



# Long-Context LLM Serving Hardware: 128K to 1M Tokens

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 · long context llm inference hardware requirements · 1m token context gpu memory · kv cache 128k context memory · context parallelism inference vllm · decode context parallel dcp · which gpu for very long context windows

Original article: <https://gpusmith.com/articles/en/long-context-serving-hardware-128k-to-1m-tokens>



## In this article

1. What changes between 8K, 128K and 1M tokens
  2. KV cache arithmetic at long context
  3. Prefill compute and TTFT at 128K and beyond
  4. Memory-bound decode at long context
  5. Tensor parallel and KV duplication
  6. Decode context parallel (DCP)
  7. Prefill context parallel and ring attention
  8. Offload tiers
  9. Hardware choices
  10. Cost and concurrency model
  11. Decision tree
  12. Sources
- 

## What changes between 8K, 128K and 1M tokens

Moving from 8K to 128K or 1M tokens changes three costs at different rates:

1. **KV cache memory grows linearly with tokens.** Every resident token stores a key and a value vector in every attention layer.
2. **Prefill compute grows quadratically in the attention layers.** Meta explained Llama 3's late long-context training this way: "the compute in self-attention layers grows quadratically in the sequence length" (Llama 3 paper, §3.4.2).
3. **Decode becomes a KV-read problem.** Each generated token reads its sequence's entire cache. At long context that read outweighs the weights, and batching does not amortize it.

Concurrency collapses from hundreds of chat sessions per node to dozens at 128K and a handful at 1M, each costing more per token. The questions become HBM bytes per GPU, HBM bandwidth, and NVLink domain size, mapped below onto vLLM's tensor parallel (TP), decode context parallel (DCP), prefill context parallel (PCP) and KV offload. For sizing against your own traffic distribution, use the [KV cache memory calculator](#).

## KV cache arithmetic at long context

For standard attention, cache bytes per token are:

```
KV bytes/token = 2 (K and V) x layers x KV heads x head_dim x bytes per element
```

The number that matters is **KV heads**, not query heads. Grouped-query attention (GQA) uses "an intermediate (more than one, less than number of query heads) number of key-value heads" and the GQA authors report that uptrained GQA reaches "quality close to multi-head attention with comparable speed to MQA" ([Ainslie et](#)

al., [arXiv 2305.13245](#)). Llama 3 8B, 70B and 405B all use 8 KV heads, with 32, 80 and 126 layers respectively, and model dimensions of 4,096, 8,192 and 16,384 ([Llama 3 paper](#), [Table 3](#)). Their head dimension is 128 (8,192 / 64 query heads for 70B; 16,384 / 128 for 405B).

Applying the formula with BF16 cache (2 bytes per element). This is our arithmetic from the published hyperparameters:

| Model (KV geometry)                          | KV bytes per token      | 8K tokens | 128K tokens | 1M tokens (1,048,576) |
|----------------------------------------------|-------------------------|-----------|-------------|-----------------------|
| Llama 3 8B (32 L, 8 KV heads)                | 131,072 B (128 KiB)     | 1.0 GiB   | 16 GiB      | 128 GiB               |
| Llama 3 70B (80 L, 8 KV heads)               | 327,680 B (320 KiB)     | 2.5 GiB   | 40 GiB      | 320 GiB               |
| Llama 3 405B (126 L, 8 KV heads)             | 516,096 B (504 KiB)     | 3.94 GiB  | 63 GiB      | 504 GiB               |
| Hypothetical 70B with full MHA (64 KV heads) | 2,621,440 B (2,560 KiB) | 20 GiB    | 320 GiB     | 2,560 GiB             |
| Qwen2.5-7B-Instruct-1M (28 L, 4 KV heads)    | 57,344 B (56 KiB)       | 0.44 GiB  | 7 GiB       | 56 GiB                |

Llama 3 supports "a context window of up to 128K tokens" ([Llama 3 abstract](#)), so its 1M column is arithmetic only. The Qwen row uses the published config (28 layers, 4 KV heads, hidden size 3,584 over 28 query heads) ([config.json](#)); its card claims "Full 1,010,000 tokens and generation 8192 tokens" ([Qwen2.5-7B-Instruct-1M model card](#)).

GQA buys a constant factor, not a different curve: the 64-to-8 head reduction cuts the 70B cache 8x at every length, but one 128K request still uses as much cache as sixteen 8K requests. Latent attention compresses further; DeepSeek-V2's Multi-head Latent Attention compresses the KV cache "into a latent vector" and reports a 93.3% KV reduction relative to DeepSeek 67B ([DeepSeek-V2](#), [arXiv 2405.04434](#)). Architecture choice is the cheapest long-context hardware decision you will make.

Sanity check: the Qwen 7B cache at 1,010,000 tokens is 57,344 x 1,010,000 = 57.9 GB; add 7.61B BF16 parameters (about 15.2 GB) and prefill buffers, and the card's "At least 120GB VRAM (total across GPUs)" for 1M tokens is plausible ([model card](#)).

Because vLLM's PagedAttention reaches "near-zero waste in KV cache memory" ([Kwon et al.](#), [arXiv 2309.06180](#)), engine capacity tracks this formula once weights and runtime buffers are subtracted; the [vLLM KV capacity planner](#) covers reading the engine's reported pool.

## Prefill compute and TTFT at 128K and beyond

Kaplan et al. give forward cost per token as  $C_{\text{forward}} = 2N + 2 \times n_{\text{layer}} \times n_{\text{ctx}} \times d_{\text{attn}}$  ( $N$  = non-embedding parameters; the second term is attention against the context), noting that for "contexts and models with  $d_{\text{model}} > n_{\text{ctx}}/12$ , the context-dependent computational cost per token is a relatively small fraction of the total compute" ([Kaplan et al.](#), [arXiv 2001.08361](#), [Table 1](#) and [§2.1](#)). For  $d_{\text{model}} = 8,192$  that boundary is  $n_{\text{ctx}} = 98,304$ ; long-context serving lives beyond it.

For a causal prefill of  $T$  tokens, token  $t$  attends to  $t$  predecessors, so summing the per-token cost gives approximately:

$$\text{Prefill FLOPs} \approx 2 \times N \times T + n_{\text{layer}} \times d_{\text{attn}} \times T^2$$

GQA does not shrink the quadratic term, because  $d_{\text{attn}}$  is the query width: GQA saves memory, not attention FLOPs. The two terms cross at  $T = 2N / (n_{\text{layer}} \times d_{\text{attn}})$ .

Worked example, a Llama 3 70B-class model ( $N \sim 70\text{e}9$ , 80 layers,  $d_{\text{attn}} 8,192$ , so  $n_{\text{layer}} \times d_{\text{attn}} = 655,360$ ). This is our arithmetic:

| Prompt length T | Linear term 2NT | Attention term $L \cdot d \cdot T^2$ | Total   | Attention share |
|-----------------|-----------------|--------------------------------------|---------|-----------------|
| 8,192           | 1.15e15         | 4.40e13                              | 1.19e15 | 3.7%            |
| 131,072         | 1.84e16         | 1.13e16                              | 2.96e16 | 38%             |
| 1,048,576       | 1.47e17         | 7.21e17                              | 8.67e17 | 83%             |

Crossover:  $2 \times 70\text{e}9 / 655,360 \sim 213,600$  tokens. Beyond roughly 200K tokens, attention is most of the prefill work for this model shape.

Converting to a TTFT floor: the H200 SXM is listed at 1,979 TFLOPS BF16 and 3,958 TFLOPS FP8, both "With sparsity" ([NVIDIA H200](#)), so dense peaks are half of that: 989.5 and 1,979 TFLOPS. Across eight GPUs at 100% of dense BF16 peak:

- 128K prompt:  $2.96\text{e}16 / (8 \times 989.5\text{e}12) = \mathbf{3.7\ s}$
- 1M prompt:  $8.67\text{e}17 / (8 \times 989.5\text{e}12) = \mathbf{110\ s}$

These are floors; achieved utilization is below peak and TTFT scales up in proportion. A 1M-token prompt on one 8-GPU H200 node takes minutes unless the architecture avoids full attention. The Qwen 1M model card reports that its sparse-attention vLLM fork "achieves a 3 to 7 times speedup for sequences up to 1M tokens" ([model card](#)). That is a vendor claim, and as of 2026-10-28 the card still requires building QwenLM's `dev/dual-chunk-attn` vLLM branch. Upstream vLLM merged a dual-chunk flash-attention backend in May 2025 (PR #11844) for its since-removed V0 engine; the v0.32.0 backend registry has no dual-chunk backend, only the dual-chunk RoPE, so the card's sparse-attention path still needs that branch.

Chunked prefill does not reduce these FLOPs. It slices them so that decodes of other requests keep running in between. vLLM chunks prefills "based on the remaining `max_num_batched_tokens`" ([vLLM engine args](#)), and the Qwen card recommends `--max-num-batched-tokens 131072` with `--enable-chunked-prefill` for 1M serving. For how TTFT and inter-token latency (ITL) interact under load, see [TTFT and ITL metrics](#).

## Memory-bound decode at long context

A decode step for a batch reads the model weights once, then reads each sequence's entire KV cache. At long context the second read dominates and grows with batch size. Worked example on 8x H200 (aggregate  $8 \times 4.8\text{ TB/s} = 38.4\text{ TB/s}$  ([NVIDIA H200](#))), 70B weights in FP8 ( $\sim 70\text{ GB}$ ) and BF16 KV at 327,680 bytes per token. This is our arithmetic, a bandwidth floor that ignores compute and kernel efficiency:

| Batch x context | Bytes read per step              | Step time floor | Per-sequence tok/s | Aggregate tok/s |
|-----------------|----------------------------------|-----------------|--------------------|-----------------|
| 1 x 128K        | $70 + 42.9 = 112.9$ GB           | 2.94 ms         | 340                | 340             |
| 16 x 8K         | $70 + 16 \times 2.68 = 112.9$ GB | 2.94 ms         | 340                | 5,440           |
| 16 x 128K       | $70 + 16 \times 42.9 = 757$ GB   | 19.7 ms         | 51                 | 811             |
| 22 x 128K       | 1,015 GB                         | 26.4 ms         | 38                 | 832             |
| 352 x 8K        | 1,015 GB                         | 26.4 ms         | 38                 | 13,318          |

The last two rows read the same bytes per step because both fill the same KV budget, yet the 128K workload produces **16x fewer tokens per second from the same hardware**. The bill scales with resident tokens, not requests. When HBM is full of KV, step time approaches HBM capacity divided by HBM bandwidth:

- H100 SXM: 80 GB / 3.35 TB/s ~ 23.9 ms ([NVIDIA H100](#))
- H200 SXM: 141 GB / 4.8 TB/s ~ 29.4 ms
- B200 (DGX B200): 1,440 GB / 64 TB/s per 8 GPUs ~ 22.5 ms ([NVIDIA DGX B200](#))
- GB200 superchip: 372 GB / 16 TB/s ~ 23.3 ms ([NVIDIA GB200 NVL72](#))
- MI355X: 288 GB / 8 TB/s ~ 36 ms ([AMD MI355X](#))

Capacity sets how many long sessions fit; bandwidth sets the ITL they get. More capacity without proportionally more bandwidth makes a full GPU slower per token.

## Tensor parallel and KV duplication

TP shards the KV cache along the head dimension H. Because H is fixed by the architecture, vLLM warns that "when we continue to increase the tensor parallel size, the KV cache for each GPU will be duplicated for  $tp\_size / H$  times" ([vLLM Context Parallel Deployment](#)).

| Model                                 | KV heads (H) | TP | Duplication (tp/H) |
|---------------------------------------|--------------|----|--------------------|
| Llama 3 70B-class                     | 8            | 8  | 1x (none)          |
| Llama 3 70B-class                     | 8            | 16 | 2x                 |
| Qwen3-235B-A22B (per vLLM guide)      | 4            | 8  | 2x                 |
| DeepSeek-R1 with MLA (per vLLM guide) | 1            | 8  | 8x                 |
| Kimi-K2 (per vLLM guide)              | 1 (MLA)      | 16 | 16x                |

The last three rows are vLLM's own case studies. Duplication silently divides effective KV capacity: an 8-GPU MLA deployment at TP 8 holds one-eighth of the context it could.

## Decode context parallel (DCP)

DCP shards the KV cache along the token dimension instead. In vLLM you add `-dcp <size>`; "size does not increase the number of GPUs we need to launch, but just reduces the KV cache duplication", it "should lie in the range of  $[1, \text{tp\_size}/H]$ ", and larger values cut duplication at higher communication cost ([vLLM Context Parallel Deployment](#)). The flag is `--decode-context-parallel-size` (default 1) ([vLLM engine args](#)).

- **Sharding is interleaved** along T; the knob is now `--cp-kv-cache-interleave-size`, replacing `--dcp-kv-cache-interleave-size` ([vLLM engine args](#)).
- **Block accounting changes.** "a physical block of 16 tokens with DCP=4 represents 64 tokens", reported as `effective_attention_block_size` ([vLLM guide](#)). Capacity dashboards that multiply physical blocks by block size will under-report by the DCP factor.
- **Keep DCP inside the NVLink domain where possible.** For Kimi-K2 at TP 16, vLLM suggests `-dcp 8` as a compromise that leaves 2x duplication but keeps "DCP communication only ... inside one node".
- **Order:** "try to increase `-tp` size until you get satisfactory performance, and then add `-dcp`." DCP supports "both MLA and GQA models."

Worked example: an MLA model at TP 8 with a 300 GB per-node KV budget holds  $300 / 8 = 37.5$  GB of unique KV, because every GPU stores a full copy of its share. With `-dcp 8` the same node holds 300 GB of unique KV, 8x more resident context with no new hardware.

## Prefill context parallel and ring attention

DCP does nothing for the quadratic prefill floor; for that the sequence itself must be split across GPUs. vLLM describes two strategies ([vLLM Context Parallel Deployment](#)):

1. **Partial query, full key/value.** Each GPU computes Q/K/V for its chunk, K/V are gathered, and each GPU computes attention for its own queries. This works when the full K/V of one request still fits per GPU.
2. **Partial query, partial key/value.** When the request is "too long", K/V stay sharded and are passed chunk by chunk using ring attention.

Ring Attention overlaps "the communication of key-value blocks with the computation of blockwise attention", supports sequences "up to device count times longer" than prior memory-efficient Transformers, and reports "millions of tokens context size" (Liu et al., [arXiv 2310.01889](#)). Llama 3's training stack instead all-gathers K and V, accepting exposed latency for attention-mask flexibility ([Llama 3 paper](#), §3.3.2).

Status in vLLM v0.32.0 (released 2026-10-18): the deployment guide still says of both prefill strategies, "Both approaches are under active development", while the engine arguments include `--prefill-context-parallel-size` / `-pcp`, which "expands the process world size but does not increase the KV-cache shard count" ([vLLM engine args](#)). Treat PCP as something to qualify on your pinned version, not as a default production path.

Prefill CP moves K/V for the whole sequence in every layer. On NVLink (900 GB/s per H200; 1.8 TB/s GPU-to-GPU on HGX B200/B300 ([NVIDIA HGX](#))) that can be overlapped; across scale-out NICs it is much harder to hide, which makes long-context prefill a strong argument for a larger NVLink domain.

## Offload tiers

vLLM exposes `--kv-offloading-size` ("Size of the KV cache offloading buffer in GiB"), with native or lmcache backends ([vLLM engine args](#)). What matters is the link the KV crosses:

| Tier                                        | Example capacity       | Link to GPU (as listed)                                                                                 | Time to move 42.9 GB (one 70B 128K context) |
|---------------------------------------------|------------------------|---------------------------------------------------------------------------------------------------------|---------------------------------------------|
| HBM (read in place)                         | 141 GB per H200        | 4.8 TB/s                                                                                                | 8.9 ms                                      |
| Grace LPDDR5X via NVLink-C2C (GH200, GB200) | up to 480 GB per Grace | 900 GB/s C2C ( <a href="#">Grace Hopper</a> ); LPDDR5X "Up to 512 GB/s" ( <a href="#">GB200 NVL72</a> ) | 48 ms at 900 GB/s; 84 ms at 512 GB/s        |
| x86 host DRAM over PCIe Gen5                | platform-dependent     | "PCIe Gen5: 128GB/s" (NVIDIA's listed figure)                                                           | 335 ms or more                              |

Our arithmetic (bytes / listed bandwidth); real one-direction PCIe throughput is lower.

**Offload is for parking, not streaming.** Reading a 1M-token 70B-class context (343.6 GB) from LPDDR5X at 512 GB/s takes 0.67 s per pass, unusable per decode step. It pays when idle sessions, shared prefixes or agent histories are parked and pulled back once, because **reloading beats recomputing**: a 128K prefill for the 70B example is at least 2.96e16 FLOPs, 15 s on one H200 at dense FP8 peak, versus about 48 ms over C2C. GH200 offers "up to 624GB of combined CPU and GPU fast memory" ([NVIDIA Grace Hopper](#)); a GB200 NVL72 rack lists "17 TB LPDDR5X | 14 TB/s" ([GB200 NVL72](#)).

## Hardware choices

Vendor-listed figures as of 2026-10-28. NVIDIA's H200 page carries the note "Preliminary specifications. May be subject to change."

| Platform     | HBM per GPU / unit                                      | HBM bandwidth                    | Scale-up domain         | Long-context fit                                          |
|--------------|---------------------------------------------------------|----------------------------------|-------------------------|-----------------------------------------------------------|
| H100 SXM     | 80 GB                                                   | 3.35 TB/s                        | 8 GPUs, NVLink 900 GB/s | 128K for 70B-class only at low concurrency                |
| H200 SXM     | 141 GB                                                  | 4.8 TB/s                         | 8 GPUs, NVLink 900 GB/s | 128K workhorse; 1M only for small or compressed-KV models |
| DGX/HGX B200 | 1,440 GB per 8 GPUs                                     | 64 TB/s per 8 GPUs               | 8 GPUs, NVLink 1.8 TB/s | More capacity and bandwidth per node; same 8-GPU domain   |
| HGX B300     | 2.1 TB per 8 GPUs (DGX B300 guide: 8 x 288 GB = 2.3 TB) | 8 TB/s per GPU (not on HGX page) | 8 GPUs, NVLink 1.8 TB/s | Most HBM per 8-GPU node in NVIDIA's HGX line              |

| Platform    | HBM per GPU / unit            | HBM bandwidth         | Scale-up domain                | Long-context fit                                           |
|-------------|-------------------------------|-----------------------|--------------------------------|------------------------------------------------------------|
| GB200 NVL72 | 372 GB per superchip (2 GPUs) | 16 TB/s per superchip | 72 GPUs, 130 TB/s NVLink       | Large DCP/PCP groups in one domain plus 17 TB LPDDR5X tier |
| AMD MI355X  | 288 GB                        | 8 TB/s                | 8-GPU Infinity Fabric platform | High capacity per GPU; lower bandwidth-to-capacity ratio   |

Figures are from the vendor pages linked above ([HGX](#) for B300 capacity, [NVIDIA's Blackwell Ultra post](#) for its bandwidth); the "fit" column is our judgment.

Ask which constraint binds first. **Capacity** binds when one request approaches a GPU group's KV budget: a 1M-token 70B-class BF16 cache (343.6 GB) cannot share one GB200 superchip's 372 GB with weights, so it needs DCP or FP8 KV. **Bandwidth** binds when many long sessions are resident. **Domain size** binds when  $TP \times DCP \times PCP$  needs more than 8 GPUs per request, which is the case for rack-scale NVLink.

## Cost and concurrency model

Concurrency per 8x H200 node for the 70B example, with our assumptions: 1,128 GB HBM total, 70 GB FP8 weights, and 10% of HBM (112.8 GB) reserved for activations, CUDA graphs and runtime, leaving 945 GB for KV. TP 8 equals the 8 KV heads, so there is no duplication.

| Context per request | KV per request (BF16) | Max resident (BF16 KV) | Max resident (FP8 KV) | Decode step floor at full |
|---------------------|-----------------------|------------------------|-----------------------|---------------------------|
| 8K                  | 2.68 GB               | 352                    | 704                   | 26.4 ms                   |
| 128K                | 42.9 GB               | 22                     | 44                    | 26.4 ms                   |
| 1M                  | 343.6 GB              | 2                      | 5                     | ~26 ms (bounded by HBM)   |

FP8 KV halves bytes per token. vLLM's `--kv-cache-dtype` accepts fp8 among other options ([vLLM engine args](#)). Validate accuracy at your target length before relying on it.

At full occupancy step time is roughly constant, so **cost per output token scales almost linearly with context**: a 128K session costs roughly 16x an 8K session per generated token, before prefill. Meter long-context tiers by resident token-seconds, not requests. Prefill adds at least 3.7 s of a full node per uncached 128K prompt; if prompts are mostly unique (document Q&A) rather than repeated prefixes (agent loops), prefill capacity may set the node count. When the two phases need different hardware, separating them becomes worth costing; see [disaggregated prefill/decode break-even](#).

## Decision tree

1. **Check the architecture first**: KV heads and attention type (GQA, MLA, sliding or sparse). Compressed KV or sparse attention changes every number above.
2. **Compute KV per request at p95 context** against HBM after weights and reserve. If it fits one GPU with headroom, use plain replicas; otherwise raise TP up to the number of KV heads.

3. **If TP exceeds KV heads** (always for MLA models, often for 4-8 KV-head GQA models), add `-dcp` up to `tp/H`. Keep DCP within one NVLink domain.
4. **If TTFT at p95 prompt length misses the SLO** after chunked prefill tuning, evaluate prefill context parallel on your pinned vLLM version, or disaggregate prefill onto its own pool.
5. **If sessions are long-lived or share prefixes**, add a CPU KV tier (`--kv-offloading-size`); prefer Grace-class C2C when reload latency matters.
6. **If one request needs more than 8 GPUs** to hold or prefill, budget for a rack-scale NVLink domain rather than stretching CP across InfiniBand.
7. **Price by resident tokens**, with separate SLOs and quotas for the long-context tier.

GPU Smith can help run this arithmetic against a specific model and traffic profile.

## Sources

1. vLLM, "Context Parallel Deployment" (v0.32.0 docs, released 2026-10-18, accessed 2026-10-28): [https://docs.vllm.ai/en/v0.32.0/serving/context\\_parallel\\_deployment/](https://docs.vllm.ai/en/v0.32.0/serving/context_parallel_deployment/)
2. vLLM, "Engine Arguments" (v0.32.0 docs, accessed 2026-10-28): [https://docs.vllm.ai/en/v0.32.0/configuration/engine\\_args/](https://docs.vllm.ai/en/v0.32.0/configuration/engine_args/)
3. Liu, Zaharia, Abbeel, "Ring Attention with Blockwise Transformers for Near-Infinite Context", arXiv 2310.01889: <https://arxiv.org/abs/2310.01889>
4. Ainslie et al., "GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints", arXiv 2305.13245: <https://arxiv.org/abs/2305.13245>
5. Llama Team, Meta, "The Llama 3 Herd of Models", arXiv 2407.21783 (abstract): <https://arxiv.org/abs/2407.21783>
6. Llama Team, Meta, "The Llama 3 Herd of Models", HTML v3 (Table 3, §3.3.2, §3.4.2): <https://arxiv.org/html/2407.21783v3>
7. Kaplan et al., "Scaling Laws for Neural Language Models", arXiv 2001.08361 (Table 1): <https://arxiv.org/abs/2001.08361>
8. Kwon et al., "Efficient Memory Management for Large Language Model Serving with PagedAttention", arXiv 2309.06180: <https://arxiv.org/abs/2309.06180>
9. DeepSeek-AI, "DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model", arXiv 2405.04434: <https://arxiv.org/abs/2405.04434>
10. Qwen, "Qwen2.5-7B-Instruct-1M" model card and config.json (accessed 2026-10-28): <https://huggingface.co/Qwen/Qwen2.5-7B-Instruct-1M>
11. NVIDIA, "H200 Tensor Core GPU" (accessed 2026-10-28): <https://www.nvidia.com/en-us/data-center/h200/>
12. NVIDIA, "H100 Tensor Core GPU" (accessed 2026-10-28): <https://www.nvidia.com/en-us/data-center/h100/>
13. NVIDIA, "DGX B200" (accessed 2026-10-28): <https://www.nvidia.com/en-us/data-center/dgx-b200/>
14. NVIDIA, "HGX Platform" (accessed 2026-10-28): <https://www.nvidia.com/en-us/data-center/hgx/>

15. NVIDIA, "GB200 NVL72" and "Grace Hopper Superchip" (accessed 2026-10-28):  
<https://www.nvidia.com/en-us/data-center/gb200-nvl72/> and <https://www.nvidia.com/en-us/data-center/grace-hopper-superchip/>
16. AMD, "Instinct MI355X" (accessed 2026-10-28):  
<https://www.amd.com/en/products/accelerators/instinct/mi350/mi355x.html>
17. NVIDIA Technical Blog, "Inside NVIDIA Blackwell Ultra: The Chip Powering the AI Factory Era" (2025-08-22, updated 2025-09-24; accessed 2026-10-28): <https://developer.nvidia.com/blog/inside-nvidia-blackwell-ultra-the-chip-powering-the-ai-factory-era>

## 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/html/2407.21783v3>
2. <https://arxiv.org/abs/2305.13245>
3. <https://arxiv.org/abs/2407.21783>
4. <https://huggingface.co/Qwen/Qwen2.5-7B-Instruct-1M/raw/main/config.json>
5. <https://huggingface.co/Qwen/Qwen2.5-7B-Instruct-1M>
6. <https://arxiv.org/abs/2405.04434>
7. <https://arxiv.org/abs/2309.06180>
8. <https://arxiv.org/abs/2001.08361>
9. <https://www.nvidia.com/en-us/data-center/h200>
10. [https://docs.vllm.ai/en/v0.32.0/configuration/engine\\_args](https://docs.vllm.ai/en/v0.32.0/configuration/engine_args)
11. <https://www.nvidia.com/en-us/data-center/h100>
12. <https://www.nvidia.com/en-us/data-center/dgx-b200>
13. <https://www.nvidia.com/en-us/data-center/gb200-nvl72>
14. <https://www.amd.com/en/products/accelerators/instinct/mi350/mi355x.html>
15. [https://docs.vllm.ai/en/v0.32.0/serving/context\\_parallel\\_deployment](https://docs.vllm.ai/en/v0.32.0/serving/context_parallel_deployment)
16. <https://arxiv.org/abs/2310.01889>
17. <https://www.nvidia.com/en-us/data-center/hgx>
18. <https://www.nvidia.com/en-us/data-center/grace-hopper-superchip>
19. <https://developer.nvidia.com/blog/inside-nvidia-blackwell-ultra-the-chip-powering-the-ai-factory-era>

## 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.