Safe Fractional GPUs for AI Jobs

Capped jobs packed onto one GPU completed 1.482× the jobs per GPU-hour of a whole GPU each, with every output within tolerance of its exclusive run

Pranav Varshney
Nodus Compute

Viswa Kotra
Nodus Compute

Abstract

Most AI jobs that teams run, such as small fine-tunes, evaluation runs, embedding jobs and batch generation, use a fraction of a large GPU, yet each job usually holds a whole GPU. In this study, a research prototype from Nodus, the infrastructure execution layer for AI workloads, packed such jobs onto one GPU. Each job got a hard memory cap sized from a short probe, a ledger admitted jobs only while their caps fit, and a job that exceeded its cap failed alone and was re-run on the whole GPU. On one GPU, a seeded queue of 12 heterogeneous jobs completes 1.482× the jobs per GPU-hour of whole-GPU exclusive scheduling across 3 seeds, with A/A noise of at most 1.0%. An independent verifier measured 1.593× on 3 fresh seeds. Every job's output is within a fixed tolerance of its exclusive run: the compared evaluation, embedding, generation and vision training outputs are bit-identical, and LoRA and full fine-tune losses differ slightly, as they also do between two exclusive runs. An injected job that over-allocates memory is stopped at its cap in every run without harming any neighbour. Co-location without caps completes only 8 of the 12 jobs. The pre-registered version of the packer reached 1.345×, short of its 1.5 gate.

1 Introduction

A large share of the AI jobs a team submits are short and small: a LoRA fine-tune of a sub-billion-parameter model, an evaluation pass, a corpus embedding job, a batch of generations from a small model. Schedulers that hand out whole GPUs give each of these jobs an entire GPU, and the GPU spends much of each job's lifetime waiting on process start, model loading and data preparation.

Running several jobs on one GPU recovers that time, but only if one job cannot take its neighbours down. A job that allocates more memory than expected will crash the jobs beside it when sharing is unmanaged. Prior systems share GPUs among deep learning jobs [1] [2] [3] [4], and recent work co-locates training jobs and recovers jobs that run out of memory [5]. We measure a packer for heterogeneous modern AI jobs that fences each job in with a hard memory cap and proves containment with a deliberately leaking job.

The metric is completed jobs per GPU-hour. A job counts as completed when its final attempt exits cleanly and its output is within a pre-registered tolerance of the same job run alone on the GPU. The denominator is the time from the first job's start to the last job's exit, including every probe and every re-run.

2 Experimental setup

Workload. Each queue holds 12 jobs drawn from 8 templates in a seeded order: LoRA fine-tunes of Qwen3-0.6B and Qwen3-1.7B, a full fine-tune of Qwen3-0.6B, a loglikelihood evaluation of Qwen3-1.7B, a corpus embedding job with Qwen3-Embedding-0.6B, batch generation with Qwen3-1.7B-FP8 on vLLM [6], ResNet-50 training, and an injected job that keeps allocating memory far beyond what its probe shows. The experiment uses 3 seeds, 1, 2 and 3, on one GPU. The hypotheses, arms, tolerances and gate were pre-registered before the first measured run.

Arms. Exclusive runs one job at a time on the whole GPU, which is how whole-GPU products run jobs today, and runs twice per seed so that the two runs measure noise. Nodus packs up to 6 jobs at once. Each job's memory cap is its probed peak plus 15% and 1.5 GB, with no cap on its share of compute, the ledger admits a job only while the sum of caps stays within 76 GB, and a job that exceeds its cap is re-run on the whole GPU. Uncapped co-location runs up to 6 jobs at once with no caps. Static slots split the GPU into 2 equal fixed memory slots.

Guards. Outputs are compared with the exclusive run of the same job: training loss, evaluation likelihood, embeddings and generated text, each with a fixed tolerance. No job other than the injected one may fail. The pre-registered gate requires a mean of at least 1.5× the completed jobs per GPU-hour of exclusive scheduling.

3 Results

Table 1: Completed jobs per GPU-hour on one GPU, relative to whole-GPU exclusive scheduling, per seed.

Arm	Seed 1	Seed 2	Seed 3	Mean	Jobs completed
Nodus, warm footprint profile	1.524	1.455	1.465	1.482	12 of 12
Nodus, pre-registered version	1.410	1.347	1.278	1.345	12 of 12
Static slots, 2 per GPU	1.266	1.296	1.237	1.266	11 of 12
Uncapped co-location	1.128	1.209	1.157	1.165	8 of 12
Exclusive, repeat run	1.010	0.998	1.004	1.004	12 of 12

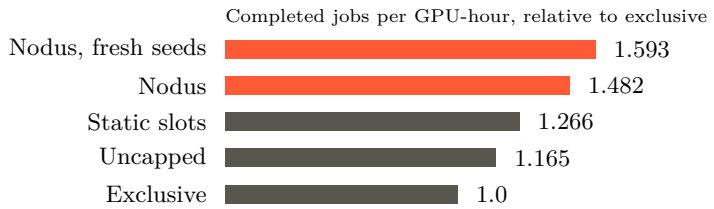


Figure 1: Completed jobs per GPU-hour relative to exclusive scheduling, mean of 3 seeds.

More jobs per GPU-hour with nothing lost. Nodus completes 172.32, 160.64 and 160.04 jobs per GPU-hour on the 3 seeds against 113.05, 110.39 and 109.22 for exclusive scheduling, a mean of 1.482×. All 12 jobs complete in every run. LoRA and full fine-tune losses differ from their exclusive runs within the first few steps, as they do between two exclusive runs of the same job, and their mean over the final steps stays within a relative 0.00043 in every packed run, including the verifier's. Evaluation likelihoods, sampled embeddings and every ResNet-50 loss are bit-identical. Generated text is token-identical. The two exclusive runs of each seed agree within 1.0%.

Containment. The injected job is stopped at its 9.25 GB cap about 6 s after it starts in every packed run, fails inside its own process, and then completes on the whole GPU at 76 GB. No other job fails in any packed run. Without caps, the jobs beside such a job are not protected: uncapped co-location completes only 8 of the 12 jobs per seed, with 6, 7 and 8 failed attempts by jobs that did nothing wrong.

Against a fixed split. Splitting the GPU into 2 equal fixed slots is the common way to share a GPU today. It also contains memory, but a job that does not fit its slot fails, and it completes 11 of the 12 jobs at 1.266×. Nodus beats it by 1.171× on the same seeds while completing every job.

Where the gain comes from. The SM-active time of the queue is the same in every arm. Exclusive scheduling leaves the GPU's SMs idle for much of each job's lifetime, with an SM-active fraction of 0.587, while Nodus keeps it at 0.898. The gain is the idle time of short jobs, during start-up, model loading and the gaps between steps, filled with other jobs. It is largest for queues of short jobs and smaller for long jobs that keep the GPU busy on their own. The version of the packer we pre-registered, which probes every job cold, reached 1.345× and missed the 1.5 gate. The warm footprint profile, which reuses each template's measured footprint, was added before its own measurements and reached 1.482×.

4 Independent verification

A separate verifier reran exclusive scheduling and Nodus with the same code on 3 fresh seeds it chose itself, 7, 11 and 13, each a different queue mix. Nodus completed 1.592×, 1.496× and 1.691× the jobs per GPU-hour of exclusive scheduling, a mean of 1.593×, with all 12 jobs completed within output tolerance in every run and the injected job contained every time. The pre-registered cold-probe version reached 1.423× on fresh seed 11. The verifier found the gain larger on queues with more batch generation jobs, whose engine start-up leaves the GPU idle.

5 Conclusion

On one GPU, the research prototype packed a queue of heterogeneous AI jobs with a hard memory cap per job and completed 1.482× the jobs per GPU-hour of whole-GPU scheduling, and 1.593× on fresh seeds, with every output within tolerance of its exclusive run and a leaking job contained in every run. It beat a fixed two-way split by 1.171×. The gain comes from short jobs leaving the GPU idle while they start and load, so it applies to queues of short jobs and shrinks for long, GPU-saturating ones. For teams that run many small AI jobs, the result suggests those jobs could share GPUs safely instead of each holding a GPU of its own.

References

[1] W. Xiao et al. Gandiva: Introspective Cluster Scheduling for Deep Learning. OSDI 2018.

[2] W. Xiao et al. AntMan: Dynamic Scaling on GPU Clusters for Deep Learning. OSDI 2020.

[3] P. Yu and M. Chowdhury. Salus: Fine-Grained GPU Sharing Primitives for Deep Learning Applications. MLSys 2020.

[4] B. Wu et al. Transparent GPU Sharing in Container Clouds for Deep Learning Workloads. NSDI 2023.

[5] E. Yousefzadeh-Asl-Miandoab et al. CARMA: Collocation-Aware Resource Manager. arXiv preprint, 2025.

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