

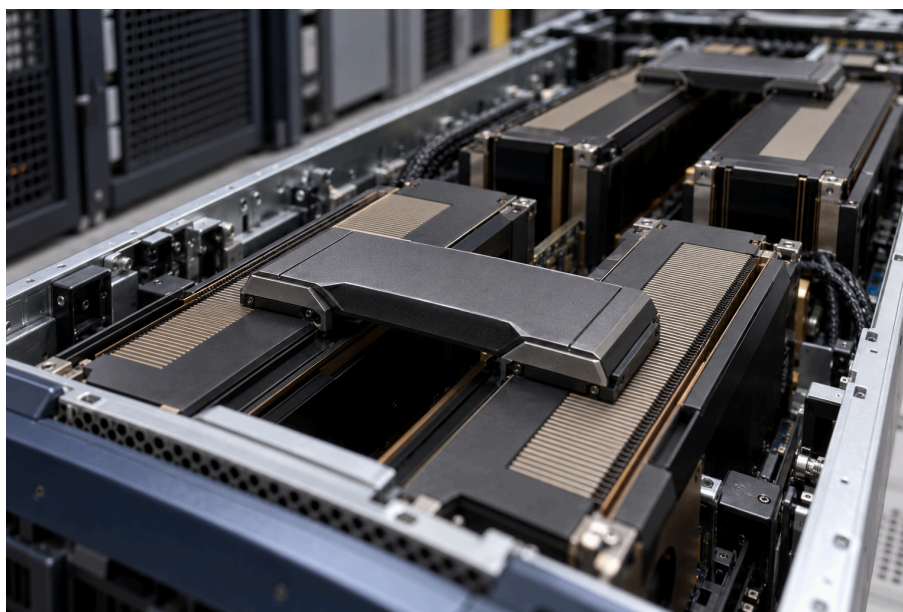


Tensor vs Pipeline Parallelism for Inference: NVLink vs PCIe

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 · 2026-10-29 · tensor parallel vs pipeline parallel inference · vllm tensor_parallel_size pipeline_parallel_size · tensor parallelism pcie vs nvlink all-reduce · l40s multi gpu llm inference pipeline parallel · how many gpus tensor parallel 70b · multi node vllm pipeline parallelism ray

Original article: <https://gpusmith.com/articles/en/tensor-vs-pipeline-parallelism-inference-pcie-nvlink>



In this article

1. The decision in one paragraph
 2. What TP and PP actually communicate
 3. Bandwidth numbers that matter
 4. Per-token communication estimate
 5. TP on PCIe: when it still works
 6. When pipeline parallelism wins
 7. GPU count granularity
 8. Multi-node: keep TP inside the node
 9. Interaction with MFU and batch size
 10. Test plan with NCCL tests
 11. Recommendations table
 12. Sources
-

The decision in one paragraph

When a model does not fit on one GPU, you split either the math inside every layer (tensor parallelism, TP) or the stack of layers (pipeline parallelism, PP). TP cuts per-token latency because every GPU works on every token, but puts two all-reduces per layer on the critical path, so it needs a fast interconnect. PP sends one activation tensor per stage boundary and tolerates slow links, but each token walks through the stages serially. vLLM's documentation states the rule: "if the model is too large for a single GPU but fits on a single node with multiple GPUs, use tensor parallelism", and "if the model is too large for a single node, combine tensor parallelism with pipeline parallelism", with `tensor_parallel_size` set to GPUs per node and `pipeline_parallel_size` to the number of nodes ([vLLM Parallelism and Scaling](#)). The same page carves out the PCIe case: if the GPUs "do not have NVLINK interconnect (e.g. L40S), leverage pipeline parallelism instead of tensor parallelism for higher throughput and lower communication overhead." Below is why, with arithmetic you can redo for your model.

What TP and PP actually communicate

Tensor parallelism. Megatron-LM splits the first MLP matrix by columns and the second by rows, so the MLP needs one all-reduce at its output; attention is split by heads the same way, giving "only two all-reduces in the forward path and two in the backward path" per transformer layer ([Shoeybi et al., Megatron-LM](#)). Inference has no backward pass, so the count is two all-reduces per layer per forward step. Sarathi-Serve describes inference TP the same way, "two all-reduce operations per layer – one in attention computation and the other in FFN", and notes that "since these communication operations are in the critical path, TP is preferred only within a single node" ([Agrawal et al., Sarathi-Serve](#)).

Each all-reduce carries the full hidden state, (tokens x hidden size), and no GPU can start the next layer until it completes, so the slowest link sets the pace.

Pipeline parallelism. Each GPU (or TP group) owns a contiguous block of layers and passes the hidden state to the next stage with one point-to-point send. Narayanan et al. give the training version: traffic between consecutive pipeline devices is *bsh* (batch x sequence x hidden) per micro-batch, whereas TP all-reduces tensors of size *bsh* "twice each in the forward and backward pass for each layer" (Narayanan et al., [Megatron-LM at scale](#)).

PP pays in time instead: a token's forward pass is serial across stages, and stages idle when the schedule cannot keep each one busy (the pipeline bubble). Mixed prefill and decode micro-batches have uneven run times, which makes bubbles worse in inference than in uniform training batches.

vLLM's tuning guide puts the trade-off in one line each: increasing TP "may cause excessive synchronization overhead", while increasing PP "may cause latency penalties" ([vLLM Optimization and Tuning](#)).

Bandwidth numbers that matter

Datasheet interconnect figures are usually bidirectional totals. A ring all-reduce sends and receives at the same time, so the number to divide by is the per-direction rate.

Interconnect	Datasheet figure	Per direction	Source
H100 SXM, NVLink 4	900 GB/s	450 GB/s (18 links x 25 GB/s each way)	NVIDIA H100, Hopper in depth
H100 NVL, NVLink bridge (pairs only)	600 GB/s bidirectional	~300 GB/s	NVIDIA H100, H100 NVL product brief
PCIe Gen5 x16 (H100 SXM and NVL)	128 GB/s	~64 GB/s	NVIDIA H100
L40S, PCIe Gen4 x16, no NVLink	64 GB/s bidirectional	~32 GB/s	NVIDIA L40S

NVIDIA's Hopper post describes "18 fourth-generation NVLink links to provide 900 GB/sec total bandwidth", each with "25 GB/sec effective bandwidth in each direction" ([NVIDIA Hopper architecture in depth](#)). The L40S page lists "PCIe Gen4 x16: 64GB/s bidirectional" and NVLink "No" ([NVIDIA L40S](#)); Gen5 doubles that, so its 128 GB/s is also a bidirectional figure.

These are link rates. Achieved all-reduce bandwidth is lower, and on a PCIe server the ring may cross a switch uplink or the CPU socket interconnect. The H100 NVL's 600 GB/s applies only within a bridged pair: NVIDIA's product brief says the card bridges "with a single adjacent NVIDIA H100 NVL card", so a TP=4 or TP=8 group of H100 NVLs crosses PCIe between pairs. The H200 NVL adds 4-way bridges ([NVIDIA H200](#)).

Per-token communication estimate

This is a derived estimate, not a sourced benchmark: it counts bytes and divides by link rate, ignoring latency, overlap and protocol overhead, so treat outputs as lower bounds.

Inputs.

- L = number of transformer layers, h = hidden size, b = bytes per activation element (2 for BF16).
- T = tokens in the step (decode batch size, or prompt tokens for prefill).
- N = TP degree.

Step 1: message size. Each TP all-reduce reduces a tensor of $T \times h$ elements: $S = T \times h \times b$ bytes.

Step 2: all-reduces per forward pass. Two per layer gives $2L$.

Step 3: ring cost. NCCL's performance notes derive that an all-reduce of S bytes over n ranks needs " $2(n-1)$ data transfers", a best-case time of $t = S \times 2(n-1) / (n \times B)$ with B the per-rank link bandwidth, a factor "independent of the algorithm used (ring, tree, or other)" ([NCCL tests PERFORMANCE.md](#)). So each GPU sends (and receives) $2(N-1)/N \times S$ bytes per all-reduce.

Result (estimate). Bytes each GPU sends per forward step:

$$V_{TP} = 2L \times T \times h \times b \times 2(N-1)/N$$

and a lower bound on TP communication time is $V_{TP} / B_{per_direction}$. For comparison, PP sends $T \times h \times b$ bytes once per stage boundary.

Worked example: Llama 3.1 70B

Llama 3 70B has 80 layers, model dimension 8,192, 64 attention heads and 8 key/value heads ([Llama Team, The Llama 3 Herd of Models, Table 3](#)). Activations in BF16, so $b = 2$.

1. Bytes per token per all-reduce: $8,192 \times 2 = 16,384$ bytes (16 KiB).
2. All-reduces per token: $2 \times 80 = 160$. Sum of message sizes: $160 \times 16,384 = 2,621,440$ bytes, about 2.62 MB per token.
3. Ring factor: $2(N-1)/N = 1.5$ at $N = 4$, 1.75 at $N = 8$.
4. Per-GPU traffic per token: $2.62 \text{ MB} \times 1.5 = 3.93 \text{ MB}$ at $TP=4$; $2.62 \text{ MB} \times 1.75 = 4.59 \text{ MB}$ at $TP=8$.

Scaled to three step sizes and divided by per-direction rates:

Step	TP	Per-GPU bytes	NVLink 4 (450 GB/s)	PCIe Gen5 (64 GB/s)	PCIe Gen4 (32 GB/s)
Decode, batch 1	4	3.93 MB	0.009 ms	0.061 ms	0.12 ms
Decode, batch 64	4	252 MB	0.56 ms	3.9 ms	7.9 ms
Decode, batch 64	8	294 MB	0.65 ms	4.6 ms	9.2 ms
Prefill, 4,096 tokens	4	16.1 GB	36 ms	252 ms	503 ms
Prefill, 4,096 tokens	8	18.8 GB	42 ms	294 ms	587 ms

Going from $TP=4$ to $TP=8$ raises per-GPU traffic (1.5x to 1.75x the message) while each GPU's compute halves, so communication's share of step time grows with TP degree.

Decode in context. Decode is usually bound by weight reads. With roughly 140 GB of BF16 weights (70B x 2 bytes) split over four H100 SXM GPUs at 3.35 TB/s each ([NVIDIA H100](#)), each GPU reads 35 GB per step, a floor of about 10.4 ms. At batch 64, NVLink adds 0.56 ms of transfer (about 5%). The same bytes over PCIe Gen5 would add 3.9 ms (about 38%) unless overlapped.

Latency, not bandwidth, at batch 1. At batch 1 the bandwidth term is microseconds, but 160 all-reduces run one after another in each step. If each costs a fixed 5 to 25 μ s (an assumption; measure it), that is 0.8 to 4 ms per token from launch and synchronization latency alone. Small-batch interactive serving therefore feels TP overhead even when bandwidth looks ample.

Prefill in context. A 4,096-token prefill of a 70B model is about $2 \times 70e9 \times 4,096 = 5.7e14$ FLOP. Four H100 SXM at half the 1,979 TFLOPS BF16 sparsity figure ([NVIDIA H100](#)), assuming 50% utilization, need about 290 ms. NVLink's 36 ms is small next to that; PCIe Gen5's 252 ms is nearly as large as the compute, so without overlap TP over PCIe can nearly double time to first token on long prompts.

PP comparison. At batch 64 a stage boundary carries $64 \times 16,384 = 1$ MiB (16 μ s at 64 GB/s), about 240 times less than TP=4's 252 MB per GPU.

TP on PCIe: when it still works

TP over PCIe is tolerable with:

- **Small TP degree.** TP=2 has a ring factor of 1.0; use it when the model just misses one card.
- **Short prompts, moderate batches**, where the decode bandwidth term is a few milliseconds against a weight-read floor several times larger (see the L40S numbers below).

The L40S case shows the cost. Take Llama 3.1 70B with FP8 weights (about 70 GB) on four L40S cards (48 GB, 864 GB/s each; [NVIDIA L40S](#)):

- **TP=4.** Each GPU reads about 17.5 GB per decode step, a floor of about 20 ms. At batch 64, BF16 all-reduce traffic adds about 7.9 ms at 32 GB/s, roughly 39% on top, plus per-message latency.
- **PP=4.** Each stage also reads about 17.5 GB in about 20 ms, but a token crosses all four stages, so per-token latency is about 81 ms. With four micro-batches in flight, aggregate tokens per second can match TP's while the links carry only 1 MiB per boundary.

These idealized estimates match vLLM's L40S advice: PP for aggregate throughput, TP for lower inter-token latency if you can afford the PCIe tax. Four micro-batches of 64 means 256 concurrent sequences, so PP needs enough KV cache and load. For card-level trade-offs on PCIe-only servers see [RTX PRO 6000 vs L40S](#).

When pipeline parallelism wins

- **Across nodes.** Narayanan et al.'s first takeaway is that "tensor model parallelism should generally be used up to degree g when using g -GPU servers, and then pipeline model parallelism can be used to scale up to larger models across servers" ([Narayanan et al.](#)). That is a training result, but the reasoning

(all-reduce on slow links) carries over.

- **Inference evidence.** Sarathi-Serve ran Falcon-180B on two 4xA100 nodes "connected over 100 Gbps Ethernet". TP-8 had median decode latency " $\sim 2\times$ higher" than TP-4 x PP-2 "because TP incurs high communication overhead due to cross-node all-reduces". With their bubble-reducing scheduler they report "up to $5.6\times$ gains in end-to-end serving capacity" for that deployment ([Agrawal et al., Sarathi-Serve](#)). The $5.6\times$ is against a 2024 vLLM baseline and measures scheduling, not PP alone.
- **Throughput-oriented serving on PCIe cards**, as above.
- **GPU counts TP cannot use**, covered next.

PP loses at low concurrency with tight inter-token latency targets: stages sit idle and each token pays the full serial path.

GPU count granularity

vLLM rejects any TP size that does not divide the attention head count: "Total number of attention heads (...) must be divisible by tensor parallel size". KV heads are divided by the TP size, and replicated when there are fewer KV heads than ranks, "so each GPU has at least one KV head" ([vLLM source](#), `vllm/config/model.py`, v0.32.0 tag, released 2026-10-18).

For Llama 3 70B (64 query heads, 8 KV heads):

GPUs	Valid TP?	Notes
2, 4, 8	Yes	KV heads split 4, 2, 1 per GPU
16	Yes	8 KV heads replicated to 1 per GPU, so per-GPU KV memory stops shrinking past TP=8
3, 6, 12	No for TP alone	64 is not divisible by 3

A 6-GPU box therefore runs TP=2 x PP=3, or PP=6 with uneven stages (14, 14, 13, 13, 13, 13 layers). vLLM notes PP "splits the model along layers and supports uneven splits", and recommends `tensor_parallel_size=1` with `pipeline_parallel_size` equal to the GPU count when the count does not divide the model ([vLLM Parallelism and Scaling](#)). The 14-layer stages then set throughput. For PaLM 540B on TPU v4, Pope et al. "padded the number of attention heads up from 48 to 64 in order to partition more" effectively ([Pope et al.](#)).

Multi-node: keep TP inside the node

vLLM's multi-node examples run `--tensor-parallel-size 8 --pipeline-parallel-size 2` on two 8-GPU nodes, with Ray (`--distributed-executor-backend ray`) or multiprocessing (`--nnodes, --node-rank, --master-addr`). If `NCCL_DEBUG=TRACE` logs show `[send] via NET/Socket`, "NCCL used a raw TCP socket, which is *not* efficient for cross-node tensor parallelism"; look for `NET/IB/GDRDMA` instead ([vLLM Parallelism and Scaling](#)).

The Megatron papers behind this layout report **training** results, not inference:

- Shoeybi et al. trained models "up to 8.3 billion parameters using 512 GPUs" with intra-layer TP, reporting 15.1 PetaFLOPs at 76% scaling efficiency ([Megatron-LM](#)).
- Narayanan et al. combined TP within DGX A100 servers, PP across them and data parallelism: 502 petaFLOP/s on 3,072 GPUs for a 1-trillion-parameter model, 52% of peak per-GPU throughput. They found "sub-optimal combinations of tensor and pipeline model parallelism can lead to up to 2× lower throughput, even with high-bandwidth network links between servers" ([Narayanan et al.](#)).

Training adds backward-pass and gradient traffic that inference lacks, so treat these as evidence for the topology rule, not inference figures. The exception is a scale-up domain larger than a server (NVLink Switch racks), where the NVLink domain is the boundary; see [NVLink 5 vs NVLink 6](#). For MoE models see [MoE expert parallelism sizing](#).

Interaction with MFU and batch size

Pope et al. built an analytical model for choosing inference partitionings on TPU v4 (270 GB/s interconnect per chip) ([Pope et al.](#)). In their 1D layout, the Megatron-style split, communication time per layer is about $2BLE / \text{bandwidth}$ (batch x length x embedding) and does not shrink as chips are added, while compute does. That is the same effect as the rising per-GPU bytes above. They also found the best layout changes with tokens per batch, reporting 29 ms per token at low batch (int8 weights) and 76% MFU on large-batch prefill for PaLM 540B. These are TPU results, not GPU results.

In practice: small-batch decode is latency-bound, so TP's per-message cost dominates. Large-batch decode is weight-read-bound, so the bandwidth term scales with link speed. Long prefills are compute-bound, and the PCIe bandwidth term can match compute time. Chunked prefill caps tokens per step, so it also caps each all-reduce.

Test plan with NCCL tests

Measure before you choose:

1. **Sweep sizes.** `./build/all_reduce_perf -b 8 -e 128M -f 2 -g 8` starts at 8 bytes, doubles each step and uses 8 GPUs ([NCCL tests](#)); use `-g 4` for a 4-GPU group.
2. **Read small-message time** at 16 KiB to 1 MiB: that is your per-all-reduce latency at decode batch sizes. Multiply by 2L for per-step overhead.
3. **Read busbw at large sizes** (busbw = algbw x 2(n-1)/n, per [NCCL tests PERFORMANCE.md](#)) and use it instead of the datasheet rate in the estimate.
4. **Test the real groups.** On PCIe servers, test GPUs under one switch separately from GPUs across sockets.
5. **Confirm in the engine** with TP=N, PP=N and a hybrid under your real prompt lengths and concurrency, comparing time to first token, inter-token latency and goodput.

Algorithm and protocol choice shifts these curves; see [NCCL algorithms and protocols](#).

Recommendations table

Model fits on	Interconnect	Start with	Why
1 GPU	n/a	No model parallelism; scale with replicas	No communication at all
2-8 GPUs, one node	NVLink / NVSwitch	TP = GPUs needed (must divide heads)	Low latency, small comm share
2 GPUs, one node	PCIe only	TP=2	Ring factor 1.0; modest overhead
4-8 GPUs, one node	PCIe only (e.g. L40S)	PP for throughput; TP if inter-token latency matters more and you measured the cost	vLLM guidance plus the estimate above
GPU count not dividing heads (3, 6)	Any	PP, or TP x PP using a valid TP	Divisibility rule
More than one node	NVLink inside, IB or Ethernet between	TP = GPUs per node, PP = nodes	Keep all-reduces off the network
Rack-scale NVLink domain	NVLink Switch	TP or EP within the domain	The domain, not the chassis, is the boundary

All rows are starting points to confirm with the test plan. GPU Smith helps teams with this kind of parallelism and interconnect sizing.

Sources

1. [Parallelism and Scaling](#), vLLM documentation, v0.32.0 (released 2026-10-18; accessed 2026-10-28)
2. [Optimization and Tuning](#), vLLM documentation, v0.32.0 (accessed 2026-10-28)
3. [vllm/config/model.py](#), vLLM project, GitHub, v0.32.0 tag (accessed 2026-10-28)
4. [NVIDIA H100 Tensor Core GPU](#), NVIDIA (accessed 2026-10-28)
5. [NVIDIA Hopper Architecture In-Depth](#), NVIDIA Technical Blog, 2022 (accessed 2026-10-28)
6. [NVIDIA L40S GPU](#), NVIDIA (accessed 2026-10-28)
7. [NCCL tests: Performance reported by NCCL tests](#), NVIDIA, GitHub (accessed 2026-10-28)
8. [NCCL tests](#), NVIDIA, GitHub (accessed 2026-10-28)
9. [Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism](#), Shoeybi et al., arXiv, 2019 (accessed 2026-10-28)
10. [Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM](#), Narayanan et al., arXiv, 2021 (accessed 2026-10-28)
11. [Efficiently Scaling Transformer Inference](#), Pope et al., arXiv, 2022 (accessed 2026-10-28)
12. [Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve](#), Agrawal et al., arXiv, 2024 (accessed 2026-10-28)
13. [The Llama 3 Herd of Models](#), Llama Team, AI at Meta, arXiv, 2024 (accessed 2026-10-28)
14. [NVIDIA H100 NVL product brief, PB-11773-001_v01](#), NVIDIA (accessed 2026-10-28)

Source links

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

1. https://docs.vllm.ai/en/v0.32.0/serving/parallelism_scaling
2. <https://arxiv.org/abs/1909.08053>
3. <https://arxiv.org/abs/2403.02310>
4. <https://arxiv.org/abs/2104.04473>
5. <https://docs.vllm.ai/en/v0.32.0/configuration/optimization>
6. <https://www.nvidia.com/en-us/data-center/h100>
7. <https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth>
8. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/h100/PB-11773-001_v01.pdf
9. <https://www.nvidia.com/en-us/data-center/l40s>
10. <https://www.nvidia.com/en-us/data-center/h200>
11. <https://github.com/NVIDIA/nccl-tests/blob/master/doc/PERFORMANCE.md>
12. <https://arxiv.org/abs/2407.21783>
13. <https://github.com/vllm-project/vllm/blob/v0.32.0/vllm/config/model.py>
14. <https://arxiv.org/abs/2211.05102>
15. <https://github.com/NVIDIA/nccl-tests>

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.