One GPU, More Work.

Nodus Compute's GPU Partitioning vs. Native NVIDIA MPS

Pranav Varshney, Viswa Kotra
October 30, 2026

Abstract

GPU multiplexing is usually treated as a low-level concurrency mechanism. We evaluate a higher-level alternative: resident, software-defined GPU partitions with explicit tenant memory budgets and coordinated serving. On a single datacenter GPU, using Qwen3-1.7B-FP8 and an identical deterministic two-tenant trace, Nodus produced 15.0% and 15.1% more completed output tokens per second than native NVIDIA MPS at 12 and 16 requests per second, respectively. Nodus also reduced p99 time to first token by 15.0% and 16.6%. The evidence supports a narrow result for this workload and a broader systems thesis: the useful abstraction is a schedulable GPU partition, not merely a CUDA process-sharing mechanism.

1 Introduction

A single accelerator increasingly serves multiple models, teams, or tenants. NVIDIA Multi-Process Service (MPS) enables concurrent CUDA execution, but operators must still compose memory enforcement, admission, lifecycle, telemetry, and serving policy. Nodus instead treats each partition as an infrastructure object with an explicit size, tenant identity, engine, and control-plane contract.

We ask a simple experimental question: under the same GPU, model, memory budgets, engine configuration, and request trace, does this software-defined arrangement deliver more useful inference work than native MPS?

Across both tested loads, it does. The result is consistent in throughput and tail latency, while remaining bounded to one seed, GPU, model, traffic family, and two-tenant configuration.

2 Experimental setup

Both arms ran on a single datacenter GPU with Qwen/Qwen3-1.7B-FP8 and vLLM v0.30.0. Each used two independent tenant engines, 18 GiB hard caps per tenant, a 16.2 GiB vLLM target per engine, 8,192 maximum batched tokens, 64 maximum sequences, and prefix caching disabled.

The native arm ran both clients under MPS with a 100% active-thread cap per client. The Nodus arm ran two resident software partitions. Inputs ranged from 1,023 to 1,026 tokens. Every request produced exactly 256 output tokens. Each offered-load window lasted 120 seconds and then drained fully.

$$SLO = (TTFT \le 1\ s) \wedge (\text{mean } TPOT \le 100\ ms)$$

We report completed output throughput and SLO-qualified output throughput. The latter prevents a large post-window drain from appearing healthy when request latency has collapsed.

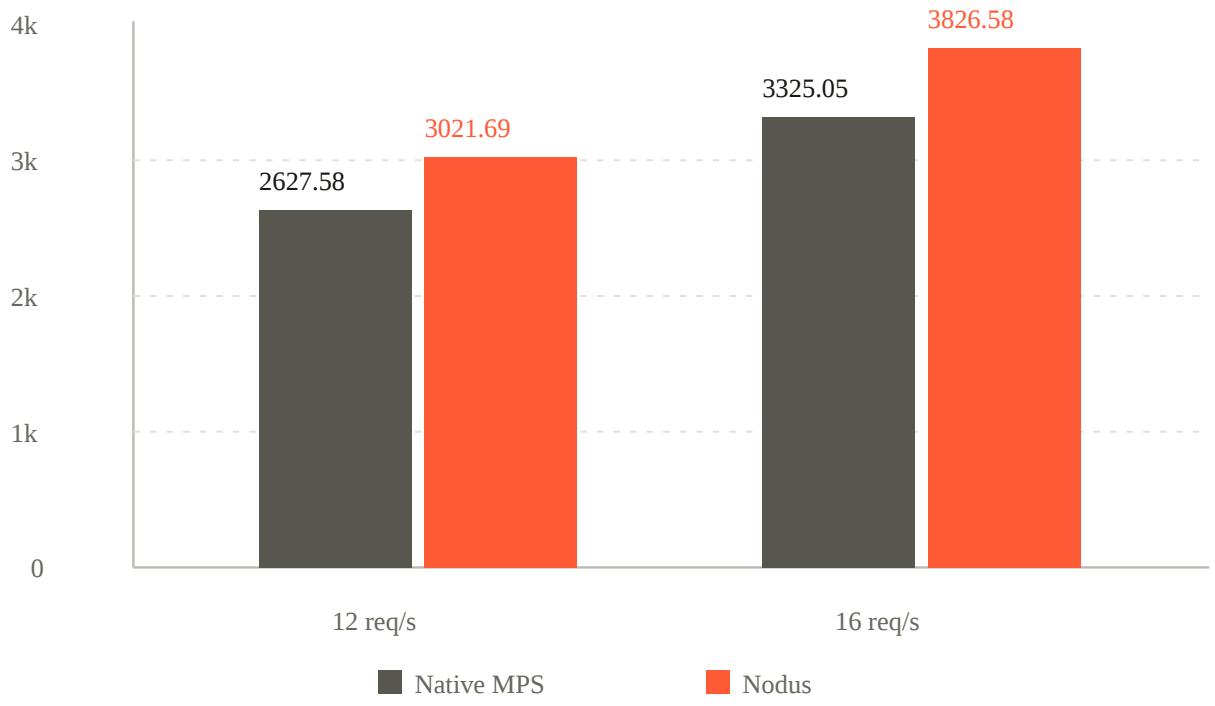

Figure 1. Completed output tokens per second. Nodus improves throughput by 15.0% at 12 req/s and 15.1% at 16 req/s.

3 Results

At 12 requests per second, both systems completed all 1,440 offered requests. Nodus increased completed output throughput from 2,627.58 to 3,021.69 tokens per second, reduced p99 TTFT from 81.2 to 69 milliseconds, and moved latency-target attainment from 86.96% to 100%.

At 16 requests per second, both systems completed all 1,920 requests after drain. Nodus increased completed output throughput from 3,325.05 to 3,826.58 tokens per second and SLO-qualified throughput from 1,212.26 to 1,421.02 tokens per second.

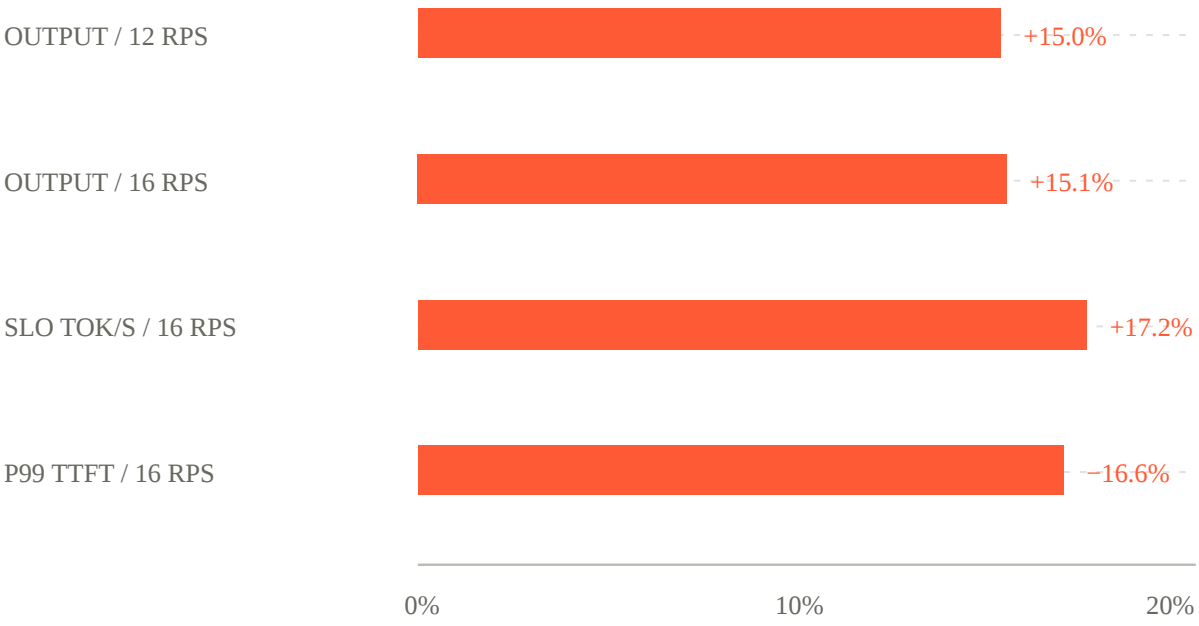

Figure 2. Relative improvement over the matched native MPS arm. For TTFT, a reduction is an improvement.

Load	System	Output tok/s	Within targets	p99 TTFT	SLO tok/s
12	MPS	2,627.58	86.96%	81.2 ms	2,627.58
12	Nodus	3,021.69	100.00%	69 ms	3,021.69
16	MPS	3,325.05	31.70%	6.104 s	1,212.26
16	Nodus	3,826.58	37.14%	5.091 s	1,421.02

Table 1. Matched benchmark results. Orange rows mark the Nodus arm.

Stress-regime caveat. At 16 requests per second, both systems missed the experiment's 99% latency-attainment target. Nodus behaved better past the knee. Neither configuration met the target.

4 System interpretation

Mean SM activity rose from 78.78% to 91.0% at 12 req/s and from 82.78% to 95.2% at 16 req/s. In these runs, higher activity became useful inference work rather than overhead.

The broader product thesis follows from the abstraction boundary. VMware made physical compute programmable through a resource abstraction and a control plane for placement, isolation, lifecycle, and operations. GPU infrastructure needs the equivalent: a software-defined partition with explicit memory, compute policy, tenant identity, lifecycle, telemetry, and a portable control-plane contract.

NVIDIA MIG remains an option for hardware-enforced fault and memory-bandwidth isolation on supported GPUs. This experiment did not include a MIG arm and does not claim to beat MIG. A unified control plane should place hardware and software mechanisms behind one operational model and choose the appropriate isolation tier.

5 Conclusion

Software-defined partitioning did more than reproduce native MPS behavior with a control plane. On this workload, it delivered more completed inference work and lower tail latency from the same GPU. The result motivates treating a GPU partition as a first-class infrastructure primitive.

6 Reproducibility statement

All reported values come from the machine-readable result files published with this report. The comparison fixes the physical GPU, model, tenant count, memory caps, engine limits, request trace, output length, offered window, and drain policy across both arms.

Artifacts

The current benchmark package is the source of truth for every figure, table, and claim in this report.

[1] MATCHED COMPARISON	JSON ↗
[2] NATIVE MPS	JSON ↗
[3] NODUS	JSON ↗
[4] SHA-256 MANIFEST	JSON ↗