

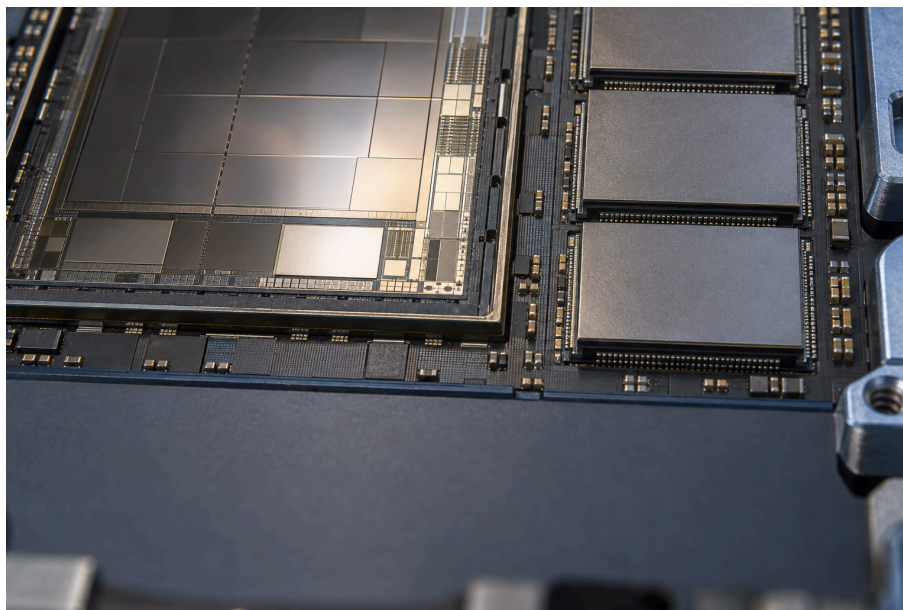


Decode Is Memory-Bound: Tokens per Second from HBM Bandwidth

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 · llm tokens per second memory bandwidth formula · estimate tokens per second from gpu bandwidth · is llm inference memory bound or compute bound · roofline model llm inference decode · b300 memory bandwidth · h100 memory bandwidth 3.35 tb/s · dgx spark memory bandwidth · dgx spark tokens per second

Original article: <https://gpusmith.com/articles/en/decode-memory-bandwidth-roofline-tokens-per-second>



In this article

1. The one-line model
 2. Why decode is memory-bound
 3. Bytes per token: weights
 4. Bytes per token: KV cache
 5. Batching changes the math
 6. Datasheet bandwidth table
 7. Worked example A: 70B dense at FP8
 8. Worked example B: large MoE (DeepSeek-V3 on 8× H200)
 9. Realistic efficiency factors
 10. Checking the ceiling against measurements
 11. When the model fails
 12. FAQ
 13. Sources
-

The one-line model

During autoregressive decode, each generated token requires a forward pass that reads the active model weights and the request's KV cache out of GPU memory. If memory bandwidth is the bottleneck, the speed limit for a single stream is:

$$\text{tokens/s per stream} \leq \text{memory bandwidth} \div \text{bytes read per token}$$

and its inverse is the floor on inter-token latency (ITL):

$$\text{ITL} \geq \text{bytes read per token} \div \text{memory bandwidth}$$

For a 70B-parameter dense model with FP8 weights (about 70 GB) on an H100 SXM at 3.35 TB/s ([NVIDIA H100](#)), the weights alone give $3.35 \text{ TB/s} \div 70 \text{ GB} \approx 48 \text{ tokens/s}$, which is an ITL floor of about 21 ms. No kernel tuning gets past that number at batch size 1. Only reading fewer bytes per token or adding bandwidth does.

The rest of this article covers what goes into "bytes read per token", how batching changes the picture, where the roofline says decode turns compute-bound, and how far real systems fall short of the ceiling. All throughput figures below are derived upper bounds computed from vendor peak specifications. They are not measurements.

Why decode is memory-bound

The roofline model compares a kernel's **arithmetic intensity** (FLOPs per byte moved from memory) with the hardware's **ridge point** (peak FLOP/s ÷ peak bytes/s). Below the ridge, a kernel is limited by bandwidth. Above it, the limit is compute. The [LLM Inference Unveiled survey](#) builds its whole analysis of inference techniques on this framework, starting from the question of "why LLMs are memory-bound".

Arithmetic intensity of a decode GEMM

Take one weight matrix of $d_{in} \times d_{out}$ stored at b bytes per parameter, multiplied by a batch of B token vectors. The work is $2 \cdot B \cdot d_{in} \cdot d_{out}$ FLOPs. The weight traffic is $b \cdot d_{in} \cdot d_{out}$ bytes, and at small B the activations are negligible next to it. So:

$$\text{intensity} \approx 2B \div b \text{ FLOP/byte}$$

With FP8 weights ($b = 1$), intensity is $2B$. At batch 1 that is **2 FLOP/byte**.

Ridge points from datasheets

NVIDIA lists Tensor Core figures "with sparsity". The Blackwell pages say outright that "Dense is $\frac{1}{2}$ sparse spec shown" ([NVIDIA HGX](#)). The table below applies the same halving to Hopper.

GPU	Dense FP8 (FLOP/s)	Peak bandwidth	Ridge point (FLOP/byte)	Batch where FP8 GEMMs cross the ridge
H100 SXM	1,979 T (3,958 T sparse $\div 2$)	3.35 TB/s	591	~295
H200 SXM	1,979 T (3,958 T sparse $\div 2$)	4.8 TB/s	412	~206
B200 or B300 in HGX/DGX	4.5 P (72 P sparse per 8 GPUs $\div 8 \div 2$)	8 TB/s	563	~281
B200 or B300 in GB200/GB300 NVL72	5 P (720 P sparse per 72 GPUs $\div 72 \div 2$)	8 TB/s	625	~313
MI355X	5 P (OCP-FP8, no sparsity)	8 TB/s	625	~313

Inputs: [H100](#), [H200](#), [HGX](#), [DGX B200](#), [GB200 NVL72](#), [NVIDIA's Blackwell Ultra architecture post](#) (Table 2, "Dense FP8: 5 PFLOPS" for both Blackwell and Blackwell Ultra) and the [AMD MI355X page](#). Ridge points and crossover batches are our arithmetic. The HGX and DGX B200/B300 pages list 72 PFLOPS FP8 per 8 GPUs, footnoted sparse, so 4.5 PFLOPS dense per GPU; the GB200 and GB300 NVL72 pages list 720 PFLOPS per 72 GPUs, 5 PFLOPS dense per GPU, which is the architecture post's figure (all read 2026-10-28). The gap reflects configuration and power envelope, not an error; use the row for the system you buy.

At batch 1, an H100 running an FP8 decode GEMM sits at 2 FLOP/byte against a ridge of 591. The GPU uses roughly 0.3% of its dense FP8 compute before the memory system saturates. The [How To Scale Your Model inference chapter](#) reaches the same conclusion for TPUs: matmuls become compute-bound only once the per-replica token batch exceeds a critical size set by the FLOPs-to-bandwidth ratio (240 tokens for bf16 on TPU v5e). The analysis in [Pope et al.](#), which reports 29 ms per token at low batch on PaLM 540B with int8 weights, rests on the same weight-loading argument.

Attention never crosses the ridge

Weights are shared across the batch. KV cache is not, because each sequence reads only its own keys and values. With grouped-query attention, each KV element is used by g query heads, so attention intensity is about $2g \div b_{kv}$ FLOP/byte no matter how large the batch gets. Llama 3 70B has 64 query heads and 8 KV

heads ([Llama 3 paper, Table 3](#)), which gives $g = 8$. With an FP8 KV cache that is about 16 FLOP/byte, far below every ridge in the table. This is the main reason long-context decode stays memory-bound even at large batch.

Bytes per token: weights

Weight bytes per token = **active parameters** × **bytes per parameter**.

- **Dense models** read every parameter for every token. A 70B model needs ~140 GB at BF16, ~70 GB at FP8 and ~39 GB at NVFP4.
- **NVFP4** is not a flat 0.5 bytes. NVIDIA describes it as "one 4-bit value plus minor overhead of one FP8 scale per 16 values (4.5 bits per value)" ([NVIDIA NVFP4 post](#)). That works out to 0.5625 bytes per parameter, or 39.4 GB for 70B.
- **Mixture-of-experts (MoE) models** read only the routed experts for each token. DeepSeek-V3 has "671B total parameters with 37B activated for each token" ([DeepSeek-V3 technical report](#)). At batch 1 that means ~37 GB of FP8 reads per token. The full 671 GB still has to sit in memory.

Bytes per token: KV cache

KV bytes read per token = **context length** × **KV bytes per token of context**. For standard multi-head or grouped-query attention:

KV bytes per context token = $2 \text{ (K and V)} \times \text{layers} \times \text{KV heads} \times \text{head dim} \times \text{bytes per element}$

For Llama 3 70B (80 layers, 8 KV heads, model dimension 8,192 over 64 heads = head dim 128, per [Table 3](#)) with an FP8 cache:

$2 \times 80 \times 8 \times 128 \times 1 = \mathbf{163,840 \text{ bytes (160 KiB) per context token}}$

Context	KV read per decode step, one sequence (FP8)	Share of a 70 GB weight read
8,192	1.34 GB	1.9%
32,768	5.37 GB	7.7%
131,072	21.5 GB	30.7%

DeepSeek-V3 uses multi-head latent attention. Its config lists `kv_lora_rank 512`, `qk_rope_head_dim 64` and 61 layers ([DeepSeek-V3 config.json](#)), so the cached latent is $(512 + 64) \times 61 \times 2 \text{ bytes} = \mathbf{70,272 \text{ bytes (~69 KiB) per context token}}$ at BF16. That is our derivation, assuming the latent is cached in BF16. For full capacity math across architectures, use the [KV cache memory calculator](#).

At batch 1, KV barely matters for short chats. It dominates once you batch many long contexts.

Batching changes the math

One decode step produces one token for each of the B sequences in the batch. Weights are read once per step, and each sequence's KV is read once per step:

$$\text{bytes per step} = \text{active weights} + \sum (\text{context_i} \times \text{KV bytes per token}) \text{ aggregate tokens/s} \leq B \times \text{bandwidth} \div \text{bytes per step per-stream tokens/s} \leq \text{bandwidth} \div \text{bytes per step}$$

Batching amortizes the weight read and leaves the KV read untouched. Aggregate throughput rises, per-stream speed falls, and HBM capacity eventually caps how large B can grow. That capacity cap is covered in our [LLM inference hardware sizing](#) guide.

Datasheet bandwidth table

As of October 2026, peak bandwidth per GPU from vendor pages:

Device	Memory	Peak bandwidth	Status of the per-GPU figure
H100 SXM	80 GB HBM3	3.35 TB/s	Stated (NVIDIA)
H100 NVL	94 GB	3.9 TB/s	Stated (NVIDIA)
H200 SXM	141 GB	4.8 TB/s	Stated (NVIDIA)
B200 (Blackwell)	180 GB in DGX B200	8 TB/s	Stated in NVIDIA's architecture post; DGX B200 page gives 64 TB/s for 8 GPUs
B300 (Blackwell Ultra)	288 GB HBM3e (HGX B300 page: 2.1 TB per 8 GPUs)	8 TB/s	Stated in NVIDIA's architecture post; GB300 NVL72 "up to 576 TB/s" $\div 72 = 8$ TB/s (derived)
GB200 Superchip (2 GPUs)	372 GB	16 TB/s	Stated per superchip (NVIDIA); 8 TB/s per GPU derived
AMD MI355X	288 GB HBM3E	8 TB/s	Stated (AMD)
RTX PRO 6000 Blackwell Server	96 GB GDDR7	1,597 GB/s	Stated (NVIDIA)
DGX Spark	128 GB LPDDR5x, 256-bit	273 GB/s	Stated (NVIDIA)

Notes on the Blackwell rows:

- The [DGX B200](#) page lists "1,440 GB total, 64 TB/s HBM3e bandwidth". The [HGX](#) and [DGX B300](#) pages have no memory bandwidth row.
- The explicit per-GPU figure comes from [NVIDIA's Blackwell Ultra post](#): "Bandwidth: 8 TB/s per GPU, 2.4x improvement over H100 (3.35 TB/s)". Its Table 2 lists 8 TB/s for both Blackwell and Blackwell Ultra. B300 added capacity, not bandwidth.
- AMD's comparison chart on the [MI355X](#) page lists the "B200 SXM5 180GB" at 7.7 TB/s. That is a competitor's figure; this article uses NVIDIA's 8 TB/s (DGX B200: 64 TB/s $\div$ 8).

Worked example A: 70B dense at FP8

Inputs: 70 GB of FP8 weights, FP8 KV at 160 KiB per context token, and an 8,192-token context.

Batch 1, single GPU

Bytes per token = 70 + 1.34 = 71.34 GB.

GPU	Ceiling (tokens/s)	ITL floor
H100 SXM (3.35 TB/s)	47.0	21.3 ms
H200 (4.8 TB/s)	67.3	14.9 ms
B200 / B300 / MI355X (8 TB/s)	112.1	8.9 ms
RTX PRO 6000 Server (1.597 TB/s)	22.4	44.7 ms

At 128K context, KV adds 21.5 GB and the H200 ceiling drops from 67.3 to 52.5 tokens/s. On an 80 GB H100, 70 GB of weights leaves about 10 GB for KV and runtime buffers, so even modest batches at 8K context do not fit.

Batching on one GPU (8K context per sequence)

Batch	Bytes per step	H200: per-stream / aggregate tokens/s	B200: per-stream / aggregate tokens/s
1	71.3 GB	67 / 67	112 / 112
8	80.7 GB	60 / 476	99 / 793
32	112.9 GB	43 / 1,360	71 / 2,266
64	155.9 GB	does not fit in 141 GB	51 / 3,284

Going from batch 1 to batch 32 on H200 raises the aggregate ceiling 20× and lowers per-stream speed by 37%. Batch 32 is still memory-bound: $2 \times 70e9 \times 32 = 4.5e15$ FLOPs per step takes 2.3 ms at 1,979 TFLOPS dense FP8, against 23.5 ms of memory time. The capacity limit arrives long before the ridge.

Tensor parallelism across two H100s

With TP=2, each GPU reads half the weights and half the KV heads: 35.67 GB per step, giving a 93.9 tokens/s ceiling (10.6 ms). Megatron-style TP adds two all-reduces per layer, or 160 per token for 80 layers.

Assumption: small-message all-reduce latency of 5-25 μ s adds 0.8-4.0 ms per token, which lowers the ceiling to roughly 68-87 tokens/s. Measure this on your own interconnect. The point is that TP divides the bandwidth term and adds a latency term that does not shrink.

Worked example B: large MoE (DeepSeek-V3 on 8× H200)

DeepSeek-V3 has 58 MoE layers (61 layers, of which the first 3 are dense), 256 routed experts plus 1 shared expert per MoE layer, 8 experts per token, hidden size 7,168 and expert FFN size 2,048 ([config.json](#)). Derived from those values:

- Parameters per expert = $3 \times 7,168 \times 2,048 \approx 44.0\text{M}$
- Routed expert parameters = $44.0\text{M} \times 256 \times 58 \approx 654\text{B}$, which is nearly all of the 671B
- Everything else (attention, dense layers, shared experts, embeddings) $\approx 17\text{B}$

Batch 1: ~ 37.5 GB of FP8 reads per token. Spread perfectly across eight H200s (38.4 TB/s aggregate), that is a $\sim 1,000$ tokens/s ceiling. This number is not useful on its own. At batch 1, 61 layers of expert dispatch, combine and collective operations dominate the step, and real single-stream speed is a small fraction of it.

Batching: different tokens route to different experts, so the weight read grows with batch until every expert is touched. **Assumption:** routing is uniform and independent. Then the expected number of distinct experts per layer is $256 \times (1 - (1 - 8/256)^B)$. KV uses MLA at ~ 69 KiB per context token with 8K context.

Batch	Distinct experts / layer	Weight bytes / step	KV / step	Aggregate ceiling (tokens/s)
1	8	38 GB	0.6 GB	$\sim 1,000$
8	57	164 GB	4.6 GB	$\sim 1,800$
32	163	434 GB	18.4 GB	$\sim 2,700$
64	222	585 GB	36.8 GB	$\sim 3,950$
256	256	671 GB	147 GB	$\sim 12,000$

MoE batching amortizes weights only after the batch is large enough to cover most experts. Up to that point, bytes per step grow almost as fast as batch size. It also helps to look at bytes per generated token: 37.5 GB at batch 1 versus 9.1 GB at batch 64, compared with 70 GB for the dense 70B model at batch 1. The 671 GB residency is what decides how many GPUs you buy. The active parameter count, combined with batch size, decides speed.

Realistic efficiency factors

The ceiling assumes the full peak bandwidth goes to useful bytes. Real systems lose some of it in several places:

- Achievable HBM bandwidth.** Peak figures are theoretical. Streaming kernels reach less, and decode kernels add launch gaps and tails between layers.
- End-to-end bandwidth utilization.** Databricks defines model bandwidth utilization (MBU) as achieved bandwidth $\div$ peak, with achieved = (model parameter size + KV cache size) $\div$ time per output token. It reported "a higher MBU of 60% on 2xH100-80GBs" at batch size 1, compared with 55% on 4xA100-

40GB, and noted that "MBU decreases as batch size increases" ([Databricks, October 2023](#)). That measurement is from 2023 software. **Assumption for planning:** budget for 50-80% of peak at small batch on a well-tuned modern stack, and confirm with your own runs.

- **Communication.** TP all-reduces and MoE all-to-all add latency that does not shrink with bandwidth (see example A).
- **Speculative decoding** works as a multiplier on the ceiling. One weight read can verify several drafted tokens, so tokens per weight read rise with the acceptance rate, at the cost of the draft model's own reads. The draft model's cost is its own sizing question.

Applying a 0.6-0.8 factor to example A's H200 batch-1 ceiling gives a planning range of about 40-54 tokens/s. That is an estimate, not a benchmark.

Checking the ceiling against measurements

Turn your measured ITL into achieved bandwidth and compare it with peak:

achieved bandwidth = bytes per step ÷ measured ITL

Illustrative: if a 70B FP8 model on one H200 at batch 1 with 8K context shows a 20 ms ITL, achieved bandwidth = $71.3 \text{ GB} \div 0.020 \text{ s} = 3.57 \text{ TB/s}$, which is 74% of 4.8 TB/s. That is healthy. At 35% you should look for a misconfiguration, such as weights not running in the precision you think, CUDA graphs disabled, a CPU-bound scheduler, or unintended TP communication. Measure ITL and TPS the way our [ITL and TPS metrics](#) article describes. Use steady-state decode only, excluding time to first token.

When the model fails

- **Prefill.** Prompt processing multiplies weights by thousands of tokens at once, so it runs far above the ridge and is compute-bound. Bandwidth math says nothing about time to first token.
- **Very large batches.** Once $B \cdot 2/b$ passes the ridge (about 200-310 tokens of FP8 batch in the table above), weight GEMMs become compute-bound. Attention still is not.
- **Small models.** An 8B BF16 model (16 GB) has a 500 tokens/s ceiling on an 8 TB/s GPU, which is 2 ms per token. Fixed per-step costs (kernel launches, sampling, scheduling, detokenization) become a large share of that 2 ms, so measured numbers fall well short of the ceiling.
- **CPU-bound serving.** Schedulers, structured-output masks and tokenization can stall the GPU between steps.
- **Multi-GPU and MoE.** Communication latency, expert load imbalance and pipeline bubbles are outside the model entirely.

FAQ

Why does DGX Spark feel slow on large models? Its 128 GB of LPDDR5x has 273 GB/s of bandwidth ([NVIDIA](#)). A 70B model at FP8 gives a ceiling of $273 \div 70 \approx 3.9$ tokens/s. NVFP4 (39.4 GB) gives about 6.9 tokens/s, and an 8B BF16 model about 17 tokens/s. Large capacity lets big models fit. It does not make them fast. Our [DGX Spark review](#) covers the rest of the trade-offs.

LPDDR vs HBM: how big is the gap? DGX Spark's 256-bit LPDDR5x gives 273 GB/s. H100 SXM HBM3 is about 12× that, and the 8 TB/s parts about 29×. Batch-1 decode speed follows the same ratio.

What is B300 memory bandwidth? NVIDIA states 8 TB/s per Blackwell Ultra GPU, the same as B200, with 288 GB of HBM3e (NVIDIA). The B300 therefore holds more KV cache than a B200, but its batch-1 decode ceiling for the same model is the same.

Does FP4 double decode speed over FP8? In the bandwidth model, weight bytes drop by about 1.8× (4.5 versus 8 bits per value), so a weight-dominated decode ceiling rises by up to ~1.8×. KV bytes only shrink if the KV cache is quantized as well.

If you want a second pair of eyes on a sizing model like this before a hardware purchase, [GPU Smith](#) does that work.

Sources

- [NVIDIA H100 Tensor Core GPU](#) — NVIDIA, accessed 2026-10-28
- [NVIDIA H200 Tensor Core GPU](#) — NVIDIA, accessed 2026-10-28
- [NVIDIA DGX B200](#) — NVIDIA, accessed 2026-10-28
- [NVIDIA HGX Platform](#) — NVIDIA, accessed 2026-10-28
- [NVIDIA DGX B300](#) — NVIDIA, accessed 2026-10-28
- [NVIDIA GB200 NVL72](#) — NVIDIA, accessed 2026-10-28
- [NVIDIA GB300 NVL72](#) — NVIDIA, accessed 2026-10-28
- [Inside NVIDIA Blackwell Ultra: The Chip Powering the AI Factory Era](#) — NVIDIA Technical Blog, 2025-08-22 (updated 2025-09-24), accessed 2026-10-28
- [AMD Instinct MI355X GPU](#) — AMD, accessed 2026-10-28
- [NVIDIA RTX PRO 6000 Blackwell Server Edition](#) — NVIDIA, accessed 2026-10-28
- [NVIDIA DGX Spark](#) — NVIDIA, accessed 2026-10-28
- [Introducing NVFP4 for Efficient and Accurate Low-Precision Inference](#) — NVIDIA Technical Blog, 2025-06-24, accessed 2026-10-28
- [LLM Inference Unveiled: Survey and Roofline Model Insights](#) — Yuan et al., arXiv, 2024, accessed 2026-10-28
- [Efficiently Scaling Transformer Inference](#) — Pope et al., arXiv, 2022-11-09, accessed 2026-10-28
- [All About Transformer Inference \(How To Scale Your Model\)](#) — Austin et al., accessed 2026-10-28
- [LLM Inference Performance Engineering: Best Practices](#) — Databricks, 2023-10-12, accessed 2026-10-28
- [The Llama 3 Herd of Models](#) — Meta, arXiv, 2024, accessed 2026-10-28
- [DeepSeek-V3 Technical Report](#) — DeepSeek-AI, arXiv, 2024-12-27, accessed 2026-10-28
- [DeepSeek-V3 config.json](#) — DeepSeek-AI on Hugging Face, 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://www.nvidia.com/en-us/data-center/h100>

2. <https://arxiv.org/abs/2402.16363>
3. <https://www.nvidia.com/en-us/data-center/hgx>
4. <https://www.nvidia.com/en-us/data-center/h200>
5. <https://www.nvidia.com/en-us/data-center/dgx-b200>
6. <https://www.nvidia.com/en-us/data-center/gb200-nvl72>
7. <https://developer.nvidia.com/blog/inside-nvidia-blackwell-ultra-the-chip-powering-the-ai-factory-era>
8. <https://www.amd.com/en/products/accelerators/instinct/mi350/mi355x.html>
9. <https://jax-ml.github.io/scaling-book/inference>
10. <https://arxiv.org/abs/2211.05102>
11. <https://arxiv.org/abs/2407.21783>
12. <https://developer.nvidia.com/blog/introducing-nvfp4-for-efficient-and-accurate-low-precision-inference>
13. <https://arxiv.org/abs/2412.19437>
14. <https://huggingface.co/deepseek-ai/DeepSeek-V3/raw/main/config.json>
15. <https://www.nvidia.com/en-us/data-center/gb300-nvl72>
16. <https://www.nvidia.com/en-us/data-center/rtx-pro-6000-blackwell-server-edition>
17. <https://www.nvidia.com/en-us/products/workstations/dgx-spark>
18. <https://www.nvidia.com/en-us/data-center/dgx-b300>
19. <https://www.databricks.com/blog/llm-inference-performance-engineering-best-practices>

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.