

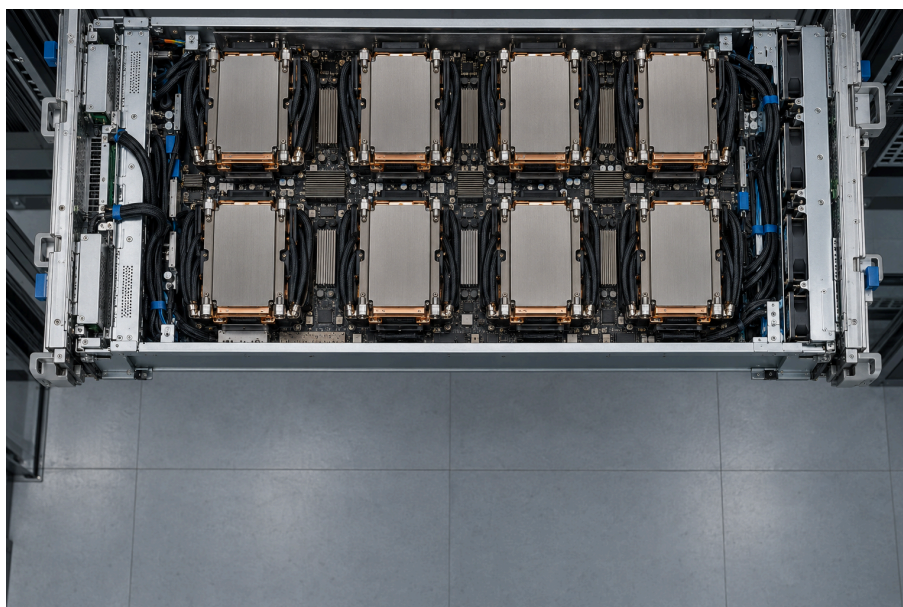


Estimating TTFT From FLOPs: The Prefill Compute Floor

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 · ttft llm · ttft itl · dgx spark prefill speed · how to estimate time to first token from gpu flops · prefill compute bound vs decode memory bound · llm prefill flops formula $2 \times \text{parameters} \times \text{tokens}$ · mfu prefill h100 realistic

Original article: <https://gpusmith.com/articles/en/prefill-compute-flops-ttft-floor>



In this article

1. Why TTFT has a compute floor
 2. The core formula
 3. Datasheet trap: dense vs sparse TFLOPS
 4. Choosing an MFU assumption
 5. Worked example 1: 8B dense, 8K prompt, one H100
 6. Worked example 2: 70B dense, 32K prompt, TP=8
 7. Worked example 3: MoE with active parameters
 8. Queueing, chunked prefill and batching effects
 9. Prefix cache hits lower the floor
 10. Validate with a benchmark
 11. Checklist and spreadsheet formula
 12. Sources
-

Why TTFT has a compute floor

Time to first token (TTFT) for a long prompt is dominated by prefill: the model has to run every prompt token through every layer before it can sample the first output token. Prefill processes those tokens in parallel, so it is limited by arithmetic, not by memory bandwidth. That gives a useful lower bound you can compute before you buy or benchmark anything:

TTFT floor $\approx (2 \times \text{matmul parameters} \times \text{prompt tokens} + \text{attention FLOPs}) \div (\text{dense peak FLOPS} \times \text{MFU})$

Queueing, scheduling, network and tokenization sit on top of this number. A measured TTFT below the floor at a plausible MFU means a cache hit, a different precision or a different clock boundary; one far above it points to queueing or interference. For what TTFT measures and how to report it, see the [TTFT, ITL and goodput definitions](#); this piece covers the arithmetic.

The Sarathi-Serve authors state the asymmetry: "Prefill iterations have high latency but saturate GPU compute due to parallel processing of the input prompt," while decode iterations "have low latency but also low compute utilization" ([Sarathi-Serve, arXiv 2403.02310](#)).

You can check this with datasheet numbers. An H100 SXM lists 3.35 TB/s of memory bandwidth and 1,979 teraFLOPS of BF16 Tensor Core throughput with sparsity, so 989.5 dense ([NVIDIA H100](#)). The ridge point is $989.5 / 3.35 \approx 295$ FLOPs per byte. In prefill each BF16 weight (2 bytes) is read once and used for 2 FLOPs per prompt token, so arithmetic intensity roughly equals the number of tokens in the batch: above about 300 tokens, dense prefill is compute-bound on H100. FP8 doubles both peak and FLOPs per weight byte, so the crossover stays put.

The core formula

Linear layers: 2 FLOPs per parameter per token

Each weight does one multiply-accumulate (2 FLOPs) per token. Kaplan et al. write the forward-pass cost as $C_{\text{forward}} \approx 2N + 2 \cdot n_{\text{layer}} \cdot n_{\text{ctx}} \cdot d_{\text{model}}$ per token, "where the factor of two comes from the multiply-accumulate operation used in matrix multiplication" (Kaplan et al., arXiv 2001.08361). Here N counts non-embedding parameters.

For prefill, count only weights multiplied for every prompt token: exclude the embedding lookup, exclude the LM head (logits are needed only for the last position; under 0.1% here), and for mixture-of-experts use **active** parameters. Computed from the published Llama 3 hyperparameters (layers, model dimension, FFN dimension, 32/64 query heads, 8 key/value heads; Llama 3 Herd of Models, Table 3), the matmul parameter counts are:

Model	Per layer (attention projections + SwiGLU FFN)	Layers	Matmul params
Llama 3 8B	41.9M + 176.2M = 218.1M	32	6.98B
Llama 3 70B	151.0M + 704.6M = 855.6M	80	68.45B

Using the nominal 8B would overstate the linear term by about 15%.

Attention: the quadratic term, derived

The Q, K, V and O projections are inside N . Not in N are the two activation-by-activation matmuls, $S = QK^T$ and $O = S \cdot V$. For one layer, s prompt tokens and query width d_{attn} (query heads $\times$ head dimension):

- QK^T : $s \times s$ outputs, each a dot product of length d_{attn} : $2 \cdot s^2 \cdot d_{\text{attn}}$ FLOPs.
- $S \cdot V$: $s \times d_{\text{attn}}$ outputs, each a dot product of length s : $2 \cdot s^2 \cdot d_{\text{attn}}$ FLOPs.

That is $4 \cdot s^2 \cdot d_{\text{attn}}$ per layer without masking. FlashAttention-2 uses exactly this count, " $4 \cdot \text{seqlen}^2 \cdot \text{head dimension} \cdot \text{number of heads}$," and divides it by 2 under a causal mask "to account for the fact that approximately only half of the entries are calculated" (FlashAttention-2, arXiv 2307.08691). For a causal decoder prompt across L layers:

Attention FLOPs (prefill, causal) $\approx 2 \cdot L \cdot s^2 \cdot d_{\text{attn}}$

This equals Kaplan's per-token term $2 \cdot n_{\text{layer}} \cdot n_{\text{ctx}} \cdot d_{\text{attn}}$ summed over s tokens with $n_{\text{ctx}} = s$. Grouped-query attention shrinks the KV cache, not these FLOPs, which scale with **query** heads (Llama 3 70B: $64 \times 128 = 8,192 = d_{\text{model}}$). If your kernel computes the masked half anyway, use $4 \cdot L \cdot s^2 \cdot d_{\text{attn}}$; sliding-window models need their own count.

When the attention term matters

The ratio of attention to linear FLOPs is $(2 \cdot L \cdot s^2 \cdot d) / (2 \cdot N \cdot s) = L \cdot s \cdot d / N$. It grows with the prompt and is larger for smaller models:

Prompt tokens	Llama 3 8B	Llama 3 70B	Mixtral 8x7B (active)
2,048	3.8%	2.0%	2.1%
8,192	15.4%	7.8%	8.5%
32,768	61.5%	31.4%	34.0%
131,072	246%	126%	136%

At 8K the term is minor for 70B; at 32K it is not; by 128K it dominates. Kaplan et al. call the context term relatively small while $d_{\text{model}} > n_{\text{ctx}}/12$ (below about 49K tokens at d_{model} 4,096); the table shows that can still mean 30-60% at 32K, so keep it.

Datasheet trap: dense vs sparse TFLOPS

NVIDIA's headline Tensor Core numbers assume 2:4 structured sparsity, which dense LLM inference does not use; take half. On the H100 page the Tensor Core rows, including BF16, FP16 and FP8, carry an asterisk, and the footnote reads "* With sparsity" ([NVIDIA H100](#), read 2026-10-28). The HGX page says it more plainly: "Specification in Sparse. Dense is $\frac{1}{2}$ sparse spec shown" ([NVIDIA HGX](#), read 2026-10-28).

GPU (as listed)	BF16 datasheet	BF16 dense	FP8 datasheet	FP8 dense	Memory BW
H100 SXM	1,979 TFLOPS*	989.5 TFLOPS	3,958 TFLOPS*	1,979 TFLOPS	3.35 TB/s
H100 NVL (PCIe)	1,671 TFLOPS*	835.5 TFLOPS	3,341 TFLOPS*	1,670.5 TFLOPS	3.9 TB/s
HGX B200 or B300, per GPU (8-GPU table $\div$ 8)	4.5 PFLOPS (36 $\div$ 8)	2.25 PFLOPS	9 PFLOPS (72 $\div$ 8)	4.5 PFLOPS	8 TB/s (DGX B200: 64 TB/s $\div$ 8)
GB200/GB300 NVL72, per GPU (rack $\div$ 72)	5 PFLOPS (360 $\div$ 72)	2.5 PFLOPS	10 PFLOPS (720 $\div$ 72)	5 PFLOPS	8 TB/s (576 TB/s $\div$ 72)

*Sparse. HGX B200 and HGX B300 list the same FP16/BF16 (36 PFLOPS) and FP8/FP6 (72 PFLOPS) figures, footnoted sparse; the NVL72 row divides the [GB200 NVL72](#) and [GB300 NVL72](#) rack figures by 72 (pages read 2026-10-28). Same silicon, different power envelope: quote the system you buy.

A published check: Llama 3 405B pre-training reports 430 TFLOPs per GPU at 43% BF16 MFU on H100 ([Llama 3, Table 4](#)). $430 / 989.5 = 43.5\%$, so Meta used the dense figure as the denominator. Divide by the sparse 1,979 and you would get 21.7%.

Use the sparse number and every floor here halves: a team budgets 67 ms for a prefill that needs at least 133 ms.

FP8 dense throughput is twice BF16, so FP8 GEMMs halve the linear-term floor; whether attention also runs in FP8 depends on the kernel, so the examples assume BF16 attention. Quality and capacity trade-offs are in [FP8 vs FP16 inference cost](#).

Choosing an MFU assumption

Model FLOPs utilization is "the ratio of the observed throughput to the theoretical maximum throughput if the benchmarked hardware setup were operating at peak FLOPS with no memory or communication overhead" (Pope et al., [arXiv 2211.05102](#)). Public anchors:

Source	Workload	Hardware	MFU
Pope et al., Table 2	PaLM 540B prefill, batch 1, 2,048 tokens, int8 weights	64 TPU v4	43% (0.29 s)
Pope et al., Table 2	PaLM 540B prefill, batch 512, bfloat16 weights	64 TPU v4	76% (85.2 s)
Llama 3, Table 4	405B training , BF16	8K-16K H100	38-43%
FlashAttention-2	GPT-style training , end to end	A100	up to 72%

The 76% is in the paper body, not only the abstract: weight-gathered layouts are inefficient at low batch sizes, "but eventually they become the most efficient at high batch sizes, achieving 76% MFU when the communication overhead is almost negligible." That batch is about a million prompt tokens ($512 \times 2,048$), not one user's TTFT; batch-1 prefill ran at 43%.

The formula reproduces Pope's batch-1 row (using the nominal 540B; the paper padded attention heads, adding 18B): $2 \times 540B \times 2,048 \text{ tokens} = 2.21 \times 10^{15} \text{ FLOPs}$. Spread over 64 chips $\times$ 275 TFLOPS bfloat16 at 43%, that is 0.29 s, matching Table 2 (a TPU check, not a GPU one).

Training MFU is not inference MFU: it includes backward passes and pipeline bubbles at a steady large batch, while single-request prefill loses time to launch gaps, non-matmul work, collectives and, for MoE, uneven expert loads.

Planning bracket used below: 40% (cautious) and 60% (optimistic) MFU for dense prefill of several thousand tokens. This is an assumption, not a measurement; expect less below about 2K tokens or with heavy sharding, and measure your stack.

Worked example 1: 8B dense, 8K prompt, one H100

Assumptions: Llama 3 8B, BF16, $s = 8,192$, one idle H100 SXM, no cache hit.

1. Linear FLOPs = $2 \times 6.98 \times 10^9 \times 8,192 = \mathbf{1.144 \times 10^{14}}$
2. Attention FLOPs = $2 \times 32 \text{ layers} \times 8,192^2 \times 4,096 = \mathbf{1.759 \times 10^{13}}$ (15.4% of linear)
3. Total = **1.319×10^{14} FLOPs**
4. Dense peak = $989.5 \times 10^{12} \text{ FLOP/s}$

MFU	Floor
100% (hard ceiling)	133 ms
60%	222 ms
40%	333 ms

MFU	Floor
100% of the <i>sparse</i> 1,979 figure (wrong)	67 ms

Memory check: reading 13.96 GB of weights at 3.35 TB/s takes about 4.2 ms, so compute binds. With FP8 GEMMs and BF16 attention the 100% ceiling becomes $57.8 + 17.8 = 76$ ms; attention is now a quarter of the time, which is why FP8 helps long prompts less than 2×.

Worked example 2: 70B dense, 32K prompt, TP=8

Assumptions: Llama 3 70B, BF16, $s = 32,768$, eight H100 SXM at TP=8 (141 GB of weights do not fit one 80 GB card).

1. Linear FLOPs = $2 \times 68.45 \times 10^9 \times 32,768 = \mathbf{4.486 \times 10^{15}}$
2. Attention FLOPs = $2 \times 80 \times 32,768^2 \times 8,192 = \mathbf{1.407 \times 10^{15}}$ (31% of linear)
3. Total = **5.893×10^{15} FLOPs**
4. Aggregate dense peak = $8 \times 989.5 = 7,916$ TFLOP/s

MFU	Floor
100%	0.74 s
60%	1.24 s
40%	1.86 s

An 8K prompt on the same node is 1.21×10^{15} FLOPs (0.38 s at 40%): four times the tokens cost 4.9 times the compute.

Tensor parallelism adds communication. A Megatron-style layout does two activation all-reduces per layer of $s \times d_{\text{model}} \times 2$ bytes ≈ 0.54 GB, so 85.9 GB over 80 layers; a ring all-reduce moves about $2(n-1)/n$ of that per GPU, ≈ 150 GB. NVIDIA's 900 GB/s NVLink figure counts both directions (18 links $\times$ 25 GB/s each way, per the [Hopper architecture post](#)), so at 450 GB/s per direction that is roughly 0.33 s if nothing overlaps compute. This rough illustration is why TP=8 prefill tends toward the low end of the bracket, and PCIe-only nodes lower still.

Worked example 3: MoE with active parameters

Assumptions: Mixtral 8x7B (32 layers, dim 4,096, FFN 14,336, 32 query heads, 8 KV heads, 8 experts, top-2; [Mixtral, Table 1](#)), $s = 8,192$, one H100 SXM, FP8 weights for GEMMs, BF16 attention.

Mixtral "has access to 47B parameters, but only uses 13B active parameters during inference." From the architecture, active matmul parameters are $32 \times (41.9\text{M attention} + 2 \times 176.2\text{M experts}) = \mathbf{12.62B}$.

1. Linear FLOPs = $2 \times 12.62 \times 10^9 \times 8,192 = \mathbf{2.067 \times 10^{14}}$, at FP8 dense 1,979 TFLOPS $\rightarrow$ 104.5 ms at 100%

2. Attention FLOPs = 1.759×10^{13} (same attention shape as the 8B example), at BF16 dense 989.5 TFLOPS $\rightarrow$ 17.8 ms at 100%
3. Floor = 122 ms at 100%, 204 ms at 60%, 306 ms at 40%

Two MoE caveats:

- **Memory follows total parameters, compute follows active.** 16,384 expert assignments ($8,192 \times \text{top-2}$) over 8 experts means every expert is read: about 46.7 GB in FP8, 14 ms at 3.35 TB/s. Still under the compute time, but with far less margin than a dense model; short prompts can turn memory-bound.
- **Routing costs utilization.** The authors note overhead "due to the routing mechanism and due to the increased memory loads when running more than one expert per device." Plan at the low end of the bracket until measured.

Queueing, chunked prefill and batching effects

The floor is one request on an idle GPU. Production adds three things.

Queueing. Divide prompt FLOPs arriving per second by dense peak $\times$ MFU. As a rule of thumb (our assumption, not a sourced threshold), past about 0.6-0.7 expect P99 TTFT well above the floor.

Chunked prefill. vLLM's documentation says chunked prefill "allows vLLM to process large prefills in smaller chunks and batch them together with decode requests," that in V1 it "is enabled by default whenever possible," and that the scheduler "batches all pending decode requests before scheduling any prefill operations" ([vLLM optimization and tuning, v0.32.0](#), released 2026-10-18). The token budget sets the trade-off: smaller `max_num_batched_tokens` values "(e.g., 2048) achieve better ITL," while "Higher values achieve better time to first token (TTFT)." With a 2,048-token budget a 32K prompt needs at least 16 iterations shared with decode work, all added to its TTFT.

Interference. DistServe frames prefill-decode interference as the reason to separate the phases and reports serving "7.4 $\times$ more requests or 12.6 $\times$ tighter SLO" against the systems it compared under its workloads ([DistServe](#), [arXiv 2401.09670](#)). That is a result on its own traces, not a general multiplier; see [disaggregated prefill/decode break-even](#).

Prefix cache hits lower the floor

Only uncached prompt tokens need prefill. vLLM's automatic prefix caching "caches the KV cache of existing queries" so a new query can "skip the computation of the shared part," and the docs say it "only reduces the time of processing the queries (the prefilling phase)" ([vLLM automatic prefix caching](#)).

New tokens still attend to cached ones: a token at position i costs about $4 \cdot L \cdot d \cdot i$ attention FLOPs. Example 2 with 28,672 cached tokens and 4,096 new:

1. Linear = $2 \times 68.45 \times 10^9 \times 4,096 = 5.61 \times 10^{14}$
2. Attention = $\sum 4 \times 80 \times 8,192 \times i$ for $i = 28,672 \dots 32,767 \approx 3.30 \times 10^{14}$
3. Total 8.91×10^{14} , which is 15% of the uncached 5.89×10^{15} . That is 0.23 s at 50% MFU, versus 1.49 s uncached.

The expected cache-hit length is often a more important input than the prompt length.

Validate with a benchmark

The formula is a hypothesis; a benchmark gives your stack's real MFU.

1. Isolate prefill: one request at a time, fixed prompt length, tiny output. vLLM's benchmark exposes `--dataset-name random`, `--random-input-len`, `--random-output-len`, `--max-concurrency` and `--percentile-metrics` (which can include `ttft`) ([vLLM bench serve](#)).
2. Prevent cache hits: random prompts with `--random-prefix-len 0`, or serve with `--no-enable-prefix-caching`.
3. Sweep 2K, 8K and 32K prompts to expose the quadratic term.
4. Implied MFU = $\text{FLOPs(s)} \div (\text{median TTFT} \times \text{dense peak})$, after subtracting a short-prompt baseline for network and tokenization.

Implied MFU above about 80% suggests a cache hit, a different precision or a smaller model. Around 20% (2× your 40% floor) usually means unoverlapped TP communication, a chunking token budget, weak attention kernels at that length, or queueing inside the clock. MFU falling with prompt length points at attention kernels. The same method turns the [DGX Spark prefill speed](#) figures in our review into implied MFU once precision and dense peak are fixed.

Checklist and spreadsheet formula

Inputs: L , d_{attn} , matmul parameters N (active, no embeddings), precisions, GPUs in the TP group, dense peak per precision, MFU, prompt length s and cached length c (0 if none).

```
new      = s - c
linear   = 2 * N * new
attn     = 2 * L * d_attn * (s^2 - c^2)      // causal; ≈ Σ 4·L·d·i for i = c..s
floor_s  = linear / (GPUs * dense_linear * MFU) + attn / (GPUs * dense_attn * MFU)
```

Red flags:

- Peak TFLOPS taken from an asterisked datasheet line.
- Total MoE parameters, or nominal size with embeddings.
- Training MFU applied to single-request TTFT.
- TTFT without prompt length, cache state, concurrency or clock boundary.
- No attention term above about 16K tokens.

GPU Smith runs this kind of prefill sizing as part of hardware selection work.

Sources

1. [NVIDIA H100 Tensor Core GPU](#), NVIDIA, product page (accessed 2026-10-28)

2. [NVIDIA HGX Platform](#), NVIDIA, product page (accessed 2026-10-28)
3. [Scaling Laws for Neural Language Models](#), Kaplan et al., arXiv, January 2020 (accessed 2026-10-28)
4. [FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning](#), Tri Dao, arXiv, July 2023 (accessed 2026-10-28)
5. [The Llama 3 Herd of Models](#), Llama Team, Meta, arXiv, July 2024 (accessed 2026-10-28)
6. [Efficiently Scaling Transformer Inference](#), Pope et al., arXiv, November 2022 (accessed 2026-10-28)
7. [Mixtral of Experts](#), Jiang et al., arXiv, January 2024 (accessed 2026-10-28)
8. [Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve](#), Agrawal et al., arXiv, March 2024 (accessed 2026-10-28)
9. [DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving](#), Zhong et al., arXiv, January 2024 (accessed 2026-10-28)
10. [Optimization and Tuning](#), vLLM documentation, v0.32.0 (released 2026-10-18; accessed 2026-10-28)
11. [Automatic Prefix Caching](#), vLLM documentation, v0.32.0 (accessed 2026-10-28)
12. [vllm bench serve](#), vLLM documentation, v0.32.0 (accessed 2026-10-28)
13. [NVIDIA GB200 NVL72 and GB300 NVL72](#), NVIDIA, product pages (accessed 2026-10-28)
14. [NVIDIA Hopper Architecture In-Depth](#), NVIDIA Technical Blog, 2022 (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://arxiv.org/abs/2403.02310>
2. <https://www.nvidia.com/en-us/data-center/h100>
3. <https://arxiv.org/abs/2001.08361>
4. <https://arxiv.org/abs/2407.21783>
5. <https://arxiv.org/abs/2307.08691>
6. <https://www.nvidia.com/en-us/data-center/hgx>
7. <https://www.nvidia.com/en-us/data-center/gb200-nvl72>
8. <https://www.nvidia.com/en-us/data-center/gb300-nvl72>
9. <https://arxiv.org/abs/2211.05102>
10. <https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth>
11. <https://arxiv.org/abs/2401.04088>
12. <https://docs.vllm.ai/en/v0.32.0/configuration/optimization>
13. <https://arxiv.org/abs/2401.09670>
14. https://docs.vllm.ai/en/v0.32.0/features/automatic_prefix_caching
15. <https://docs.vllm.ai/en/v0.32.0/cli/bench/serve>

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.