Sub-Second Model Activation for Multi-Tenant LLM Serving

Pranav Varshney, Viswa Kotra
Nodus Compute

Abstract

Organizations that serve many fine-tuned and specialized large language models (LLMs) typically dedicate at least one GPU to each model, because moving a model onto a GPU from a cold start takes tens of seconds. Most of those GPUs sit idle most of the time. We study a Nodus research prototype whose central capability is *model activation*: changing which model a GPU served in 0.161–0.766 s (p50), versus 34.5–49.9 s for a cold start (45–300× faster), with bit-exact tensors and deterministic outputs. Nodus also suspended idle tenants to a small resident footprint (~1 GB while suspended, 4,529 MiB in a combined end-to-end test) and resumed them in 0.44–2.77 s for 8B–120B models, with 25/25 outputs matching the reference. Because activation was cheap, it was done often: on a 32-tenant replayed workload on one 8-GPU node, Nodus performed 176–203 model re-assignment events per 15 minutes at p95 0.39 s, and served all tenants at SLO on 4 GPUs instead of 32 dedicated GPUs (8× fewer), with 7.6–8.0× more SLO-met requests per GPU-second and 7.8–8.1× lower cost per SLO-met token across three traffic seeds and zero errors. At 2× traffic, a self-planned 5-GPU layout met SLO against 32 dedicated GPUs (6.4× fewer).

1 Introduction

A growing share of LLM inference demand comes not from one large model but from many: per-customer full fine-tunes, LoRA adapters on shared bases, models of different architectures, and occasional larger models served at reduced precision. Traffic to each of these tenants is bursty and, individually, low. The conventional deployment gives every tenant at least one dedicated GPU so that its weights are always resident and requests never wait for a load. The cost of this choice is that aggregate GPU utilization is low, while the bill scales with the number of tenants rather than with total load.

The obvious remedy, sharing GPUs across tenants, runs into a latency wall. If a GPU must switch from serving one model to another, the switch has historically resembled a cold start: in our measurements, bringing a model up from cold takes 34.5–49.9 s, and a full service in our combined system test became ready in 170.0 s. A switch of that duration cannot sit on the request path of an interactive SLO, so systems either avoid switching (and over-provision) or switch rarely (and tolerate SLO violations during bursts).

This paper argues that the critical quantity is the *cost of changing which model a GPU serves*. When that cost drops from tens of seconds to well under a second, re-assignment can stop being an exceptional operation and become a routine scheduling action that can be taken hundreds of times per quarter hour. Consolidation then follows: a small pool of GPUs can follow the traffic rather than reserve capacity for each tenant's peak.

We describe what Nodus achieved and how we evaluated it. We deliberately do not describe its internal mechanisms. Our contributions are:

Preprint.

- **Sub-second model activation.** Nodus changed the model a GPU served in 0.161 s (2-GPU tensor parallel) and 0.766 s (1 GPU) at p50, versus 34.5–49.9 s cold start (45–300×), with bit-exact tensors and deterministic outputs (§4.1).
- **Suspend/resume of idle tenants.** Idle tenants were held at ~1 GB of GPU memory and resumed in 0.44 s (8B), 2.77 s (32B), and 1.69 s (120B) with 25/25 outputs matching the reference (§4.2).
- **8× fleet consolidation at SLO.** On 32 tenants and three traffic seeds, Nodus used 4 GPUs instead of 32, with all per-class SLO attainments at or above 98.5% and 0 errors, while performing 176–203 re-assignment events per 15 minutes at p95 0.39 s (§4.3).

2 Problem Setting

Tenants and SLOs. A *tenant* is a model endpoint with its own request stream. We consider a single multi-GPU node hosting many tenants drawn from four classes: full fine-tunes of an 8B model, separate-architecture models, LoRA adapters on a shared 8B base, and a 32B model served in FP8. Each class has time-to-first-token (TTFT) and time-per-output-token (TPOT) targets. TPOT targets are at most 0.1 s. A deployment passes the SLO gate if window SLO attainment is at least 95%.

Objective. Given a fixed tenant set and a traffic trace, minimize the number of GPUs, and hence cost, while passing the SLO gate with zero request errors. The natural reference point is the *dedicated* deployment: one GPU per tenant, with weights permanently resident.

What makes this hard. Under the dedicated deployment, the set of models on each GPU never changes. Any deployment that uses fewer GPUs than tenants must change, over time, which models each GPU serves. Each change is on the critical path of whichever requests are waiting for the newly placed model. The latency of a change therefore bounds how aggressively a scheduler can consolidate.

Interface we evaluate. We treat Nodus as a black box offering four operations to its scheduler: *activate* a model on a GPU (or GPU group), *suspend* an idle tenant to a small resident footprint, *resume* a suspended tenant, and *harvest* spare capacity with preemptible batch work. The evaluation measures the latency, fidelity, and fleet-level consequences of these operations. Internal mechanisms, data paths, and policies are out of scope for this paper.

3 Evaluation Methodology

Hardware and pricing. Fleet experiments use one eight-GPU node. Cost is computed at on-demand GPU pricing of $31.92/h per 8 ×GPU node ($3.99/GPU-h), charging each arm for the GPUs it uses. Some component experiments use a single GPU, or eight GPUs, as noted.

Workload. 32 synthetic tenants in the four classes of §2, driven by Poisson/bursty replayed traffic at 8 req/s aggregate for 15-minute runs. We use 3 independent traffic seeds. Both arms receive the same replayed traffic. A second configuration replays the same tenant set at 2× the headline rate (single seed).

Arms. *Dedicated*: one GPU per tenant (32 GPUs). *Nodus*: the smallest GPU count we found that passes the SLO gate.

Metrics. SLO-met requests per GPU-second, dollars per million SLO-met tokens, lowest per-class SLO attainment, TTFT and TPOT percentiles, request errors, and, for Nodus, the count and p95 latency of model re-assignment events. For component experiments we report activation and resume latency, output fidelity against a reference, GPU memory in use, throughput relative to solo or partitioned baselines, and TTFT impact. All numbers are measured on real GPUs. None are simulated or extrapolated.

Table 1: Model activation versus cold start (measured).

Operation	Configuration	Latency	Fidelity
Activation (p50)	2-GPU tensor parallel	0.161 s	bit-exact tensors, deterministic outputs
Activation (p50)	1 GPU	0.766 s	bit-exact tensors, deterministic outputs
Cold start	n/a	34.5–49.9 s	n/a
Speedup of activation over cold start: 45–300×			

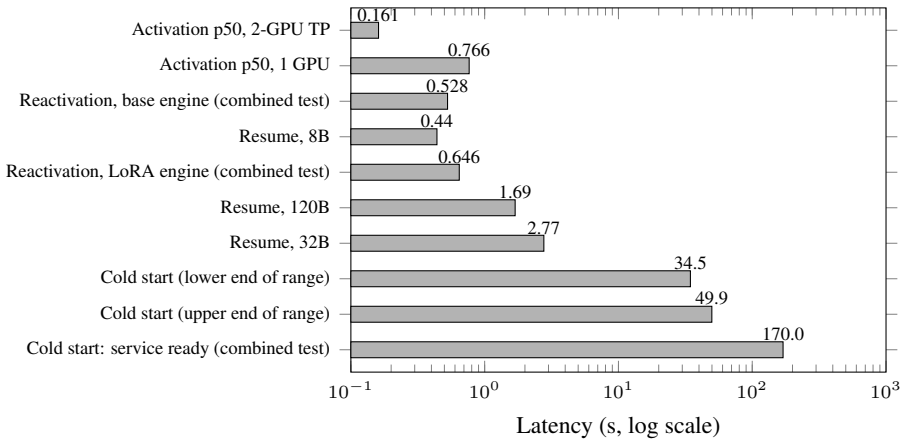

Figure 1: All latencies reported in this paper for bringing a model into service, on one log axis. Bars are drawn directly from Tables 1, 2, and 3, configurations differ between bars (see the tables), so bars are not directly paired.

4 Results

4.1 Model activation latency

Table 1 compares the time for a GPU to begin serving a different model against a cold start. At p50, activation takes 0.161 s on a 2-GPU tensor-parallel group and 0.766 s on a single GPU, against 34.5–49.9 s cold start, a 45–300× reduction. Fidelity is exact: the activated model's tensors are bit-exact with the reference and its outputs are deterministic. Figure 1 places these numbers alongside the suspend/resume and combined-system results on a common log scale.

4.2 Suspending and resuming idle tenants

Activation addresses moving a model onto a GPU, suspend/resume addresses tenants that are idle but must remain quickly available. Table 2 reports the results. While suspended, a tenant occupies ~1 GB of GPU memory. Resume latency is 0.44 s for an 8B model, 2.77 s for a 32B model, and 1.69 s for a 120B model, resume latency was not monotonic in model size in our measurements, and we report it as measured. All 25 of 25 post-resume outputs matched the reference.

Combined system test. We also ran one server with memory isolation, idle release, and harvesting enabled together (one GPU, Qwen3-8B base plus one LoRA tenant), Table 3 summarizes it. The service became ready from cold in 170.0 s. On idle release, GPU memory dropped from 66,527 MiB to 4,529 MiB (61,998 MiB, ~60.5 GiB freed) and returned to 65,381 MiB after reactivation. Reactivation took 0.528 s for the base engine and 0.646 s for the LoRA engine (~260–320× faster than the 170 s cold start), and greedy outputs after reactivation matched pre-idle outputs exactly (2/2). The resident footprint in this configuration (4,529 MiB) is larger than the ~1 GB measured in the isolated suspend/resume experiment. We did not attribute the difference and report both.

Table 2: Idle-tenant suspend/resume (measured).

Model size	Resume latency	Notes
8B	0.44 s	~1 GB resident while suspended, 25/25 outputs match reference
32B	2.77 s	
120B	1.69 s	

Table 3: Combined system test: memory isolation + idle release + harvesting on one server (one GPU, Qwen3-8B base + one LoRA tenant, measured). Errors: 0.

Quantity	Measured
Service ready (cold)	170.0 s
GPU memory: active → idle	66,527 MiB → 4,529 MiB (61,998 MiB, ~60.5 GiB freed)
GPU memory after reactivation	65,381 MiB
Reactivation latency (base / LoRA engine)	0.528 s / 0.646 s (~260–320× faster than cold)
Output match after reactivation (greedy)	2/2 exact
Harvesting, light	12 batch items completed, 0 failed, 6,144 output tokens
Harvesting, 64 batch requests active	interactive request HTTP 200 in 0.501 s
preemptions / resumptions	64 / 64
batch outcome	64 completed, 0 failures, 16,384 output tokens

4.3 Fleet consolidation at SLO

Table 4 is the headline result. Against 32 dedicated GPUs, Nodus serves the same 32 tenants and the same replayed traffic on 4 GPUs (8× fewer) in all three seeds. The dedicated arm reaches 100% SLO attainment with 0 errors, Nodus reaches a lowest per-class SLO attainment of 99.7%, 99.0%, and 98.5%, all above the 95% gate, with 0 errors out of 7,878, 7,929, and 7,983 requests. Relative to dedicated, Nodus delivers 7.6× / 8.0× / 8.0× more SLO-met requests per GPU-second and 7.8× / 8.1× / 8.1× lower cost per SLO-met token. With 2 GPUs, Nodus fails the SLO gate for this traffic, 4 GPUs is the measured floor.

Re-assignment is frequent, and cheap. During the consolidated runs, Nodus performed 203 (seed 1) and 176 (seed 2) model-weight re-assignment events per 15 minutes, with p95 latency of 0.39 s in each run. These events were what allowed 4 GPUs to follow the demand of 32 tenants: the mapping from GPUs to models was not static but revised continually as traffic shifted. Had each of those events instead cost a cold start (34.5–49.9 s in our measurements), every re-assignment would have stalled the affected tenant's requests for tens of seconds, far outside a TTFT target.

Latency cost of consolidation. Consolidation is not free in latency. Per-seed TTFT p95 for Nodus is 0.21–0.32 s and TPOT p95 is 31–33 ms. Both are within the per-class targets as reflected in the SLO attainments, but the 2× traffic experiment below shows directly that consolidated latencies are higher than dedicated ones.

4.4 Self-planned layout at 2× traffic

We replayed the same 32 tenants at 2× the headline rate (seed 0) and let Nodus plan its own layout. Table 5 reports the result. Nodus used 5 GPUs against 32 dedicated (6.4× fewer), met SLO in every class (lowest 0.9983, full-duration 0.9998) with 0 errors in both arms, and delivered 3.318667 versus 0.518854 SLO-met requests per GPU-second (6.396×) at $2.6016 versus $16.6884 per million SLO-met tokens (cost ratio 0.155893, 6.4× lower). Latency is higher than in the dedicated arm at every reported percentile (e.g., TTFT p95 0.1894 s vs. 0.0623 s, TPOT p95 0.0356 s vs. 0.0085 s), while remaining within targets. This is a single-seed result. The 3×-traffic arm has not yet been run.

4.5 Supporting component results

Consolidation relied on more than activation. We summarize the other measured capabilities in Table 6, again describing outcomes only.

Table 4: Fleet consolidation on one eight-GPU node: 32 tenants, 8 req/s aggregate replayed traffic, 15-min runs, three seeds (measured). Dedicated: one GPU per tenant, 100% SLO, 0 errors. ": ": not reported for that arm.

Seed	Arm	GPUs	SLO-met req/GPU-s	$/M SLO-met tokens	Lowest per-class SLO	TTFT p95	TPOT p95	Errors
0	Dedicated	32	0.271	31.92	100%	n/a	n/a	0
	Nodus	4	2.05	4.10	99.7%	0.21 s	31 ms	0/7,878
1	Dedicated	32	0.262	33.05	100%	n/a	n/a	0
	Nodus	4	2.09	4.09	99.0%	0.32 s	32 ms	0/7,929
2	Dedicated	32	0.264	32.80	100%	n/a	n/a	0
	Nodus	4	2.109	4.07	98.5%	0.31 s	33 ms	0/7,983

Nodus vs. dedicated: GPUs 8× fewer (all seeds), SLO-met req/GPU-s 7.6× / 8.0× / 8.0×, cost per SLO-met token 7.8× / 8.1× / 8.1× lower (seeds 0 / 1 / 2).

Table 5: Self-planned layout at 2× the headline traffic (eight GPUs, seed 0, 32 tenants, measured). Per-class SLO for classes A/B/L/X.

Metric	Nodus (5 GPUs)	Dedicated (32 GPUs)
Per-class SLO attainment A/B/L/X	1.0000/0.9983/1.0000/1.0000	all 1.0000
Full-duration SLO attainment	0.9998	n/a
Errors	0	0
SLO-met requests per GPU-second	3.318667	0.518854
$/M SLO-met tokens	2.6016	16.6884
TTFT p50/p95/p99 (s)	0.1086/0.1894/0.4722	0.0395/0.0623/0.1047
TPOT p50/p95/p99 (s)	0.0263/0.0356/0.0400	0.0078/0.0085/0.0236

Nodus vs. dedicated: 6.4× fewer GPUs, 6.396× SLO-met req/GPU-s, cost ratio 0.155893 (6.4× lower).

Shared serving versus hard partitions. With 2 tenants on 1 GPU, shared serving in Nodus achieves 1.36–1.41× the throughput of MIG partitioning [NVIDIA, n.d.a] and 1.43–1.46× that of MPS [NVIDIA, n.d.b] with a 50/50 split.

GPU memory isolation for arbitrary CUDA jobs. Nodus enforced exact memory caps across 8 CUDA allocation APIs. The open-source comparator HAMi-core [Project HAMi, n.d.] leaks on 2 of them. Overhead is −0.5% to +0.3%. VLLM [Kwon et al., 2023] running under caps shows 0.3% overhead and an unchanged 11.4 ms TPOT p95.

Memory overcommit with active jobs. On a single GPU, Nodus admits 1.5–2× the jobs of strict packing while keeping useful throughput at ≥0.90 of solo and 0 OOM, unmanaged overcommit collapses to 0.03. In detail: on a 4-job mix, strict packing admits 2 (aggregate useful throughput 0.934, 2 runs) while Nodus admits 4 (0.942–0.960), on a 3-job mix, strict admits 2 (0.961) and Nodus admits 3 (0.959–0.963). Valid arms had 0 out-of-memory errors or crashes. Unmanaged overcommit reached 0.028–0.029 throughput with 0/4 jobs completed.

Preemptible batch harvesting. Reactive harvesting added +2.03M batch tokens per GPU-hour at a cost of +26–30% interactive TTFT p95. With perfect traffic foresight (an oracle, not a deployable policy), GPU utilization rose from 51.5% to 93.1% at +7–9% TTFT p95. This is an upper bound on what better prediction could buy, not a result we achieve. The combined system test (Table 3) shows harvesting coexisting with interactive traffic: with 64 batch requests active, an interactive request returned in 0.501 s, and all 64 preempted batch items resumed and completed.

5 Related Work

Fast model loading and autoscaling. ServerlessLLM [Fu et al., 2024] targets low-latency serverless LLM inference by reducing model loading time and using locality-aware scheduling. BlitzS-

Table 6: Supporting component results (measured).

Capability	Setup	Result
Shared serving vs. hard partitions	2 tenants, 1 GPU	1.36–1.41× throughput vs. MIG, 1.43–1.46× vs. MPS 50/50
Memory caps for arbitrary CUDA jobs	:	exact caps across 8 CUDA allocation APIs (HAMi-core leaks on 2), overhead −0.5% to +0.3%, vLLM: 0.3% overhead, TPOT p95 unchanged at 11.4 ms
Memory overcommit, 4-job mix	a single GPU	strict: 2 admitted (0.934, 2 runs), Nodus: 4 admitted (0.942–0.960)
Memory overcommit, 3-job mix	a single GPU	strict: 2 admitted (0.961), Nodus: 3 admitted (0.959–0.963)
Unmanaged overcommit	a single GPU	0.028–0.029 throughput, 0/4 jobs completed
Batch harvesting, reactive	:	+2.03M batch tokens/GPU-h, +26–30% TTFT p95
Batch harvesting, oracle	:	utilization 51.5% → 93.1%, +7–9% TTFT p95

cale [Zhang et al., 2024] targets fast, live autoscaling of large models. vLLM's sleep mode [vLLM Project, 2025] lets an engine release GPU memory while idle and wake later. Nodus shares the goal of making a model available quickly on a GPU. Our emphasis is on using activation as a routine, high-frequency scheduling action for multi-tenant consolidation, and on measuring fidelity (bit-exact tensors, matching outputs) alongside latency.

Multi-model serving and multiplexing. Clockwork [Gujarati et al., 2020] achieves predictable DNN serving latency by controlling execution centrally. AlpaServe [Li et al., 2023] uses model parallelism to statistically multiplex multiple models. MuxServe [Duan et al., 2024] applies spatial-temporal multiplexing to serve multiple LLMs. Aegaeon [Xiang et al., 2025] pools GPUs for concurrent LLM serving and reports an ~82% GPU reduction in production. DeepSpeed-MII [Microsoft DeepSpeed Team, 2022] provides low-latency serving of many models. Shepherd [Zhang et al., 2023] studies DNN serving under unpredictable workloads. Our work is complementary to this line: we contribute measured activation and suspend/resume latencies and a consolidation evaluation against dedicated deployment on a mixed tenant population.

Adapter serving. S-LoRA [Sheng et al., 2024], Punica [Chen et al., 2024], and dLoRA [Wu et al., 2024] serve many LoRA adapters on a shared base model. Our workload includes such tenants but also full fine-tunes, separate-architecture models, and an FP8 32B model, which do not share base weights.

Scheduling, migration, and preemptible capacity. Llumnix [Sun et al., 2024] reschedules requests across LLM instances via live migration. SpotServe [Miao et al., 2024] serves generative LLMs on preemptible instances.

GPU sharing and isolation. MIG [NVIDIA, n.d.a] and MPS [NVIDIA, n.d.b] are vendor mechanisms for partitioning and sharing GPUs. We compare against both. HAMi [Project HAMi, n.d.] provides GPU virtualization in container environments. We compare against its HAMi-core component for memory caps. TGS [Wu et al., 2023], Tally [Zhao et al., 2025], and LithOS [Coppock et al., 2025] address transparent GPU sharing and performance isolation for concurrent workloads.

6 Conclusion

Changing which model a GPU serves need not cost a cold start. Nodus activated a model in 0.161–0.766 s at p50 with bit-exact fidelity, suspended idle tenants to a small footprint and resumed them in 0.44–2.77 s, and used these operations 176–203 times per 15 minutes at p95 0.39 s to serve 32 tenants on 4 GPUs instead of 32 at SLO, with 7.8–8.1× lower cost per SLO-met token. Yet the prototype did not make whole-fleet cold start fast (1.67×, target missed), did not guarantee compute shares, and has so far been evaluated on one node with synthetic tenants and replayed traffic. Extending the evaluation to multiple nodes, production traffic, more seeds at higher load, and a direct ablation of re-assignment are the most important next steps.

References

Lequn Chen et al. Punica: Multi-tenant LoRA serving. In *Conference on Machine Learning and Systems (MLSys)*, 2024.

Patrick H. Coppock et al. LithOS: An operating system for efficient machine learning on GPUs. In *31st ACM Symposium on Operating Systems Principles (SOSP)*, 2025.

Jiangfei Duan et al. MuxServe: Flexible spatial-temporal multiplexing for multiple LLM serving. In *International Conference on Machine Learning (ICML)*, 2024.

Yao Fu et al. ServerlessLLM: Low-latency serverless inference for large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024.

Arpan Gujarati et al. Serving DNNs like Clockwork: Performance predictability from the bottom up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.

Woosuk Kwon et al. Efficient memory management for large language model serving with PagedAttention. In *29th ACM Symposium on Operating Systems Principles (SOSP)*, 2023.

Zhuohan Li et al. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2023.

Xupeng Miao et al. SpotServe: Serving generative large language models on preemptible instances. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

Microsoft DeepSpeed Team. DeepSpeed-MII. `https://github.com/microsoft/DeepSpeed-MII`, 2022.

NVIDIA. NVIDIA multi-instance GPU user guide. `https://docs.nvidia.com/datacenter/tesla/mig-user-guide/`, n.d.a.

NVIDIA. Multi-process service. `https://docs.nvidia.com/deploy/mps/`, n.d.b.

Project HAMi. HAMi: Heterogeneous AI computing virtualization middleware. `https://github.com/Project-HAMi/HAMi`, n.d.

Ying Sheng et al. S-LoRA: Serving thousands of concurrent LoRA adapters. In *Conference on Machine Learning and Systems (MLSys)*, 2024.

Biao Sun et al. Llumnix: Dynamic scheduling for large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024.

vLLM Project. Sleep mode. vLLM documentation, `https://docs.vllm.ai`, 2025.

Bingyang Wu et al. Transparent GPU sharing in container clouds for deep learning workloads. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2023.

Bingyang Wu et al. dLoRA: Dynamically orchestrating requests and adapters for LoRA LLM serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024.

Yuxing Xiang et al. Aegaeon: Effective GPU pooling for concurrent LLM serving on the market. In *31st ACM Symposium on Operating Systems Principles (SOSP)*, 2025.

Dingyan Zhang et al. BlitzScale: Fast and live large model autoscaling with O(1) host caching. arXiv preprint, 2024.

Hong Zhang et al. SHEPHERD: Serving DNNs in the wild. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2023.

Wei Zhao et al. Tally: Non-intrusive performance isolation for concurrent deep learning workloads. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2025.