Multi-Tenant RL Post-Training on One GPU

Eight customers' LoRA GRPO jobs shared one GPU at 2.137× the steps per GPU-hour of a GPU each, with final accuracy and reward within noise

Pranav Varshney
Nodus Compute

Viswa Kotra
Nodus Compute

Abstract

Reinforcement-learning post-training is now a routine customer workload, and each job usually holds its own GPU for both rollouts and training even when every job fine-tunes the same base model with a LoRA adapter. A single job leaves its GPU idle for much of each step, while rollouts wait on training and training waits on rollouts. In this study, a research prototype from Nodus, the infrastructure execution layer for AI workloads, ran many customers' LoRA GRPO jobs on one GPU. One inference engine generated rollouts for every job's current adapter, one trainer updated all adapters in shared batches over the frozen base, and the jobs were split into two groups so that one group generated while the other trained, with no job ever sampling from a stale policy. On one GPU with 8 jobs on Qwen3-0.6B, the GPU delivers 2.137× the optimizer steps per GPU-hour of a dedicated GPU per job across 3 seeds, and an independent verifier measured 2.244× on 2 fresh seeds. Final evaluation accuracy is within noise of each job run alone, +0.035 with a paired 95% CI of [+0.012, +0.056]. The pre-registered version without per-GPU sharing controls reached 1.639× and missed its 2.0 gate.

1 Introduction

Teams now post-train models with reinforcement learning on their own tasks: math, code, tool use, formatting. A typical job alternates between generating rollouts and updating the policy, and when it runs alone on a GPU, each phase leaves the GPU partly idle. Rollout batches end with a few long responses decoded at small batch size, and training steps are short bursts between them. When many customers post-train LoRA adapters of the same base model, each one still holds its own GPU.

We measure what happens when a Nodus research prototype puts several customers' jobs on one GPU. All jobs share one stock vLLM engine [1] that serves every job's current adapter, adapters are hot-reloaded after each optimizer step, and one trainer updates all adapters in shared batches over the frozen base, with each job's update computed by the same loss, gradient clipping and optimizer step as alone. The jobs are split into two groups whose phases are offset: while one group generates, the other trains, and every rollout is sampled from that job's current policy. Prior systems interleave the rollout and training phases of several RL jobs across separate pools [2] or co-run rollout and training of one job on the same GPUs [3]. Here many customers' jobs share one GPU, one engine and one trainer, and stay strictly on-policy.

The metric is optimizer steps per GPU-hour summed over all jobs, at equal learning: each job's final evaluation accuracy and mean training reward must be within noise of the same job run alone on its own GPU.

2 Experimental setup

Workload. Eight customer jobs fine-tune LoRA adapters of Qwen3-0.6B [4] with GRPO [5] on 8 different verifiable tasks, including grade-school math [6], countdown arithmetic, modular arithmetic, linear algebra word problems, counting and sorting. Each step samples 8 prompts with 8 responses each, up to 256 new tokens, and each job runs 20 optimizer steps and is evaluated on 64 held-out items. The hypotheses, arms and gate were pre-registered before the first measured run, on seeds 1, 2 and 3, with a fresh seed 4 added later.

Arms. The dedicated baseline runs each job alone on its own GPU with standard synchronous colocated GRPO, the same code path with one job, and its throughput is the sum of steps over the sum of GPU-seconds across the 8 jobs. Shared synchronous puts all 8 jobs on one engine and one trainer without overlap. The pre-registered overlap arm adds the two-group phase offset. Nodus runs the same overlap with the GPU's sharing controls set so that the engine and the trainer run concurrently. Every arm runs on the same GPU model with the same data and seeds.

Gate. The pre-registered gate requires at least 2.0× the dedicated steps per GPU-hour on every seed, a gain above 5 times the A/A spread, and paired 95% confidence bounds on the evaluation accuracy and training reward differences no lower than -0.05.

3 Results

Table 1: Optimizer steps per GPU-hour for 8 jobs on one GPU, relative to one dedicated GPU per job.

Arm	Seed 1	Seed 2	Seed 3	Mean
Nodus, overlapped and on-policy	2.074	2.187	2.149	2.137
Shared engine and trainer, synchronous	1.625	1.895	1.764	1.762
Overlap without sharing controls (pre-registered)	1.571	1.720	1.628	1.639
Sharing controls without overlap	1.653	1.634	1.602	1.630

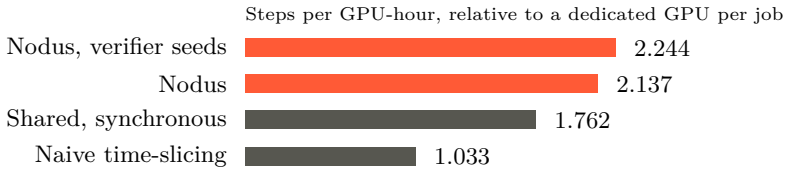


Figure 1: Optimizer steps per GPU-hour relative to one dedicated GPU per job.

More than twice the RL steps per GPU-hour. With 8 jobs on one GPU, Nodus delivers 3119.2, 2786.3 and 3123.8 optimizer steps per GPU-hour on seeds 1, 2 and 3, against 1504.1, 1273.8 and 1453.3 for one dedicated GPU per job, a mean of 2.137×. Repeating each seed gave 2.160×, 2.309× and 2.183×, and the fresh seed 4 gave 2.143×. Rollout tokens per GPU-second rise by a similar factor. The dedicated A/A on seed 1 differed by 1.9%.

Learning is within noise. Across 8 jobs, 3 seeds and 64 evaluation items each, final evaluation accuracy differs from the dedicated runs by +0.035, with a paired 95% CI of [+0.012, +0.056], and mean training reward differs by +0.016 [-0.018, +0.056]. The dedicated A/A shows a similar spread, so sharing the GPU did not measurably change final accuracy or reward. Each job's update uses the same loss, gradient clipping and optimizer step as it would alone, and every rollout comes from the job's current policy. Rollouts are sampled, so per-step rewards and gradients differ within the first steps even between two dedicated runs of the same job, and learning is compared by accuracy and reward, not bit for bit.

Where the gain comes from. Sharing one engine and one trainer is already worth 1.762×, because one engine batches all jobs' rollouts and one trainer batches all jobs' updates. Time-slicing two separate dedicated-style stacks on one GPU gives only 1.033×. Offsetting the two groups' phases pays only when the engine and the trainer can run on the GPU at the same time: overlap without that reached 1.639×, below plain sharing, and sharing controls without overlap reached 1.630×. Together they reach 2.137×. The gain grows with the number of jobs: 1.256× for 2 jobs and 1.668× for 4 on seed 1.

Pre-registration. The pre-registered overlap arm missed the 2.0 gate. The Nodus variant was defined after that arm's data and before its own, and is reported against the same gate, which it passes on every seed. Each job runs at about a quarter of its dedicated speed when 8 share the GPU, so the result raises aggregate throughput, not the speed of any single job.

4 Independent verification

A separate verifier reran the dedicated baseline and Nodus with the same code on fresh seeds 5 and 6. Nodus delivered 3292.4 and 2974.9 optimizer steps per GPU-hour, 2.275× and 2.213× the dedicated baseline, a mean of 2.244×, with repeats of 2.064× and 2.185×. Its dedicated A/A runs differed by -7.6% and 3.3%. The verifier classed the result as a novel integration of known ideas rather than a new technique, since phase interleaving across RL jobs and co-running rollout and training under GPU sharing are both published.

5 Conclusion

On one GPU, the research prototype ran 8 customers' LoRA RL post-training jobs with one shared engine and one shared trainer, kept every job on-policy, and delivered 2.137× the optimizer steps per GPU-hour of a dedicated GPU per job, 2.244× on fresh seeds, with learning within noise. Plain sharing explains most of the gain, and the phase overlap adds the rest only when the engine and trainer run concurrently. The result covers one small base model on one GPU type with short jobs. For a platform that post-trains many customers' adapters of the same base model, the result suggests the jobs could share GPUs instead of each holding one.

References

[1] W. Kwon et al. Efficient Memory Management for Large Language Model Serving with PagedAttention. SOSP 2023.

[2] T. Wu et al. RollMux: Phase-Level Multiplexing for Disaggregated RL Post-Training. arXiv preprint, 2025.

[3] Rollplex: Cross-Phase GPU Spatial Sharing for Vision Language Model Post-Training. arXiv preprint, 2026.

[4] A. Yang et al. Qwen3 Technical Report. arXiv preprint, 2025.

[5] Z. Shao et al. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. arXiv preprint, 2024.

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