. Our study concerns filling the capacity that interactive serving leaves idle with preemptible batch work, and accounting for the latency cost that this imposes.

RL post-training systems. RL post-training of LLMs [Ouyang et al., 2022] with policy-gradient methods such as PPO [Schulman et al., 2017] interleaves generation and training. OpenRLHF [Hu et al., 2024] provides a scalable RLHF framework built on distributed serving and training components. HybridFlow (veRL) [Sheng et al., 2025] supports flexible placement of RLHF models, including co-located and separate placements. RLHFuse [Zhong et al., 2024] fuses RLHF stages to reduce idle time. Our preliminary RL results are a measurement of split versus co-located placement in terms of useful output per paid GPU-hour, together with the correctness-gate finding of §6.2, which we believe is relevant to evaluations of any of these systems.

8 Conclusion

Measured in useful AI output per paid GPU-hour, idle GPU capacity was a large and recoverable cost. Behind interactive serving, reactive preemptible batch harvesting added +2.03M batch tokens per GPU-hour at +26–30% interactive TTFT p95, and perfect traffic foresight raised GPU utilization from 51.5% to 93.1% at +7–9% TTFT p95. A combined-system test showed 64 preemptions and 64 resumptions with all 64 items completed and 0 failures while an interactive request returned in

0.501 s. For RL, preliminary measurements of co-locating rollout and training showed 1.74–1.75× RL steps per GPU-hour and utilization rising from ∼48% to ∼96%, but correctness remains open: a strict loss-match gate is below run-to-run noise, and over 8 steps split-vs-co-located differences are only shown to be of the same order as run-to-run variation. The next steps are deployable traffic forecasts to close the gap to the oracle, longer multi-seed RL runs with noise-calibrated correctness gates, and multi-node evaluation.

References

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, et al. Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024.

Shuang Chen, Christina Delimitrou, and José F. Martínez. PARTIES: QoS-aware resource partitioning for multiple interactive services. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

Jian Hu et al. OpenRLHF: An easy-to-use, scalable and high-performance RLHF framework. *arXiv preprint arXiv:2405.11143*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, et al. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2023.

Jiamin Li, Hong Xu, Yibo Zhu, Zherui Liu, Chuanxiong Guo, and Cong Wang. Lyra: Elastic scheduling for deep learning clusters. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2023.

David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. Heracles: Improving resource efficiency at scale. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2015.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, et al. HybridFlow: A flexible and efficient RLHF framework. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2025.

Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, et al. Llumnix: Dynamic scheduling for large language model serving. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024.

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 European Conference on Computer Systems (EuroSys)*, 2015.

Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, et al. Gandiva: Introspective cluster scheduling for deep learning. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, et al. AntMan: Dynamic scaling on GPU clusters for deep learning. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.

Zhuo Zhang, Chao Li, Yangyu Tao, Renyu Yang, Hong Tang, and Jie Xu. Fuxi: a fault-tolerant resource management and job scheduling system at internet scale. *Proceedings of the VLDB Endowment*, 2014.

Yinmin Zhong et al. Optimizing RLHF training for large language models with stage fusion. *arXiv preprint arXiv:2409.13221*, 2024.