



MHA vs GQA vs MLA: How Attention Design Sets KV Cache Size

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 · kv cache size calculation · kv cache memory · kv cache calculator · mha vs gqa vs mla kv cache · deepseek mla kv cache size per token · llama 3 gqa kv cache bytes per token · grouped query attention memory savings · multi-head latent attention explained

Original article: <https://gpusmith.com/articles/en/gqa-mla-attention-kv-cache-size-by-architecture>



In this article

1. KV cache in one paragraph
 2. The MHA baseline formula
 3. MQA and GQA
 4. Worked example: Llama 3.1 8B
 5. Worked example: Llama 3.1 70B
 6. MLA: caching a latent
 7. Worked example: DeepSeek-V3
 8. Concurrency and context consequences
 9. Quantizing the KV cache
 10. Things the formula ignores
 11. FAQ
 12. Sources
-

KV cache in one paragraph

During autoregressive decoding, every attention layer needs the keys and values of all earlier tokens in the sequence. Serving engines keep them in GPU memory instead of recomputing them: that is the KV cache. It grows linearly with tokens per sequence and again with the number of resident sequences. Shazeer's multi-query attention paper framed the cost precisely: incremental decoding is often slow "due to the memory-bandwidth cost of repeatedly loading the large 'keys' and 'values' tensors" ([Shazeer 2019](#)). The same tensors fill HBM as context or batch grows, and the attention design, not the parameter count, sets how many bytes each cached token costs.

The MHA baseline formula

In standard multi-head attention (MHA), each layer has `n_heads` query heads and the same number of key and value heads, each of width `head_dim`. Per cached token, per layer, you store one key vector and one value vector per KV head. The general per-token formula is:

```
KV bytes per token = 2 x layers x kv_heads x head_dim x bytes_per_element
```

The factor 2 is keys plus values. DeepSeek-V2's comparison table gives the same expressions in elements: $2 \cdot n_h \cdot d_h \cdot l$ for MHA, $2 \cdot n_g \cdot d_h \cdot l$ for GQA with `n_g` groups, and $2 \cdot d_h \cdot l$ for MQA ([DeepSeek-V2, Table 1](#)). Multiply elements by 2 bytes for BF16/FP16 or 1 byte for FP8, then by tokens per sequence and by sequences.

Worked example, MHA: Llama 2 7B. The config has 32 layers, 32 attention heads, 32 KV heads and hidden size 4096, so `head_dim = 4096 / 32 = 128` (derived; the config has no `head_dim` field) ([Llama 2 7B config, unsloth mirror](#)).

- Elements per token: $2 \times 32 \times 32 \times 128 = 262,144$

- BF16: 524,288 bytes = **512 KiB per token**
- FP8: **256 KiB per token**

At the model's native 4,096-token context (`max_position_embeddings`: 4096 in the same config), one sequence holds 2 GiB of BF16 cache. At a hypothetical 128K (131,072-token) context, it would need 64 GiB per sequence.

MQA and GQA

Multi-query attention (MQA) keeps all query heads but shares a single key head and a single value head across them, "greatly reducing the size of these tensors and hence the memory bandwidth requirements of incremental decoding," with "only minor quality degradation" in Shazeer's experiments ([Shazeer 2019](#)). In KV-cache terms, `kv_heads` = 1. Pope et al. reported that, with appropriate partitioning, the lower memory requirement of multiquery attention "enables scaling up to 32x larger context lengths" on PaLM 540B ([Pope et al. 2022](#)).

Grouped-query attention (GQA) sits between the two. It uses "an intermediate (more than one, less than number of query heads) number of key-value heads," and the authors show that uptrained GQA "achieves quality close to multi-head attention with comparable speed to MQA" ([Ainslie et al. 2023](#)). The experiments uptrained T5 checkpoints with 5% of original pre-training compute, so the quality claim applies to that setting. The paper also makes a sharding point that matters for multi-GPU serving: standard sharding "replicates the single key and value head by the number of model partitions," and GQA removes that waste ([Ainslie et al. 2023](#)).

For GQA, `kv_heads` in the formula is the config's `num_key_value_heads`, not `num_attention_heads`. Mixing the two up is the most common sizing error: it overstates Llama 3.1 cache by 4x (8B) or 8x (70B).

Worked example: Llama 3.1 8B

From the config: `num_hidden_layers`: 32, `num_attention_heads`: 32, `num_key_value_heads`: 8, `head_dim`: 128, `max_position_embeddings`: 131072, `torch_dtype`: `bfloat16` ([Llama 3.1 8B config](#), [unsloth mirror](#)). Four query heads share each KV head. The meta-llama originals are gated; these counts match the Llama 3 paper's Table 3 ([arXiv 2407.21783](#)).

- Elements per token: $2 \times 32 \times 8 \times 128 = 65,536$
- BF16: 131,072 bytes = **128 KiB per token**
- FP8: **64 KiB per token**
- One full 131,072-token sequence: $128 \text{ KiB} \times 131,072 = \mathbf{16 \text{ GiB}}$ BF16, **8 GiB** FP8

Same layers and head width as Llama 2 7B, a quarter of the KV heads, a quarter of the cache (derived).

Worked example: Llama 3.1 70B

From the config: num_hidden_layers: 80, num_attention_heads: 64, num_key_value_heads: 8, hidden_size: 8192, head_dim: 128 ([Llama 3.1 70B config, unsloth mirror](#)). The mirror's config lists head_dim explicitly, and it matches $8192 / 64 = 128$. Eight query heads share each KV head.

- Elements per token: $2 \times 80 \times 8 \times 128 = 163,840$
- BF16: 327,680 bytes = **320 KiB per token**
- FP8: **160 KiB per token**
- One 131,072-token sequence: **40 GiB** BF16, **20 GiB** FP8

For contrast (derived counterfactuals, not real checkpoints): with 64 KV heads (MHA) the same model would need 2,560 KiB per token and 320 GiB per 128K sequence; with 1 KV head (MQA), 40 KiB and 5 GiB.

MLA: caching a latent

Multi-head latent attention (MLA), introduced in DeepSeek-V2, does not reduce the number of heads. It compresses keys and values jointly into a low-rank latent vector and caches that latent instead of per-head K and V. MLA "guarantees efficient inference through significantly compressing the Key-Value (KV) cache into a latent vector" ([DeepSeek-V2](#)). Per-head keys and values are reconstructed from the latent, or the up-projections are absorbed into the query and output projections.

One complication is rotary position embeddings, which cannot be folded through the compression. DeepSeek therefore adds a small "decoupled" RoPE key shared across heads, which also has to be cached. The paper states the total: "DeepSeek-V2 requires a total KV cache containing $(d_c + d_h^R) \cdot l$ elements" per token, where d_c is the latent width and d_h^R the decoupled RoPE key width ([DeepSeek-V2, section 2.1](#)). There is no factor 2: one latent serves both K and V. With $d_c = 4 \cdot d_h$ and $d_h^R = d_h / 2$, the paper notes its cache "is equal to GQA with only 2.25 groups" ([DeepSeek-V2](#)).

In config.json terms:

```
MLA KV bytes per token = layers x (kv_lora_rank + qk_rope_head_dim) x bytes_per_element
```

Worked example: DeepSeek-V3

From the config: num_hidden_layers: 61, num_attention_heads: 128, num_key_value_heads: 128, kv_lora_rank: 512, qk_rope_head_dim: 64, qk_nope_head_dim: 128, v_head_dim: 128, max_position_embeddings: 163840 ([DeepSeek-V3 config](#)). Note that num_key_value_heads: 128 is misleading for an MLA model. Plugging it into the GQA formula gives the uncompressed cache, not the latent cache. The V3 technical report states that V3 adopts MLA, "thoroughly validated in DeepSeek-V2" ([DeepSeek-V3 Technical Report](#)).

- Elements per token: $61 \times (512 + 64) = 35,136$
- BF16: 70,272 bytes = **68.6 KiB per token**
- FP8 (1 byte per element, scales ignored): **34.3 KiB per token**

- One 131,072-token sequence: **8.58 GiB** BF16, **4.29 GiB** FP8

The MHA comparison uses the paper's formula $2 \cdot n_h \cdot d_h \cdot l$ with 128 heads and $d_h = 128$: $2 \times 128 \times 128 \times 61 = 1,998,848$ elements, or 3,904 KiB per token in BF16 and 488 GiB per 128K sequence. MLA stores about 1.8% of that (derived). Per layer, DeepSeek-V3 caches 576 elements per token; Llama 3.1 70B caches $2 \times 8 \times 128 = 2,048$. So the 671B-parameter MoE stores less KV per token (68.6 KiB) than Llama 3.1 8B (128 KiB), even with nearly twice the layers.

Reconciling the 93.3% headline

The DeepSeek-V2 abstract says that, compared with DeepSeek 67B, V2 "reduces the KV cache by 93.3%" ([DeepSeek-V2](#)). Our arithmetic from the configs:

- DeepSeek 67B (GQA): 95 layers, 8 KV heads, $\text{head_dim } 8192 / 64 = 128$, so $2 \times 95 \times 8 \times 128 = 194,560$ elements per token ([DeepSeek LLM 67B config](#))
- DeepSeek-V2 (MLA): 60 layers $\times$ 576 = 34,560 elements per token ([DeepSeek-V2 config](#)). This matches the 34.6K elements the paper's Table 9 lists for its large MLA model.

At equal precision that is an 82.2% reduction. The paper also says the deployed V2 used KV cache quantization "to further compress each element in its KV cache into 6 bits on average" ([DeepSeek-V2, section 3.2.3](#)). With 6-bit V2 elements against a BF16 67B cache, the reduction becomes $1 - 25,920 / 389,120 = 93.3\%$. The same section says that "benefiting from MLA and these optimizations, actually deployed DeepSeek-V2 requires significantly less KV cache than DeepSeek 67B", which supports reading the headline as architecture plus quantization. The paper does not state the 67B cache precision, so the BF16 baseline and the split are our reconciliation. For capacity planning, use roughly 82% for the architectural share alone.

Concurrency and context consequences

The per-token number becomes a capacity number when you fix a KV budget. The table below assumes 100 GiB of HBM left for cache after weights and activations. That is an illustrative figure, not a property of any specific GPU or engine. Each sequence is a full 131,072 tokens.

Model (attention type)	KV per token, BF16	128K sequence, BF16	128K sequences per 100 GiB, BF16	128K sequences per 100 GiB, FP8
Llama 2 7B layout (MHA, hypothetical at 128K)	512 KiB	64 GiB	1	3
Llama 3.1 8B (GQA, 8 KV heads)	128 KiB	16 GiB	6	12
Llama 3.1 70B (GQA, 8 KV heads)	320 KiB	40 GiB	2	5
DeepSeek-V3 (MLA, 512 + 64)	68.6 KiB	8.58 GiB	11	23
Gemma 3 27B (GQA + 5:1 sliding window)	see below	10.4 GiB	9	n/a

Model (attention type)	KV per token, BF16	128K sequence, BF16	128K sequences per 100 GiB, BF16	128K sequences per 100 GiB, FP8
gpt-oss-120b (GQA + alternating 128-token window)	see below	4.5 GiB	22	n/a

All values derived from config.json fields using the formulas above; whole sequences rounded down.

Batch scaling is linear. Holding 32 concurrent 128K sequences needs $32 \times 40 \text{ GiB} = 1,280 \text{ GiB}$ of BF16 cache for Llama 3.1 70B: about 640 GiB at FP8, versus $32 \times 8.58 = 274.5 \text{ GiB}$ for DeepSeek-V3 at BF16. A more typical chat mix of 64 sequences at 8,192 tokens needs 160 GiB (Llama 3.1 70B, BF16), 64 GiB (Llama 3.1 8B) or 34.3 GiB (DeepSeek-V3). The [KV cache memory calculator](#) covers sizing from prompt/output distributions rather than worst-case length.

Tensor parallelism changes the per-GPU picture

GQA heads shard across tensor-parallel ranks: at TP=8, each Llama 3.1 70B rank holds one KV head, or 40 KiB per token (derived). MLA's latent is a single shared vector, so plain TP does not split it. SGLang's team wrote that DeepSeek models "use MLA and only have one KV head," so tensor parallelism on 8 GPUs "will lead to duplicated KV cache" ([LMSYS](#), [SGLang v0.4](#)). Their fix is data-parallel attention. SGLang's DeepSeek page describes it as reducing "KV-cache duplication across TP ranks," recommended for large-batch throughput and not for low-latency small batches ([SGLang DeepSeek V3 usage](#)). Under plain TP=8, each DeepSeek-V3 rank carries the full 68.6 KiB per token, more than a TP=8 Llama 3.1 70B rank (derived).

Quantizing the KV cache

FP8 halves every BF16 figure above. vLLM exposes this as `kv_cache_dtype`, with `fp8_e4m3` (CUDA 11.8+ and ROCm) and `fp8_e5m2` (CUDA 11.8+). Scales default to 1.0 unless calibrated, and per-tensor scales apply separately to Q, K and V ([vLLM quantized KV cache docs](#), page dated August 2026). That page gives no accuracy numbers. Validate FP8 KV on your own evaluation set, especially for long-context retrieval.

Engines do not always store "1 byte per element" exactly. vLLM's FlashMLA sparse backend (v0.11.0 docs; unchanged in the v0.32.0 source) defines an `fp8_ds_m1a` format of 656 bytes per token per layer: 512 FP8 latent values, 16 bytes of float32 scales, and 64 RoPE values kept in BF16 because "this part is not quantized for accuracy" ([vLLM FlashMLA sparse backend](#)). Across 61 layers that is 40,016 bytes (39.1 KiB) per token, about 14% more than the naive 34.3 KiB (derived). For the per-format accuracy and cost trade-offs, see [FP16 vs FP8 vs INT4 inference costs](#).

Serving engines also differ in how they lay out MLA's latent. vLLM's MLA backend writes "the latent and rope to kv cache" as one 576-wide entry with a single KV head per layer. It uses an MHA-style compute path for prefill and an MQA-style, "data-movement friendly" path for decode ([vLLM MLA common backend](#), v0.9.2). A fallback that materializes per-head K and V would store far more.

Things the formula ignores

Sliding-window and hybrid attention. Some models apply full attention only on a subset of layers. Gemma 3 uses "5 local layers for every global layer," with local layers limited to a 1,024-token span ([Gemma 3 Technical Report](#)). For Gemma 3 27B, vLLM's design doc counts 52 sliding-window and 10 full-attention layers ([vLLM hybrid KV cache manager](#)). With 16 KV heads and head_dim 128 ([Gemma 3 27B config](#), [unsloth mirror](#)), the derived BF16 cache for one 128K sequence is 10 GiB for the global layers plus a 416 MiB cap for the local layers: about 10.4 GiB, versus 62 GiB if all 62 layers were global. gpt-oss-120b alternates "banded window and fully dense patterns" with a 128-token band and 8 KV heads of dimension 64 ([gpt-oss model card](#)). Its config lists 36 layers, half sliding_attention ([gpt-oss-120b config](#)). The 18 full layers cost 36 KiB per token in BF16, or 4.5 GiB at 128K (derived). These savings only materialize if the engine frees out-of-window blocks. vLLM does: it allocates per-token blocks and frees "blocks that are outside the sliding window." Its doc also notes that the layer-grouping scheme wastes some memory on padding for Gemma 3 ([vLLM hybrid KV cache manager](#)).

Paging and prefix sharing. PagedAttention stores KV in fixed-size blocks with "near-zero waste in KV cache memory" and shares blocks "within and across requests" ([Kwon et al. 2023](#)). A partly filled last block adds a little per sequence; shared system prompts can push physical memory below the per-sequence sum.

Engine pools and reservations. Weights, activations and allocator headroom come first. Reconcile your arithmetic with the engine's startup report; the [vLLM KV cache capacity planner](#) walks through that check.

FAQ

Which config.json field gives the KV head count? num_key_value_heads. If it is absent, the model is MHA and KV heads equal num_attention_heads. If head_dim is absent, use hidden_size / num_attention_heads. For MLA models, ignore both and use kv_lora_rank + qk_rope_head_dim.

Is MLA always smaller than GQA? Per layer, MLA in DeepSeek-V3 stores 576 elements. A GQA layer stores $2 \times \text{kv_heads} \times \text{head_dim}$, so GQA with 2 KV heads of width 128 (512 elements) would be smaller. The paper's own framing is "GQA with only 2.25 groups" ([DeepSeek-V2](#)). Its claim is better quality than MHA at that size, not the smallest possible cache.

Does KV cache size depend on total parameters? No. A dense 70B model and a 671B MoE can be compared only through layers, KV heads (or latent width), head width and cache dtype. For DeepSeek-family deployment costs beyond the cache, see [DeepSeek R1 self-hosting cost](#).

If you need these numbers turned into a GPU count for a specific workload, GPU Smith does that sizing work ([contact](#)).

Sources

- [Fast Transformer Decoding: One Write-Head is All You Need](#) — Noam Shazeer, arXiv, 2019-11-06, accessed 2026-10-28

- [GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints](#) — Ainslie et al., arXiv, 2023-05-22, accessed 2026-10-28
- [Efficiently Scaling Transformer Inference](#) — Pope et al., arXiv, 2022-11-09, accessed 2026-10-28
- [DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model](#) — DeepSeek-AI, arXiv, 2024-05-07, accessed 2026-10-28
- [DeepSeek-V3 Technical Report](#) — DeepSeek-AI, arXiv, 2024-12-27, accessed 2026-10-28
- [Efficient Memory Management for Large Language Model Serving with PagedAttention](#) — Kwon et al., arXiv, 2023-09-12, accessed 2026-10-28
- [Gemma 3 Technical Report](#) — Gemma Team, Google, arXiv, 2025-03-25, accessed 2026-10-28
- [gpt-oss-120b & gpt-oss-20b Model Card](#) — OpenAI, arXiv, 2025-08-08, accessed 2026-10-28
- [Llama 2 7B config.json \(unsloth mirror\)](#) — Hugging Face, accessed 2026-10-28
- [Llama 3.1 8B config.json \(unsloth mirror\)](#) — Hugging Face, accessed 2026-10-28
- [Llama 3.1 70B config.json \(unsloth mirror\)](#) — Hugging Face, accessed 2026-10-28
- [DeepSeek-V3 config.json](#) — DeepSeek-AI, Hugging Face, accessed 2026-10-28
- [DeepSeek-V2 config.json](#) — DeepSeek-AI, Hugging Face, accessed 2026-10-28
- [DeepSeek LLM 67B Base config.json](#) — DeepSeek-AI, Hugging Face, accessed 2026-10-28
- [Gemma 3 27B IT config.json \(unsloth mirror\)](#) — Hugging Face, accessed 2026-10-28
- [gpt-oss-120b config.json](#) — OpenAI, Hugging Face, accessed 2026-10-28
- [Hybrid KV Cache Manager \(design doc\)](#) — vLLM, dated 2026-08-19, accessed 2026-10-28
- [Quantized KV Cache](#) — vLLM, dated 2026-08-03, accessed 2026-10-28
- [MLA common backend \(v0.9.2 API docs\)](#) — vLLM, accessed 2026-10-28
- [FlashMLA sparse backend \(v0.11.0 API docs\)](#) — vLLM, accessed 2026-10-28
- [SGLang v0.4: Zero-Overhead Batch Scheduler, Cache-Aware Load Balancer, Faster Structured Outputs](#) — LMSYS, 2024-12-04, accessed 2026-10-28
- [DeepSeek V3/V3.1/R1 Usage](#) — SGLang, accessed 2026-10-28
- [The Llama 3 Herd of Models](#) — Llama Team, Meta, arXiv, 2024-07-31, Table 3, 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/1911.02150>
2. <https://arxiv.org/abs/2405.04434>
3. <https://huggingface.co/unsloth/llama-2-7b/raw/main/config.json>
4. <https://arxiv.org/abs/2211.05102>
5. <https://arxiv.org/abs/2305.13245>
6. <https://huggingface.co/unsloth/Llama-3.1-8B/raw/main/config.json>
7. <https://arxiv.org/abs/2407.21783>
8. <https://huggingface.co/unsloth/Llama-3.1-70B/raw/main/config.json>
9. <https://huggingface.co/deepseek-ai/DeepSeek-V3/raw/main/config.json>
10. <https://arxiv.org/abs/2412.19437>

11. <https://huggingface.co/deepseek-ai/deepseek-llm-67b-base/raw/main/config.json>
12. <https://huggingface.co/deepseek-ai/DeepSeek-V2/raw/main/config.json>
13. <https://lmsys.org/blog/2024-12-04-sglang-v0-4>
14. https://docs.sglang.io/basic_usage/deepseek_v3.html
15. https://docs.vllm.ai/en/stable/features/quantization/quantized_kvcache
16. https://docs.vllm.ai/en/v0.11.0/api/vllm/v1/attention/backends/mla/flashmla_sparse.html
17. <https://docs.vllm.ai/en/v0.9.2/api/vllm/attention/backends/mla/common.html>
18. <https://arxiv.org/abs/2503.19786>
19. https://docs.vllm.ai/en/stable/design/hybrid_kv_cache_manager
20. <https://huggingface.co/unsloth/gemma-3-27b-it/raw/main/config.json>
21. <https://arxiv.org/abs/2508.10925>
22. <https://huggingface.co/openai/gpt-oss-120b/raw/main/config.json>
23. <https://arxiv.org/abs/2309.06180>

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.