

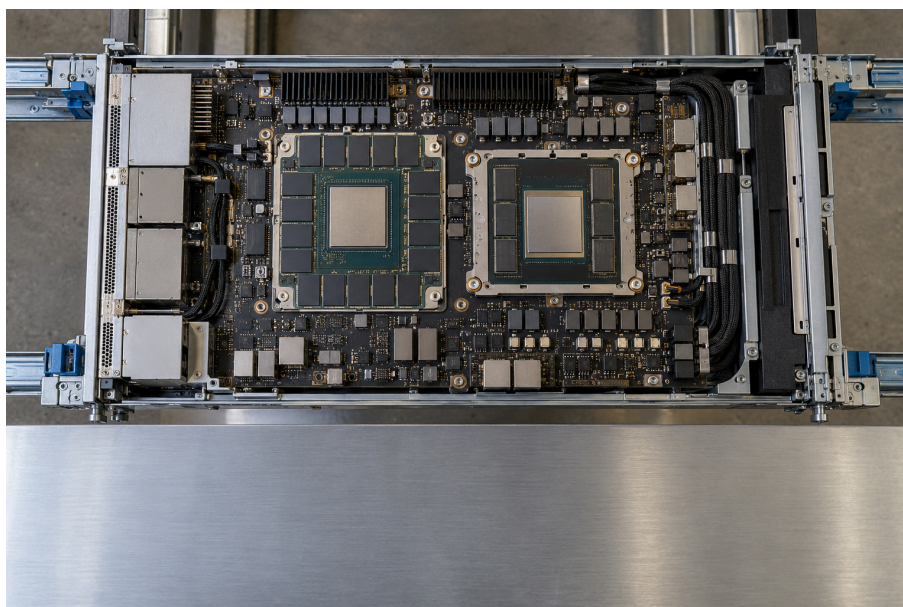


Prefix Caching and KV Cache Offload to CPU RAM and NVMe

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 · cpu offload · vllm automatic prefix caching · kv cache offload cpu nvme · lmcache vs dynamo kvbm · prefix cache hit rate economics llm · grace hopper kv cache offload nvlinc-c2c · sglang radixattention vs vllm prefix caching

Original article: <https://gpusmith.com/articles/en/prefix-caching-and-kv-cache-offload-cpu-nvme>



In this article

1. Why prefix reuse matters
 2. How automatic prefix caching works
 3. KV bytes per token: a worked example for Qwen2.5-72B
 4. Hit-rate economics
 5. When prefix caching does nothing
 6. Tiering beyond HBM: what each stack ships (October 2026)
 7. Bandwidth math: fetch vs recompute
 8. Grace Hopper and Grace Blackwell: where C2C changes the answer
 9. Operating it
 10. Security and tenancy
 11. Decision matrix
 12. Sources
-

Why prefix reuse matters

Chat, RAG and agent workloads keep resending the same long prefix: a system prompt, tool definitions, a retrieved document, the conversation so far. Without reuse, every turn pays full prefill on tokens the server has already processed. Prefix caching keeps the key/value (KV) tensors for those tokens and skips recomputing them. Offload extends that cache from HBM to CPU DRAM, local NVMe or a remote store, so more prefixes survive long enough to be hit again.

The short answer: reuse from HBM is nearly free, and fetching from CPU DRAM over PCIe Gen5 beats recomputing for almost any prefix worth caching. NVMe and remote tiers win only above a minimum prefix length, which depends on the model, the GPU and the storage path. On large GQA models that minimum is low. On small models and fast GPUs it can sit near 8k tokens or higher. The arithmetic below uses a named model so you can redo it for yours.

Prefix caching only speeds up prefill. vLLM's documentation says APC "only reduces the time of processing the queries (the prefilling phase)" and "does not reduce the time of generating new tokens (the decoding phase)" ([vLLM APC docs](#)).

How automatic prefix caching works

vLLM. Automatic Prefix Caching (APC) "caches the KV cache of existing queries", so a new query sharing a prefix can skip the shared part. It is enabled with `enable_prefix_caching=True` ([vLLM APC docs](#)). Under the hood, vLLM hashes each KV block from the parent block's hash, the block's tokens and extra keys such as LoRA IDs, multimodal input hashes and cache salts. It notes "We only cache full blocks", and freed blocks are evicted least-recently-used first. The default hash is sha256; the faster xxhash option is flagged as not cryptographically secure ([vLLM prefix caching design](#)). Because each block's hash includes its parent's, a block only matches when every token before it also matches.

SGLang. RadixAttention stores cached prefixes in a radix tree, so the longest shared prefix across many requests is found and reused automatically. The SGLang paper reports "up to 6.4x higher throughput compared to state-of-the-art inference systems" ([Zheng et al., arXiv 2312.07104](#)). That figure comes from the whole runtime on the paper's structured-program benchmarks, not from RadixAttention alone.

For operators the two behave alike: exact-prefix reuse, eviction under pressure, nothing kept beyond HBM without a lower tier. Your traffic's hit rate matters more than the engine.

KV bytes per token: a worked example for Qwen2.5-72B

The [Qwen2.5-72B-Instruct config.json](#) gives `num_hidden_layers: 80`, `num_key_value_heads: 8`, `num_attention_heads: 64`, `hidden_size: 8192` and `torch_dtype: bfloat16`. Head dimension is $8192 / 64 = 128$.

KV bytes per token = 2 (K and V) x layers x KV heads x head dim x bytes per element = $2 \times 80 \times 8 \times 128 \times 2 =$ **327,680 bytes (320 KiB) per token** at BF16, or 163,840 bytes with an FP8 KV cache.

A 32,768-token prefix (the context length in the shipped config) is therefore $32,768 \times 327,680 = 10.74$ GB. The derivation is the same as in our [vLLM KV cache capacity planning](#) article. For turning bytes into GPU counts, see the [KV cache memory calculator](#).

Recompute cost. The [model card](#) lists 72.7B parameters, 70.0B of them non-embedding. Estimates used throughout (labelled assumptions):

- Linear-layer FLOPs per prefill token are about $2 \times 70.0e9 = 140$ GFLOP.
- Attention score and value FLOPs for the token at position p are about 4×64 heads $\times 128 \times 80$ layers $\times p = 2,621,440 \times p$. Summed over a causal prefix of S tokens, that is about $2,621,440 \times S^2 / 2$.
- Hardware is 8x H100 SXM. NVIDIA lists 1,979 TFLOPS BF16 *with sparsity* ([NVIDIA H100](#)), so 989.5 TFLOPS dense per GPU. At an assumed 50% utilization, the node delivers $8 \times 989.5e12 \times 0.5 = 3.958e15$ FLOP/s.

For $S = 32,768$: linear $32,768 \times 140e9 = 4.59e15$, plus attention $2,621,440 \times 32,768^2 / 2 = 1.41e15$, total $5.99e15$ FLOP. $5.99e15 / 3.958e15 =$ **1.51 s of full-node prefill**.

Hit-rate economics

Define hit rate as the fraction of prompt tokens served from cache. The table models a 16,384-token prompt on the node above. The uncached suffix still attends over the cached prefix, so compute saved is always slightly less than the hit rate.

Prefix hit rate	Cached tokens	Prefill compute (s)	Compute saved	Prefill-bound requests/s per node	KV held per session (GB)
0%	0	0.668	0%	1.50	0
25%	4,096	0.518	22.5%	1.93	1.34

Prefix hit rate	Cached tokens	Prefill compute (s)	Compute saved	Prefill-bound requests/s per node	KV held per session (GB)
50%	8,192	0.356	46.7%	2.81	2.68
75%	12,288	0.184	72.5%	5.44	4.03
90%	14,745	0.075	88.8%	13.36	4.83

The request rate ignores decode. The curve is convex: moving from 75% to 90% more than doubles prefill capacity.

The cost side is memory. Each 90%-hit session pins about 4.8 GB of KV until it is evicted. Rough capacity for the same node:

- **HBM:** 8 x 80 GB at 90% utilization, minus 145.4 GB of BF16 weights (72.7B x 2), leaves about 430 GB. That is roughly 1.31M tokens, or about 80 prefixes of 16k tokens. Activations and fragmentation are ignored, so this is optimistic.
- **CPU DRAM:** 2 TB of host memory (an assumed configuration) holds about 6.1M tokens, or about 372 such prefixes.
- **NVMe:** 4 x 30.72 TB drives (the top [Micron 9550](#) capacity) hold about 375M tokens, or about 22,900 prefixes.

If more than about 80 sessions return within your typical reuse interval, an HBM-only cache evicts them before they come back and the hit rate collapses. Lower tiers exist to hold that working set.

When prefix caching does nothing

- **No shared prefix.** vLLM says APC brings no gain when "new queries do not share the same prefix with any of existing queries" ([vLLM APC docs](#)).
- **Decode-dominated work.** APC also brings no gain when the engine "spends most of the time generating answers".
- **Cache-busting templates.** A timestamp, request ID, user name or shuffled tool list near the top of the prompt changes the first block's hash, and every later block misses. Put static content first and per-request content last.
- **Truncation and rewriting.** Agents that summarize or drop old turns miss on everything after the first edit.

Tiering beyond HBM: what each stack ships (October 2026)

Read on 2026-10-28; these projects ship monthly, so re-check before deploying.

Stack	Tiers	How it is enabled	Notes
vLLM native OffloadingConnector	CPU pinned memory; with TieringOffloadingSpec, secondary fs, obj (NIXL OBJ) and p2p tiers	--kv-offloading-size <GiB> (total across TP ranks; --kv-offloading-backend native or lmcache), or --kv-transfer-config JSON with cpu_bytes_to_use	"Only the CPU primary tier has direct GPU access"; supports CUDA, ROCm and XPU; LRU or ARC eviction (vLLM KV offloading usage , page dated Oct 26, 2026)
SGLang HiCache	GPU (L1), host (L2), external storage (L3): file, mooncake, hf3fs, nixl, aibrix, custom dynamic	--enable-hierarchical-cache, --hichache-ratio or --hichache-size, --hichache-storage-backend	"L1 and L2 are private to a single inference instance; only L3 can be shared"; prefetch policies best_effort, wait_complete, timeout (SGLang HiCache best practices)
LMCache	CPU RAM, local SSD, Redis/Valkey, Mooncake, InfiniStore, S3-compatible object storage, NIXL, GDS	vLLM connector (--kv-offloading-backend lmcache); SGLang LMCacheUnifiedRadixCache (in SGLang v0.5.22, LMCache 0.5.6 of 2026-10-15)	Standalone multiprocess server; reuses KV across requests, sessions and engine instances (LMCache docs)
NVIDIA Dynamo KVBM	GPU (G1), host pinned (G2), disk (G3), S3-compatible object (G4)	DYN_KVBM_CPU_CACHE_GB, DYN_KVBM_DISK_CACHE_GB, DYN_KVBM_OBJECT_ENABLED	README marks KVBM supported for TensorRT-LLM and vLLM and under construction for SGLang (Dynamo README , tag v1.6.1, 2026-10-27)

First, Dynamo's configuration reference warns that if the CPU cache is smaller than the GPU KV cache, "KVBM churns (offloading blocks after every forward pass)". It suggests `DYN_KVBM_CPU_CACHE_GB >= 100` and a disk cache at least as large as the CPU cache ([Dynamo KVBM configuration](#), v1.6.1). Size the host tier as a multiple of HBM KV, not a fraction of it.

Second, vLLM's shorthand flags `--kv-offloading-size` (GiB) and `--kv-offloading-backend` have shipped since v0.11.1 (2025-11-18) and remain in v0.32.0 (2026-10-18), where native expands to the OffloadingConnector JSON above ([vLLM blog, Jan 8, 2026](#); `vllm/config/vllm.py` at v0.32.0). The usage page still documents only the JSON form.

Bandwidth math: fetch vs recompute

Fetching wins when moving cached KV takes less time than rebuilding it. Same model and node, with these transfer assumptions:

- **CPU DRAM to GPU:** about 50 GB/s per GPU in one direction, the H100 figure the vLLM team measured for its connector ([vLLM blog](#)), against NVIDIA's PCIe Gen5 rating of 128 GB/s ([NVIDIA H100](#)). With TP=8 each GPU holds one of the eight KV heads, so it loads one eighth of the bytes. We assume each GPU has its own x16 path to the host.
- **NVMe:** 14.0 GB/s sequential read per drive, a vendor spec ([Micron 9550](#)), with four drives striped ideally.

Prefix tokens	KV bytes	Recompute (s)	DRAM to GPU (ms)	1 NVMe read (s)	4 NVMe read (s)	Break-even bandwidth, whole node
1,024	0.34 GB	0.037	0.8	0.024	0.006	9.2 GB/s
4,096	1.34 GB	0.150	3.4	0.096	0.024	8.9 GB/s
8,192	2.68 GB	0.312	6.7	0.192	0.048	8.6 GB/s
16,384	5.37 GB	0.668	13.4	0.383	0.096	8.0 GB/s
32,768	10.74 GB	1.515	26.8	0.767	0.192	7.1 GB/s

Break-even bandwidth is KV bytes divided by recompute time: the slowest path that still ties. It falls as prefixes grow because attention FLOPs grow with S^2 while KV bytes grow with S . For this model, any path that sustains roughly 9 GB/s into the node, about 1.1 GB/s per GPU, wins *on paper*. Host DRAM wins by 44-56x. Even a single Gen5 NVMe wins, but NVMe data still has to cross PCIe into the GPU unless the two copies are pipelined.

That "on paper" carries weight. A September 2026 study by VU Amsterdam and IBM Research measured this tradeoff in vLLM on an H100 node ([Kanichai et al., arXiv 2609.11744](#)). Its findings:

- GPU-CPU copies sustained 54-56 GB/s, close to our assumption.
- For Llama 3.2 3B (114,688 bytes per token), the bandwidth needed to break even fell "from 23.2 GB/s at 1k tokens to 10.4 GB/s at 8k and 3.5 GB/s at 80k".
- An 8k prompt needed 77.8% prefix reuse to break even through the llm-d filesystem path. At 80k the threshold fell to 7.8%.
- "Below roughly 8k tokens ... no storage device we measured can load the prompt faster than the GPU rebuilds it."
- On Alibaba Bailian production-trace replays, the H100's average request fell below the SSD break-even point, so external caching gave no TTFT benefit there.

The authors conclude that external KV caching "should therefore be treated as a setup specific admission decision."

Two lessons for sizing:

1. **Model shape decides.** A 3B model on one GPU needs far more bandwidth per token of savings than a 72B model on eight GPUs, because each KV byte represents far less recomputation. Large GQA models, and MLA models with even smaller KV, are where storage tiers pay off soonest.
2. **Fixed overheads dominate short prefixes.** Lookups, file opens, staging and scheduling are invisible to bandwidth arithmetic. Calibrate the break-even length on your hardware and skip lower-tier loads below it.

The vLLM team's own single-request numbers fit this picture. For Llama-3.1-8B on H100, loading from CPU "reduces TTFT by X2-X22" depending on prompt size. With 10,000 requests of 512 tokens, throughput rose "by up to X9" with CPU hit rate ([vLLM blog](#)). These are project-run benchmarks, not independent results.

Throughput matters as much as latency: recomputed prefill takes GPU-seconds from other requests' decode, while a DMA copy mostly uses copy engines and the PCIe link. For pooled prefill, see our [disaggregated prefill/decode](#) analysis.

Grace Hopper and Grace Blackwell: where C2C changes the answer

On GH200, NVLink-C2C is rated at "up to 900 GB/s total bandwidth, 450 GB/s/dir". Each Grace CPU carries up to 512 GB of LPDDR5X at up to 546 GB/s, next to up to 96 GB of HBM3 at 3000 GB/s, and the GPU "can address up to 608 GB of memory within a superchip" ([NVIDIA Grace Hopper architecture blog, Nov 2022](#)). These are vendor link ratings, not measured KV transfers, for the original HBM3 part; NVIDIA's current [GH200 page](#) lists HBM3e variants with up to 624 GB of combined fast memory, so check your SKU.

Per GPU link, the difference is large. Moving the 10.74 GB, 32k-token prefix above into one GPU takes about 215 ms at a measured 50 GB/s over PCIe, against 23.9 ms at the theoretical 450 GB/s per direction over C2C. The tier also changes character: 512 GB of LPDDR5X holds about 1.56M tokens at 320 KiB per token (3.1M with FP8 KV), about five times the HBM. Host memory becomes a working cache tier rather than a slow spill area.

On GB200, each superchip pairs one Grace CPU with two Blackwell GPUs. It has up to 480 GB of LPDDR5X at up to 512 GB/s, and 372 GB of HBM3E at 16 TB/s ([NVIDIA GB200 NVL72](#)). Its NVLink-C2C interface "delivers 900 GB/s of bidirectional bandwidth" ([NVIDIA Technical Blog, Mar 18, 2024](#)). Two GPUs share that CPU memory bandwidth, so plan with about half of it per GPU when both fetch at once. That is still several times a PCIe Gen5 link; the tighter constraint is capacity, roughly 240 GB of LPDDR5X per GPU versus 512 GB on GH200.

Operating it

Metrics to export and alert on:

- **Hit rate by tier.** vLLM exposes `vllm:prefix_cache_queries` and `vllm:prefix_cache_hits` as counters, plus `vllm:kv_cache_usage_perc` ([vLLM metrics design](#)). Track connector hits separately.
- **TTFT split by hit and miss.** Bucket by cached-prefix length to see whether lower-tier hits actually beat recompute.
- **Churn.** Compare blocks written to lower tiers against blocks later read back; write-heavy, read-light means wasted bandwidth.
- **NVMe write volume.** At 0% hit, the 72B node above prefills about 24,600 tokens/s ($1.5 \times 16,384$). Writing all of it to disk is $24,600 \times 327,680$ B, about 8 GB/s or about 700 TB per day. Admission filters matter here. Dynamo's disk offload filter is on by default (`DYN_KVBM_DISABLE_DISK_OFFLOAD_FILTER` defaults to false) and its offload policies include frequency-based `presence_lfu`. vLLM's connector defaults `offload_prompt_only` to true ([vLLM KV offloading usage](#)). Compare the result against your drives' rated endurance.
- **Cold start.** HBM and DRAM tiers are empty after a restart. Treat a surviving disk or remote tier as valid only if engine version, weights, dtype, block size and hash settings are unchanged.

Routing. A cache on one replica helps only requests routed to that replica. Dynamo's KV-aware router "Routes requests based on worker load and KV cache overlap". NVIDIA's README credits it with "2x faster TTFT", attributed to a Baseten benchmark on Qwen3-Coder 480B ([Dynamo README](#)). That is a vendor-reported result. For support and version-pinning questions, see our [Dynamo support checklist](#). Without it, N replicas split each session across N caches; session affinity is the cheap fallback.

Security and tenancy

A shared prefix cache is a timing side channel: a faster first token tells tenant B that someone recently sent the same prefix. vLLM's answer is an optional per-request `cache_salt` that "is injected into the hash of the first block, ensuring that only requests with the same salt can reuse cached KV blocks". The docs say this "prevents timing-based attacks where an adversary could infer cached content by observing latency differences" ([vLLM prefix caching design](#)).

Salt per tenant by default and share only deliberately public prefixes. Keep the cryptographic default hash on multi-tenant servers. Treat a shared L3 tier (Redis, S3, a shared filesystem) as derived customer data, with encryption, access control and retention to match.

Decision matrix

Workload pattern	Typical reuse	Recommended tiering
Single-turn, unique prompts (batch classification, translation)	Near zero	APC on only for the shared system prompt; no offload
Short chat, prefixes under the calibrated break-even (often under 8k on small models)	Moderate, short-lived	HBM prefix cache plus session affinity; CPU tier optional
Long multi-turn chat or agents, many concurrent sessions	High, long gaps between turns	HBM plus CPU DRAM sized above HBM KV; KV-aware routing
RAG over a hot document set	High across users	CPU tier plus a shared L3 (LMCache, HiCache L3, KVBM G3/G4); admission by frequency
Very long contexts (64k+) on large GQA or MLA models	High	Add NVMe; even single-drive bandwidth beats recompute at this length
GH200/GB200 nodes	Any meaningful reuse	Treat LPDDR5X as the main cache tier; size it before adding NVMe

Whatever the stack: make prompts cache-friendly, measure real hit rate, size host memory above HBM KV, then add NVMe or remote tiers only where your calibrated break-even says loads beat recompute. GPU Smith helps teams run this kind of sizing on their own traffic and hardware.

Sources

1. [Automatic Prefix Caching](#), vLLM documentation, page dated Oct 13, 2026 (accessed 2026-10-28)
2. [Automatic Prefix Caching \(design\)](#), vLLM documentation (accessed 2026-10-28)

3. [KV Offloading usage](#), vLLM documentation, page dated Oct 26, 2026 (accessed 2026-10-28)
4. [Inside vLLM's New KV Offloading Connector](#), vLLM Blog (Or Ozeri, Danny Harnik, IBM Research), Jan 8, 2026 (accessed 2026-10-28)
5. [Metrics](#), vLLM documentation (accessed 2026-10-28)
6. [SGLang: Efficient Execution of Structured Language Model Programs](#), Zheng et al., arXiv, v2 Jun 6, 2024 (accessed 2026-10-28)
7. [SGLang HiCache Best Practices](#), SGLang documentation (accessed 2026-10-28)
8. [LMCache documentation](#), The LMCache Team (accessed 2026-10-28)
9. [ai-dynamo/dynamo README](#), NVIDIA, tag v1.6.1 (2026-10-27) (accessed 2026-10-28)
10. [KVBM configuration reference](#), NVIDIA Dynamo documentation v1.6.1 (accessed 2026-10-28)
11. [Qwen2.5-72B-Instruct config.json](#), Qwen team, Hugging Face (accessed 2026-10-28)
12. [Qwen2.5-72B-Instruct model card](#), Qwen team, Hugging Face (accessed 2026-10-28)
13. [NVIDIA H100 Tensor Core GPU](#), NVIDIA (accessed 2026-10-28)
14. [NVIDIA Grace Hopper Superchip Architecture In-Depth](#), NVIDIA Technical Blog, Nov 10, 2022 (accessed 2026-10-28)
15. [NVIDIA GB200 NVL72](#), NVIDIA (accessed 2026-10-28)
16. [Micron 9550 NVMe SSD](#), Micron (accessed 2026-10-28)
17. [Building py-kvcache: A Performance Characterization of External KV Caching for vLLM with NVMe SSDs](#), Kanichai, De Matteis, Trivedi, arXiv, Sep 10, 2026 (accessed 2026-10-28)
18. [NVIDIA GH200 Grace Hopper Superchip](#), NVIDIA (accessed 2026-10-28)
19. [NVIDIA GB200 NVL72 Delivers Trillion-Parameter LLM Training and Real-Time Inference](#), NVIDIA Technical Blog, Mar 18, 2024 (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/latest/features/automatic_prefix_caching
2. https://docs.vllm.ai/en/latest/design/prefix_caching
3. <https://arxiv.org/abs/2312.07104>
4. <https://huggingface.co/Qwen/Qwen2.5-72B-Instruct/blob/main/config.json>
5. <https://huggingface.co/Qwen/Qwen2.5-72B-Instruct>
6. <https://www.nvidia.com/en-us/data-center/h100>
7. <https://www.micron.com/products/storage/ssd/data-center-ssd/9550-ssd>
8. https://docs.vllm.ai/en/latest/features/kv_offloading_usage
9. https://docs.sglang.io/advanced_features/hicache_best_practices.html
10. <https://docs.lmcache.ai>
11. <https://github.com/ai-dynamo/dynamo>
12. <https://docs.nvidia.com/dynamo/reference/components/kvbm-configuration>
13. <https://vllm.ai/blog/2026-01-08-kv-offloading-connector>
14. <https://arxiv.org/abs/2609.11744>

15. <https://developer.nvidia.com/blog/nvidia-grace-hopper-superchip-architecture-in-depth>
16. <https://www.nvidia.com/en-us/data-center/grace-hopper-superchip>
17. <https://www.nvidia.com/en-us/data-center/gb200-nv172>
18. <https://developer.nvidia.com/blog/nvidia-gb200-nv172-delivers-trillion-parameter-llm-training-and-real-time-inference>
19. <https://docs.vllm.ai/en/latest/design/metrics>

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.