GPU SMITH
COMPUTE ENGINEERING

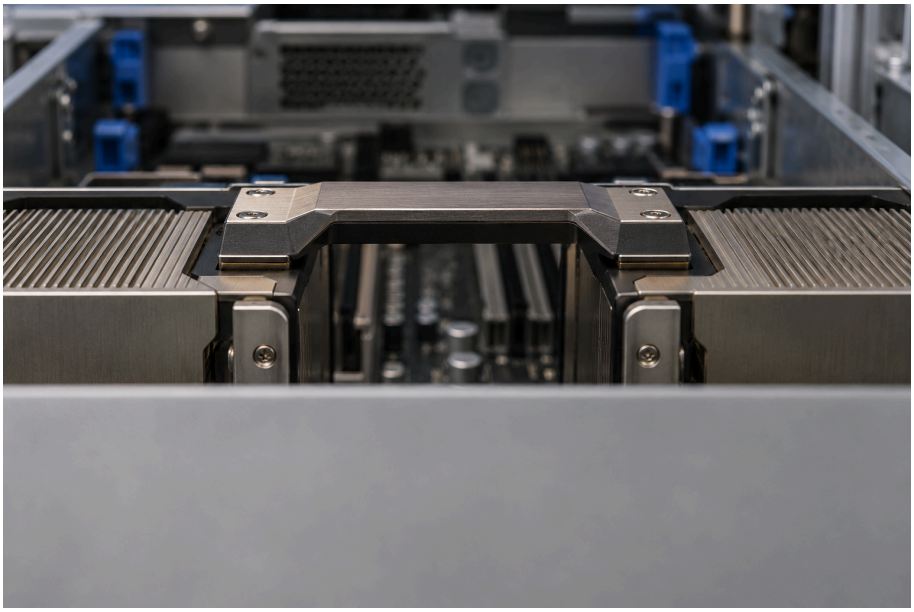
GPU SMITH / RESEARCH & PRACTICAL GUIDANCE

TensorRT Multi-Device in Dynamo-Triton: One Model, Many GPUs

> Plan and build private AI compute with GPU Smith. We connect workload requirements with GPU hardware, networking, deployment and operating decisions so your infrastructure fits the work it must perform.

Published by GPU Smith · 2026-11-13 · tensorrt multi-device · dynamo-triton · triton inference server · multi-gpu inference · kind_model · nccl · tensor parallelism · context parallelism · tensorrt 11 · gpu cost per request

Original article: https://gpusmith.com/articles/en/tensorrt-multi-device-dynamo-triton-multiple-gpus



In this article

1. Executive Summary
2. Introduction and Background
3. Definitions and Taxonomy: What Multi-Device Means Here
4. The Feature and Support Boundary
5. Implementation Guidance: An Adoption Gate
6. Data Analysis and Evidence
7. Implications and Future Directions
8. Frequently Asked Questions (FAQs)
9. Conclusion

Executive Summary

TensorRT multi-device inference reached the Triton Inference Server TensorRT backend with release 26.07, which lets one TensorRT network run across several GPUs behind a single model endpoint (docs.nvidia.com). NVIDIA now brands the server **Dynamo-Triton**, formerly Triton Inference Server (developer.nvidia.com). The mechanism is narrow and explicit: a single **KIND_MODEL** instance owns N GPUs (github.com), the engine must be sharded offline before Triton ever sees it, and collectives run through NCCL, NVIDIA's inter-GPU communication library (docs.nvidia.com). Triton configuration activates a distributed plan; it does not create one.

The headline result is NVIDIA's Cosmos 3 Nano video-generation demonstration, where end-to-end latency fell from **156.595 seconds on one GPU to 34.183 seconds on eight**, a 4.58x speedup (developer.nvidia.com). That result is real but narrow: it covers one generative-video pipeline, excludes model loading and MP4 encoding, and does not measure concurrent throughput or cost per video (see Data Analysis and Evidence). Using NVIDIA's own times, GPU-seconds per completed request rise from **156.6 on one GPU to 273.5 on eight (1.75x)**, so each unit of latency saved is paid for in extra GPU time. NVIDIA's backend guide also documents the opposite regime: single attention layers at batch 1 on two B200 GPUs are overhead and latency bound, and the speedup is lost in run-to-run noise.

The hardware boundary matters. Distributed collectives require an NVIDIA Ampere (SM 80) or later GPU (docs.nvidia.com), while the dedicated multi-device attention operator requires Blackwell (SM 100) or later (docs.nvidia.com). The Cosmos demonstration did not need the Blackwell operator, because its Ulysses implementation wraps standard attention in collective layers.

For platform teams the decision is not "is multi-device faster" but "is the latency saved worth the GPU-seconds, on the buyer's own model at the buyer's own concurrency". This report supplies a seven-step adoption gate: pick a memory-bound or latency-bound candidate, fix a single-GPU baseline, build a sharded plan, configure KIND_MODEL with an explicit GPU list, validate output and failure behavior, benchmark identical work at 1, 2, 4 and 8 GPUs across concurrency levels, and record a go or no-go decision with a rollback to single-GPU instances or more replicas. The rollback is cheap because the single-GPU path stays the default (github.com). Figures are as of November 2026 and should be re-checked against the current Triton and TensorRT release notes before any procurement decision.

Introduction and Background

Enterprise inference teams usually scale a model in one of two ways. They add **replicas**, each holding a full copy of the model on one GPU, which raises throughput without changing the latency of a single request. Or they split one request across GPUs, which can lower latency or fit a model that does not fit in one GPU's memory. TensorRT's documentation names these two motivations directly: multi-device inference applies when models exceed single-GPU memory or when parallel execution reduces latency (docs.nvidia.com). Until 2026, using the second approach through Triton typically meant leaving TensorRT or building a custom serving layer; release 26.07 changed that for TensorRT engines.

This report is written from the perspective of an independent advisor. GPU Smith is an independent private AI infrastructure engineering consultancy; it does not manufacture accelerators, resell Triton or operate a cloud, and nothing here is a benchmark of its own. All measurements discussed are NVIDIA's, quoted with their stated scope, and every derived figure is labeled as arithmetic on those published numbers. No independent third-party benchmark of TensorRT multi-device in Dynamo-Triton was found during research in November 2026, which is itself a finding: buyers should treat vendor numbers as a hypothesis to reproduce, not a forecast.

The report answers the main query and these related questions:

- **Triton and TensorRT on several GPUs:** how Triton Inference Server runs TensorRT models across more than one GPU.
- **One model, many GPUs:** how to run one model on multiple GPUs in Triton.
- **Naming:** what Dynamo-Triton is relative to Triton Inference Server.
- **Mechanism:** how TensorRT multi-device inference works.
- **Parallelism:** how tensor parallelism differs from data parallelism.
- **Configuration:** what `instance_group` does with multiple GPUs.
- **Decision:** when splitting a model is justified.

The distinctive angle is an **adoption gate** built on model fit and GPU opportunity cost: the question is what else the extra GPUs could have served while one request occupied them.

The Cosmos 3 Nano demonstration is generative video, and NVIDIA's model card describes Cosmos3-Nano as a 16B-parameter model (huggingface.co). Nothing in the demonstration shows that a large language model, a recommender or a vision classifier will speed up in the same proportion, and this report does not generalize the result.

Definitions and Taxonomy: What Multi-Device Means Here

Naming and the exact stack

Dynamo-Triton is NVIDIA's name for the product formerly called Triton Inference Server, which deploys models across major frameworks. NVIDIA's pages describe the relationship to **NVIDIA Dynamo** differently. The Dynamo-Triton page says NVIDIA also offers Dynamo, designed for LLM inference and multi-node deployment (the page's own wording reads "multi-mode deployment", apparently a typo) (developer.nvidia.com), while the Dynamo page calls Dynamo the successor to Triton (developer.nvidia.com). For this article the practical reading is that Dynamo-Triton is the Triton lineage that ships the TensorRT backend feature discussed here, and Dynamo is a separate product aimed at LLM serving. The 26.07 release notes, which list the multi-device feature (docs.nvidia.com), still carry the title "Triton Inference Server Release 26.07".

The 26.07 container carries Triton 2.71.0, TensorRT 11.1.0.106, NCCL 2.30.7, cuDNN 9.25.0.20 and a CUDA 13.3.4.1 toolkit (docs.nvidia.com). The release notes are internally inconsistent about CUDA, because one section says the release is based on CUDA 13.3.1; teams pinning drivers should confirm the CUDA level from

the container itself. TensorRT's online documentation is versioned separately from the TensorRT build inside a Triton container, so a capability described in the latest TensorRT pages may not exist in the image you deploy.

Replicas, tensor parallelism and context parallelism

Three patterns are easy to confuse. **Data parallelism**, or replication, keeps a full model per GPU; PyTorch's guidance for distributed data parallel is one process per model replica (pytorch.org). **Tensor parallelism** splits individual layers' weights across GPUs, which requires a collective such as all-reduce inside each layer; Megatron-LM's intra-layer model-parallel design (arxiv.org) needs two all-reduces in the forward path of a transformer layer (arxiv.org). **Context (sequence) parallelism** splits the token sequence instead. DeepSpeed-Ulysses partitions input along the sequence dimension and uses an all-to-all collective for attention (arxiv.org), so that each GPU receives the full sequence for a disjoint subset of attention heads (arxiv.org).

TensorRT supports building blocks for the second and third patterns. Its multi-device attention layers use context parallelism, splitting the key-value sequence across GPUs (docs.nvidia.com). NVIDIA's backend guide also describes a weight-sharded tensor-parallel mode that gives roughly 1/N per-GPU weight memory (github.com), which is the relevant mode when the weights themselves do not fit on one GPU.

The distinction decides the business case. Replicas and splits are different answers to different constraints. If the model fits and traffic is parallel, replicas win on GPU-seconds; PyTorch's guidance is that a model too large for one GPU must use model parallelism (pytorch.org). If a single request is too slow or too big, a split is the only lever that changes per-request latency or capacity, and it costs efficiency; even Megatron-LM's well-optimized training setup reports 76% scaling efficiency against a single-GPU baseline (arxiv.org).

Triton's instance model: KIND_GPU versus KIND_MODEL

For a GPU-kind `instance_group`, `count` is the number of instances created for each listed GPU (github.com). That is the replica path: independent instances, each on its own GPU, serving different requests in parallel. **KIND_MODEL** is different. In the TensorRT backend, a single instance owns N GPUs, and the backend rather than the server manages the devices. This is the answer to the question of how `instance_group` behaves with several GPUs: KIND_GPU multiplies instances, while KIND_MODEL with a GPU list makes one instance span them.

The Feature and Support Boundary

Table 1 below consolidates what NVIDIA documents as the support boundary. It is a reading of vendor documentation as of November 2026, not a tested compatibility matrix.

Area	Documented requirement or limit	Practical reading
TensorRT runtime	Multi-device is generally available from TensorRT 11.0 (github.com)	Use the TensorRT 11 line; confirm the version inside your Triton image
Triton	Backend support added in release 26.07	Older images lack the path
Collectives	Distributed collectives run through NCCL on Ampere (SM 80) or later	A100, H100 and Blackwell qualify; Turing does not

Area	Documented requirement or limit	Practical reading
Multi-device attention	Blackwell (SM 100) or later	The dedicated context-parallel attention operator is a Blackwell feature
Optimization profiles	One profile per multi-device model (github.com)	Dynamic-shape services must fix the shapes they serve
Build flag	Backend built with TRITON_ENABLE_TENSORRT_MULTI_DEVICE=ON (github.com)	Verify in your image; stock container build options were not confirmed from public docs

Before reading further, a team should be able to answer five pre-flight questions:

- **Hardware:** Are the GPUs Ampere (SM 80) or later, and Blackwell (SM 100) or later if the attention operator is needed?
- **Stack:** Does the deployed container carry TensorRT 11 and a Triton build with the multi-device path?
- **Model:** Is there a collective-based or weight-sharded recipe for this network?
- **Shapes:** Can the service live with one fixed optimization profile?
- **Fabric:** Do the chosen GPUs share an NVLink or NVSwitch domain?

The most important reading of *Table 1* is the split between Ampere-or-newer collectives and Blackwell-only attention. A team on H100 or A100 hardware can run distributed networks built from collective layers, but cannot use the dedicated multi-device attention operator. The Cosmos demonstration is an example of the first path: NVIDIA states that Ulysses was implemented with distributed-collective layers around standard attention (developer.nvidia.com). Whether your own network has a collective-based sharding recipe is a model-engineering question, not a deployment flag.

The offline sharded engine requirement

Multi-device is not a runtime switch. NVIDIA's guide says the engine is sharded offline, and the collectives live inside the engine (github.com). Configuration alone cannot parallelize an existing plan. Two consequences follow. First, a plan built for N ranks is a different artifact from the single-GPU plan, so each GPU count you intend to benchmark (2, 4, 8) needs its own build. Second, TensorRT's general compatibility rule applies: serialized engines are only guaranteed to work correctly with the same OS, CPU architecture, GPU model and TensorRT version used to build them (docs.nvidia.com). The multi-device pages reviewed did not state a separate portability rule for sharded plans, so treat each plan as pinned to the GPU model and TensorRT build on which it was created until your own tests show otherwise.

One KIND_MODEL instance, a fixed GPU list

The Triton backend documents a configuration with `instance_group` kind KIND_MODEL and `count: 1`, plus model parameters `enable_multi_device` set to the string "true" and `multi_device_gpus` listing GPU indices such as "0,1,2,3,4,5,6,7" (docs.nvidia.com). The single-GPU KIND_GPU path is unchanged and remains the default, so rollback means pointing back at the original model directory. The guide documents only `count: 1`; behavior for a higher count, a GPU missing from the list, or a failed rank is not described in the pages reviewed and must be tested rather than assumed.

NCCL, interconnect and failure behavior

All ranks must call execution concurrently, because collectives block until every rank participates, and if any rank fails to issue its call the others hang indefinitely (docs.nvidia.com). Interconnect bandwidth is also not uniform: NVIDIA lists 600 GB/s NVLink for A100 (www.nvidia.com), 900 GB/s for H100 (www.nvidia.com) and 1.8 TB/s per GPU for GB200 (www.nvidia.com), with a sixth generation at 3 TB/s per GPU announced for the Vera Rubin platform (www.nvidia.com). For comparison, the PCI-SIG announcement for PCIe 5.0 cites up to 128 GB/s over an x16 link (us.design-reuse.com). Even with NVSwitch, NVIDIA notes that Amdahl's law limits the speedup of each added GPU when communication overhead is high (developer.nvidia.com).

For diagnosis, set `NCCL_DEBUG` during bring-up and dump the detected NCCL topology to a file with `NCCL_TOPO_DUMP_FILE` in every benchmark run. That lets reviewers see whether a four-GPU list happened to span one NVLink domain or fell back to PCIe.

Implementation Guidance: An Adoption Gate

The gate follows seven steps. Steps 1 and 2 decide whether to start, steps 3 to 5 build and validate, step 6 measures, and step 7 decides.

Steps 1 and 2: choose candidates and fix the baseline

- **Prefer memory-bound or latency-bound candidates.** NVIDIA names exceeding single-GPU memory and latency reduction as the two use cases.
- **Require shardability.** The model needs a sharded recipe, either collective-based context parallelism or weight sharding for tensor parallelism; if none exists, stop.
- **Fix one baseline.** Use the same model, shapes and precision on a single GPU with an ordinary KIND_GPU instance.
- **Record the serial fraction.** Measure how much of a request sits outside the model call (preprocessing, scheduling, decoding, encoding). This caps the benefit: even small serial percentages severely penalize parallel performance (cvw.cac.cornell.edu).

Steps 3 and 4: create the plan and configure the instance

Build a separate sharded plan for each GPU count and keep the single-GPU plan untouched. In config.pbtxt use KIND_MODEL with `count: 1`, set `enable_multi_device` and list the GPUs explicitly. Pin the list to GPUs inside one NVLink domain. Because the backend supports one optimization profile for these models, fix the shapes you intend to serve. Confirm the multi-device path exists in your container by loading a small sharded test model, since the guide says the backend must be built with the multi-device flag.

Step 5: validate output and failure behavior

Distributed execution changes the order of floating-point reductions, so outputs can differ numerically from the single-GPU result. Define the acceptance metric before benchmarking: for a classifier, label agreement and a tolerance on logits; for generation, a task metric or a human-reviewed sample. Then test failure paths deliberately, because a rank that never calls execution hangs the rest. The minimum set is:

- **Rank loss:** stop one rank's worker during a request and record what the client sees.
- **Wrong GPU list:** load the model with a missing or duplicated GPU index.
- **Memory pressure:** run a request while another process holds GPU memory.
- **Restart:** time the full unload and reload, including NCCL communicator setup.
- **Mixed load:** send single-GPU and multi-device models to the same node and watch for interference.

Record what the client sees and how long recovery takes in every case.

Step 6: benchmark identical work at 1, 2, 4 and 8 GPUs

Use a load generator such as Triton's Perf Analyzer, and sweep concurrency rather than testing a single stream. Collect GPU metrics per GPU and aggregate them yourself. Separate the five quantities in *Table 4* and compute **GPU-seconds per completed request** as GPU count multiplied by measured request time. For capacity decisions, always compare against N independent replicas of the baseline on the same N GPUs. For every run, record the following:

- **Build identity:** plan build ID, TensorRT version and container tag.
- **Placement:** the GPU list and the dumped NCCL topology file.
- **Load:** concurrency levels tested and request shapes.
- **Latency:** p50, p95 and p99 end-to-end, plus model-call time.
- **Throughput:** completed requests per second at each concurrency level.
- **Cost:** GPU-seconds per completed request.
- **Quality:** the pre-agreed task metric against the single-GPU output.
- **Failures:** the outcome of each failure test above.

Step 7: go, no-go and rollback

Table 2 turns measurements into decision rules. The thresholds are illustrative starting points for a review meeting, not NVIDIA guidance, and should be set from your own service-level objectives (SLOs).

Gate	Go condition (illustrative)	No-go or rollback trigger
Latency SLO	Single-GPU p95 misses the SLO and the candidate meets it with margin	Baseline already meets the SLO
Memory fit	Weights or activations exceed one GPU, so a split is required	Model fits and replicas can serve the load
Efficiency	Strong-scaling efficiency (speedup divided by GPU count) above a floor you set, such as 70%	Efficiency below the floor at the GPU count you need
GPU-seconds	Extra GPU-seconds per request is justified by the value of latency saved	Replicas meet the SLO at lower GPU-seconds
Quality	Task metric within the pre-agreed tolerance on a held-out set	Metric drifts beyond tolerance or varies across runs
Operations	Failed-rank behavior is bounded and restart is automated	Hang or unbounded recovery after a rank failure

Gate	Go condition (illustrative)	No-go or rollback trigger
Concurrency	Throughput at target concurrency meets demand within the same GPU budget	Throughput per GPU below replicas at target concurrency

Table 2 is deliberately conservative. A go decision requires several conditions to hold together, because a split that wins on latency can lose on cost, and a split that wins under light load can lose at peak concurrency. Rollback is cheap by design: restore the single-GPU model directory or add replicas.

Data Analysis and Evidence

NVIDIA's Cosmos 3 Nano results, with derived columns

NVIDIA's results come from one Cosmos 3 Nano pipeline on a single eight-GPU system, reported as means over measured generations. The blog, dated September 21, 2026, reports end-to-end latency falling from 156.595 seconds on one GPU to 34.183 seconds on eight (developer.nvidia.com). *Table 3* restates NVIDIA's reported end-to-end and transformer RPC (remote procedure call) times and adds the columns this report derived: speedup, efficiency (speedup divided by GPUs), GPU-seconds per request (GPUs times end-to-end time) and time outside the transformer call.

Config	GPUs	End-to-end (s)	E2E speedup / efficiency	Transformer RPC (s)	RPC speedup / efficiency	GPU-seconds per request	Time outside RPC (s)
SD (single device)	1	156.595	1.00x / 100%	146.192	1.00x / 100%	156.6 (1.00x)	10.40
CP2	2	87.999	1.78x / 89%	77.548	1.89x / 94%	176.0 (1.12x)	10.45
CP4	4	53.093	2.95x / 74%	42.661	3.43x / 86%	212.4 (1.36x)	10.43
CP8	8	34.183	4.58x / 57%	23.993	6.09x / 76%	273.5 (1.75x)	10.19

The time columns are NVIDIA's reported values for the same blog table; speedup, efficiency, GPU-seconds and time outside the RPC are arithmetic by this report. Three readings follow from *Table 3*. First, the transformer call scales far better than the whole request: the eight-GPU RPC speedup is 6.09x but end-to-end is 4.58x, because roughly 10 seconds of work outside the call does not shrink. That serial portion of about 10.4 seconds caps end-to-end speedup near 15x even with unlimited GPUs. Second, efficiency decays with GPU count (89%, 74%, 57% end-to-end), the expected shape for fixed-size problems under Amdahl's law, which bounds speedup at the reciprocal of the serial fraction (as restated in Gustafson's 1988 paper, which quotes Amdahl's 1967 argument rather than being its original source) (users.cs.utah.edu). Perfect strong scaling means N processors finish in 1/N of the time (cvw.cac.cornell.edu), and tutorials show how demanding high efficiency is: holding 90% efficiency on ten processors requires parallelizing 99% of the code (hpc-wiki.info). Third, GPU-seconds per request rise monotonically, so each added GPU buys latency at a premium.

What these results do not show

NVIDIA states that model loading and MP4 encoding were excluded from the timing (developer.nvidia.com) and that the benchmark does not measure concurrent request throughput, cost per generated video or total cost of ownership (developer.nvidia.com). The blog does not name the GPU model or interconnect for its eight-GPU system, so the result cannot be mapped to a specific procurement. The Cosmos model card lists Ampere, Blackwell and Hopper among its supported microarchitectures (huggingface.co) and describes context parallelism for multi-GPU speedups in the model's own inference stack (huggingface.co), which is a different serving path from the TensorRT plan benchmarked here.

An illustrative capacity calculation (Hypothetical Example) shows why concurrency matters. Suppose eight GPUs serve this pipeline. Eight independent single-GPU replicas, each taking 156.595 seconds per request, would complete 8 / 156.595 = 0.0511 requests per second if they ran without interference. One CP8 instance, completing a request every 34.183 seconds, delivers 0.0293 requests per second. In that hypothetical, replicas deliver 1.75x the throughput on the same GPUs, while CP8 delivers each individual request 4.58 times faster. Both figures are arithmetic on NVIDIA's single-request times, not measured concurrent runs.

To translate GPU-seconds into money, a second hypothetical applies a rate. Lambda's pricing page, retrieved November 13, 2026, listed a single H100 SXM at $4.89 per GPU-hour (lambda.ai) and an eight-GPU HGX B200 instance at $7.69 per GPU-hour (lambda.ai); the page states that these prices take effect on November 16, 2026, so they were not yet in force when retrieved; treat both as indicative only. The B200 figure is not used in the calculation below. NVIDIA did not say which GPU the demonstration used, so the H100 rate is a placeholder and every dollar figure that follows is illustrative arithmetic, not a cost estimate for NVIDIA's demonstration. At the H100 rate (Hypothetical Example), 156.6 GPU-seconds costs about $0.21 per request on one GPU and 273.5 GPU-seconds costs about $0.37 on eight, a premium of about $0.16 per request for a latency reduction of roughly 122 seconds. Whether that premium is worth paying is a business question the benchmark itself does not answer.

The opposite regime: shallow models at small batch

NVIDIA's backend guide reports a deliberately simple test: single self-attention layers (4096 hidden units, 16-bit floating point) on one versus two B200 GPUs over NVLink at batch 1. The guide's own interpretation is that single-attention-layer models at batch 1 are overhead and latency bound on B200 (github.com). At sequence length 65,536 the two-GPU latency (about 2240 ms) was no better than the single-GPU latency (about 2224 ms) (github.com). The guide describes its numbers as noisy run-to-run (about ±10%), and its 65,536-token row lists an observed speedup range of about 0.99 to 1.03x, with a cold run noted at about 2x. A stack of six attention blocks at sequence length 32,768 gave a consistent gain in the guide: speedups of 1.05x to 1.15x across four runs, about 1.1x on average. NVIDIA expects a full multi-layer transformer with larger batches to approach, but not necessarily reach, the ideal 2x on two GPUs; that is an expectation, not a measurement. The contrast with Cosmos is the point: speedup depends on how much parallel work each layer and request carries relative to collective cost.

Interpreting the metrics

Table 4 separates quantities that headline numbers tend to merge. It is the main tool for reading any vendor claim about multi-device serving, including those above.

Metric	What it tells you	Cosmos result	What it cannot tell you	How to measure
Transformer RPC time	Speed of the model call alone	146.192 s to 23.993 s (6.09x)	End-user latency; anything outside the call	Server-side compute time per request
End-to-end latency	Full single-request time through the pipeline	156.595 s to 34.183 s (4.58x)	Load behavior; steady-state throughput	Client-side time for one request on a warm, idle server
Throughput	Completed requests per second at saturation	Not measured by NVIDIA	Latency of any one request	Load generator at several concurrency levels
Concurrency	How latency degrades as requests overlap	Not measured by NVIDIA	Single-request speedup	Sweep concurrency; record p50 and p95
GPU-seconds per completed request	Cost of each answer in GPU time	Derived here: 1.00x to 1.75x	Whether the latency was worth it	GPU count times request time, summed over completed requests

The central lesson of *Table 4* is that RPC time, end-to-end latency and GPU-seconds move in different directions: the RPC speedup is the largest number, the end-to-end speedup is smaller, and GPU-seconds worsen. A report that quotes only the first is presenting the best case. Throughput and concurrency are exactly the two rows that NVIDIA's single-stream benchmark leaves empty, and they are the rows that decide whether replicas beat a split.

Implications and Future Directions

For platform owners the near-term effect is organizational as much as technical. A multi-GPU instance consumes N GPUs of scheduler capacity per request in flight, so it interacts with quotas, bin-packing and autoscaling. A fleet running a latency-critical model on eight-GPU nodes might dedicate a node to one CP8 instance or run eight replicas on the same node: the first optimizes lone-request latency, the second aggregate throughput. Interconnect becomes a procurement variable. With NVSwitch, every Hopper GPU in a server can talk to any other at 900 GB/s (developer.nvidia.com), and eight-GPU H100 cloud instances expose 640 GB of HBM3 per node (aws.amazon.com). Teams without a fast fabric should expect weaker scaling than the demonstration and should test GPU lists that stay inside one NVLink domain.

For model owners, sharding is an engineering project. The offline plan, the single optimization profile and the numerical-equivalence check mean each model adds a build and validation pipeline. Communication scaling is also model dependent: the Ulysses authors report that communication volume stays constant when sequence length and device count grow together (arxiv.org), which favors long-sequence workloads over short ones.

Before go-live, a platform team should confirm the following:

- **Capacity model:** how many N-GPU instances fit per node and per cluster.
- **Scheduling:** whether the scheduler can place N GPUs from one NVLink domain.
- **Alerting:** a rank-hang alert based on request-time outliers.
- **Versioning:** plan, TensorRT and NCCL versions recorded together.
- **Rollback:** the single-GPU model directory retained and tested.
- **Review date:** a re-benchmark whenever the container or driver changes.

Open questions remain practical. Public documentation does not describe KIND_MODEL counts above one or failure recovery, so operations teams should write those tests themselves. Documentation and container versions differ, so features in the newest TensorRT pages may need a newer container. Buyers who want an independent second opinion on workload sizing, interconnect choice and the economics of splitting versus replicating can consult advisers such as GPU Smith; none of the conclusions above depends on such an engagement.

Frequently Asked Questions (FAQs)

Can Triton Inference Server run a TensorRT model across multiple GPUs?

Yes, from release 26.07 the Triton TensorRT backend supports multi-device inference. The engine must be sharded offline and served through a single KIND_MODEL instance that owns the GPUs, as the Executive Summary and the support-boundary section describe.

How do you run one model on multiple GPUs in Triton, and how does that differ from `instance_group` with several GPUs?

A KIND_GPU `instance_group` with several GPU indices creates independent copies of the model, so different requests run in parallel. KIND_MODEL with `enable_multi_device` and `multi_device_gpus` creates one instance whose GPUs cooperate on every request.

What is the difference between Dynamo-Triton and Triton Inference Server?

NVIDIA says Dynamo-Triton is the former Triton Inference Server, and the 26.07 release notes still use the Triton title. NVIDIA Dynamo is described on NVIDIA's pages as a separate LLM-oriented offering and as the successor to Triton, and the two pages frame it differently, as noted above.

What GPUs does TensorRT multi-device require?

Distributed collectives require Ampere (SM 80) or later, and the multi-device attention operator requires Blackwell (SM 100) or later, as *Table 1* summarizes.

Tensor parallelism or data parallelism?

Use data parallelism (replicas) when the model fits on one GPU and traffic is parallel. Use tensor or context splits when a single request must be faster or the weights do not fit. PyTorch notes that a model too large for one GPU requires model parallelism (pytorch.org).

When should a model be split across GPUs?

When the single-GPU baseline misses the latency SLO or the memory budget, a verified sharding recipe exists, quality survives numerical differences, and the extra GPU-seconds per request are acceptable under *Table 2*.

Conclusion

TensorRT multi-device in Dynamo-Triton is a well-bounded capability: an offline-sharded plan, a single KIND_MODEL instance with an explicit GPU list, NCCL collectives on Ampere or newer hardware, and Blackwell-only attention operators. The Cosmos 3 Nano result (a 16B-parameter model (huggingface.co)) shows a 4.58x end-to-end speedup on eight GPUs for one generative-video pipeline, while the backend guide shows near-neutral gains for a shallow attention test. Every other model falls somewhere between those two reported results, and the evidence for where must come from the buyer's own run sheet.

The recommended posture is conditional adoption. Adopt where a model is memory-bound or misses its latency SLO on one GPU, where a sharded recipe exists, where quality passes a predefined gate, and where the additional GPU-seconds per completed request are accepted explicitly. Otherwise keep single-GPU instances and add replicas. Measure transformer time, end-to-end latency, throughput, concurrency and GPU-seconds separately, and decide on all five together.

Source links

Links cited in this article, in order of first appearance. Full addresses are provided for readers using extracted text.

1. https://docs.nvidia.com/deeplearning/triton-inference-server/release-notes/rel-26-07.html
2. https://developer.nvidia.com/dynamo-triton
3. https://github.com/triton-inference-server/tensorrt_backend/blob/main/docs/multi_device.md
4. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/overview.html
5. https://developer.nvidia.com/blog/simplifying-model-serving-across-multiple-gpus-with-nvidia-tensorrt-multi-device-integration-in-nvidia-dynamo-triton
6. https://docs.nvidia.com/deeplearning/tensorrt/latest/inference-library/multi-device-inference.html
7. https://huggingface.co/nvidia/Cosmos3-Nano
8. https://developer.nvidia.com/dynamo
9. https://pytorch.org/tutorials/intermediate/ddp_tutorial.html
10. https://arxiv.org/pdf/1909.08053
11. https://arxiv.org/pdf/2309.14509
12. https://github.com/triton-inference-server/common/blob/main/protobuf/model_config.proto
13. https://docs.nvidia.com/deeplearning/tensorrt/latest/architecture/how-trt-works.html
14. https://docs.nvidia.com/deeplearning/triton-inference-server/archives/triton-inference-server-2710/user-guide/docs/tensorrt_backend/docs/multi_device.html
15. https://www.nvidia.com/en-us/data-center/a100
16. https://www.nvidia.com/en-us/data-center/h100

17. https://www.nvidia.com/en-us/data-center/gb200-nvl72
18. https://www.nvidia.com/en-us/data-center/nvlink
19. https://us.design-reuse.com/news/46222/32gt-s-pci-express-5-0-specification.html
20. https://developer.nvidia.com/blog/nvidia-nvlink-and-nvidia-nvswitch-supercharge-large-language-model-inference
21. https://cvw.cac.cornell.edu/scalability/scalability-speedup/faster-or-more
22. https://users.cs.utah.edu/~hari/teaching/paralg/Gustafson88.pdf
23. https://hpc-wiki.info/hpc/Benchmarking_%26_Scaling_Tutorial/Introduction
24. https://lambda.ai/pricing
25. https://aws.amazon.com/ec2/instance-types/p5

THE PUBLISHER

About GPU Smith

GPU Smith provides independent engineering and research for private AI infrastructure. We help technical buyers and operators reason about GPU workload sizing, hardware selection, procurement, deployment and inference operations, from the compute system to the networking, power and cooling requirements around it.

Start with the workload

Useful infrastructure decisions begin with the models, data, throughput, latency and operating constraints a team actually has. GPU Smith helps connect those requirements with system design and deployment choices. The goal is a privately controlled AI environment whose capacity and operational demands are understood before a purchasing decision.

Hardware and supplier research

Our public reference library covers GPUs, complete systems and networking components. The server-vendor directory and manufacturing research help teams investigate suppliers, compare options and follow sources behind technical claims. We distinguish manufacturer specifications from measured benchmarks and advertised capabilities from tested configurations. Reference pages are research resources, not a live inventory listing or a binding equipment quote.

Deployment and operations

GPU Smith's engineering scope includes infrastructure integration, networking, power, cooling and the practical operation of inference workloads. Our published articles explain the tradeoffs behind deployment, procurement and ongoing operations so teams can ask better questions and document decisions.

Work with GPU Smith

Explore the hardware library, GPU references, networking references and vendor research. Contact GPU Smith with your workloads and deployment constraints to discuss a project and confirm current scope, pricing and availability.

Inclusion of a manufacturer or product in our research does not imply a partnership, certification or endorsement.

Website: https://gpusmith.com

This article is educational. Verify consequential medical, legal, financial, regulatory and technical decisions with appropriate professionals and the cited primary sources. Company services are described separately from the article's research findings.