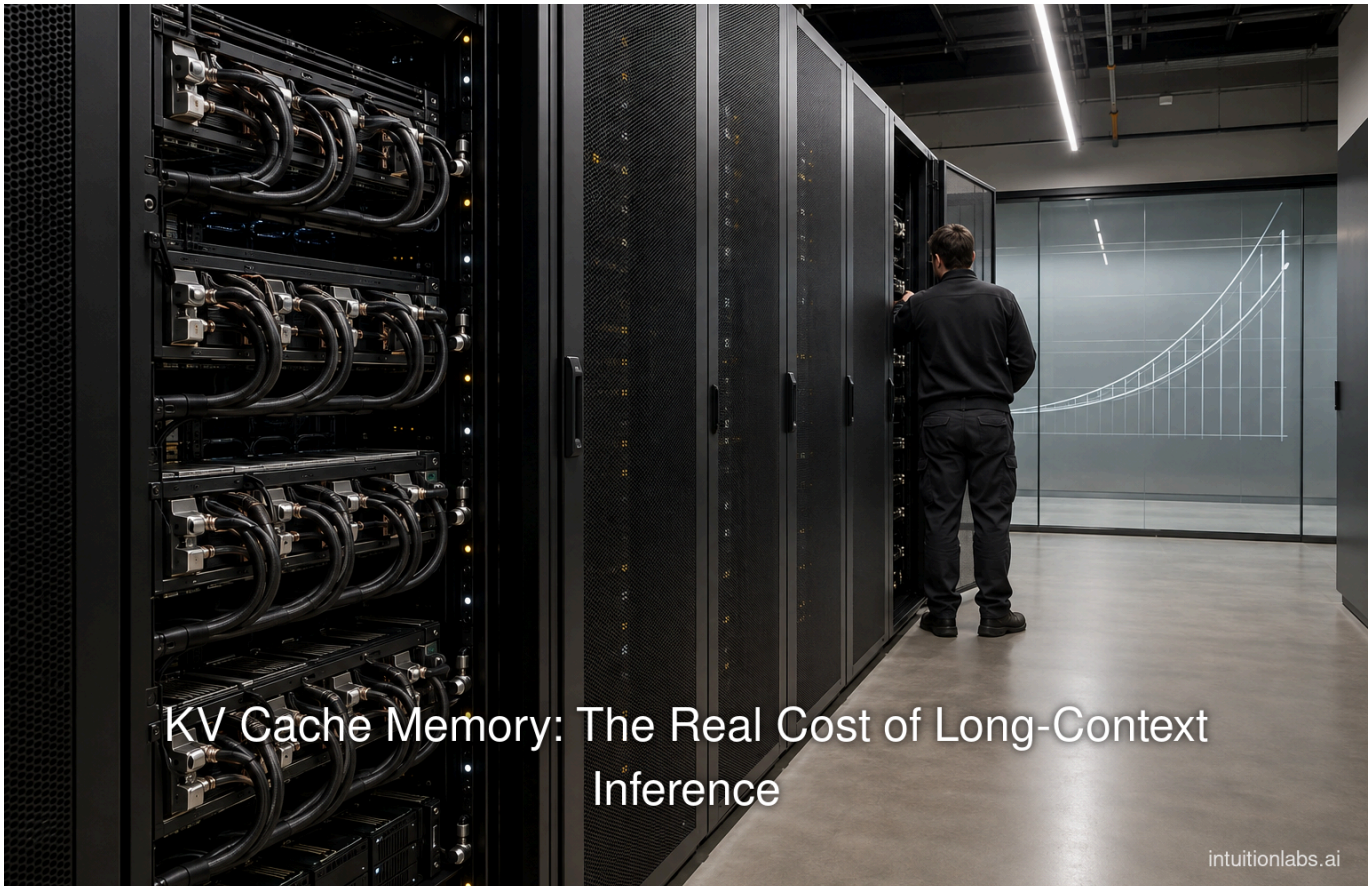
KV Cache Memory: The Real Cost of Long-Context Inference

IntuitionLabs.ai · Adrien Laurent · 2026-09-05 · kv cache · long context inference · llm inference cost · gpu memory · grouped query attention · kv cache quantization · transformer memory · inference optimization · multi head latent attention



Executive Summary

Long-context inference has turned the **key-value (KV) cache**, the mechanism transformer models use to avoid recomputing attention over every prior token at each decoding step (huggingface.co), into one of the largest and fastest-growing line items in GPU memory and cost budgets. As of September 2026, this report finds that KV cache memory follows a well-documented formula: twice the product of layer count, KV head count, head dimension, sequence length, batch size, and bytes-per-parameter. Applied to real, published model architectures, that formula shows a **Llama 3 70B** model needs approximately 40 GB of KV cache for a single 128,000-token request, a figure Hugging Face's own published table lists precisely as 39.06 GB at that context length (huggingface.co), on top of roughly 140 GB just to hold the model's weights in FP16.

Because KV cache memory scales linearly with batch size and context length, serving multiple long-context users concurrently multiplies this cost directly, which is why current data-center GPUs are built with ever-larger memory pools: 80 GB for the NVIDIA H100 (nvidia.com), 141 GB for the H200 (nvidia.com), and 372 GB per Grace

Blackwell Superchip in the GB200 NVL72 rack (nvidia.com). On-demand cloud pricing for these accelerators ranged from about $3.99 per GPU-hour for an H100 to $5.99 for an H200 on the providers checked for this article, as of September 2026 (lambda.ai) (together.ai), and AWS cut its own On-Demand GPU instance pricing by up to 45% in a June 2025 change (aws.amazon.com), underscoring how quickly these cost baselines move.

The industry has responded with a stack of memory-reduction techniques at every layer: architectural changes such as grouped-query attention and DeepSeek's multi-head latent attention, which its own paper reports cuts KV cache by over 90% versus a comparable dense architecture (detailed below); precision changes such as FP8 KV cache quantization, which vendor documentation reports enables 2 to 3x larger batch sizes on H100 GPUs; and memory-management changes such as PagedAttention, which cut memory waste from 60 to 80% down to under 4% in the original vLLM paper's own benchmark (vllm.ai). A caution runs through the evidence: the independent RULER benchmark found only about half of 17 evaluated long-context models perform reliably even at 32,000 tokens despite advertised windows up to 1 million (arxiv.org), a gap Anthropic's own documentation names "context rot" (platform.claude.com). The consistent conclusion is that KV cache memory is a controllable but easily underestimated cost, driven as much by model architecture choice as by hardware and software optimization.

Introduction and Background

Every large language model (LLM) that generates text one token at a time relies on a mechanism called the **key-value cache**, or **KV cache**, to avoid recomputing the same attention math over and over. As of September 2026, the industry's push toward **long-context inference** (prompts and conversations spanning tens of thousands to over a million tokens) has turned this once-obscure implementation detail into one of the largest line items in GPU memory budgets and inference cost. Hugging Face's transformer documentation frames the underlying problem directly: to predict a token deep into a sequence, the model needs information from every token that came before it, and recomputing that information at every step is wasteful (huggingface.co). The KV cache exists to store the intermediate attention values so they can be reused rather than recalculated (huggingface.co).

The practical consequence is that KV cache memory now competes directly with model weights for scarce GPU high-bandwidth memory (HBM), and it does so in a way that scales linearly with context length, batch size, and the number of concurrent users being served, as NVIDIA's own inference-optimization engineering guidance states (developer.nvidia.com). A single long-context request from one user can consume tens of gigabytes of memory on top of the model's weights, and NVIDIA has published a concrete figure: a 128,000-token context window for **Llama 3 70B** consumes about 40 GB of memory per user, scaling linearly as more users are served concurrently (developer.nvidia.com).

This explainer answers **kv cache memory cost long context inference** from first principles: what the KV cache is, the exact formula used to calculate its memory footprint, worked examples against real published model architectures (Llama 3, Mistral, Qwen2, DeepSeek-V2), the GPU hardware and cloud pricing that determine what long-context serving actually costs, and the architectural and software techniques (grouped-query attention, multi-head latent attention, quantization, paged memory management, sliding windows, and cache eviction) that the industry uses to bring that cost down. Every quantitative claim below is traced to the primary paper, vendor specification page, or official documentation that produced it, with an observation date, because prices, model specifications, and benchmark results change quickly.

What Is the KV Cache and Why Long Context Makes It Expensive

Transformer-based LLMs generate text **autoregressively**: each new token is predicted one at a time, conditioned on every token that precedes it. During this process, the **self-attention** mechanism computes a **query (Q)**, **key (K)**, and **value (V)** vector for every token in the sequence. Without caching, generating the next token would require recomputing the K and V vectors for the entire preceding sequence at every single decoding step, an approach whose cost grows quadratically with sequence length. The KV cache eliminates this redundant work by storing the K and V vectors for previously processed tokens the first time they are computed, so that only the new token's K and V need to be calculated at each subsequent step (huggingface.co).

The tradeoff is that this cache is not free: it lives in the same GPU HBM pool as the model's weights, and it grows for the entire duration of a request. NVIDIA's technical guidance identifies the two dominant consumers of inference-time GPU memory as the model's weights and the KV cache, noting explicitly that KV cache memory requirements grow linearly with both batch size and sequence length and can quickly scale beyond what a single accelerator provides, as described above. NVIDIA's own TensorRT-LLM documentation similarly names weights, activation tensors, and the KV cache as the three major GPU-memory contributors during inference, and by default allocates fully 90% of whatever GPU memory remains after loading weights and activations to the KV cache (nvidia.github.io) (nvidia.github.io).

Attention variants determine how large the cache is per token. The original **multi-head attention (MHA)** design gives every attention head its own K and V projections, which is expensive to cache. Two families of architectural changes reduce this:

- **Multi-query attention (MQA)**, proposed by Noam Shazeer in 2019, shares a single set of keys and values across all attention heads, which Shazeer's paper states "greatly reduc[es] the size of these tensors" that must be cached and read back at each decoding step (arxiv.org).
- **Grouped-query attention (GQA)**, introduced by Ainslie et al., sits between MHA and MQA: it uses more than one but fewer than the full number of query heads' worth of key-value heads, and the paper reports that models "uptrained" into GQA reach accuracy close to full MHA while decoding at speeds comparable to MQA (arxiv.org).

Meta adopted GQA for exactly this memory reason: Meta's Llama 2 paper states directly that "the memory costs associated with the KV cache size in multi-head attention (MHA) models grow significantly" as context and batch size increase, which is why the 34B and 70B Llama 2 variants use a GQA variant with 8 KV projections rather than one projection per attention head (arxiv.org). Every subsequent Llama 3 model, from 8B to 70B, also uses GQA (github.com).

Calculating KV Cache Memory: The Formula and Worked Examples

The standard formula for KV cache memory, as published in NVIDIA's LLM inference-optimization technical blog, has two equivalent forms. Per token, per request:

> Size of KV cache per token in bytes = 2 x (num_layers) x (num_heads x dim_head) x precision_in_bytes (developer.nvidia.com)

And for a full batch across a full sequence:

> Total size of KV cache in bytes = (batch_size) x (sequence_length) x 2 x (num_layers) x (hidden_size) x sizeof(FP16) (developer.nvidia.com)

The **factor of 2** accounts for storing both the key and the value tensor; `num_layers` is the transformer's depth; `hidden_size` (or `num_heads x dim_head`) is the model's hidden dimension; and `precision_in_bytes` is 2 for FP16/BF16, 1 for FP8/INT8, or 0.5 for 4-bit formats. NVIDIA's own worked example applies this formula to a **Llama 2 7B** model at batch size 1 and a 4,096-token sequence in FP16: 1 x 4096 x 2 x 32 x 4096 x 2 bytes, which NVIDIA states comes to approximately 2 GB of KV cache for that single request (developer.nvidia.com).

Applying the identical formula to models with **grouped-query attention** requires substituting the number of KV heads (not the number of query heads) for `num_heads`, since only the KV projections are cached. *Table 1* below applies this method to several openly documented model architectures, using each model's official configuration values, to show how architecture alone changes the memory bill for a 128,000-token context at FP16 precision and batch size 1.

Table 1 below compares architecture parameters and KV cache size at a 128K-token context length across five publicly documented model families. Layer counts, head counts, and KV head counts are taken from each model's official Hugging Face configuration file or technical report; the "Calculated KV cache" column applies the NVIDIA formula above rather than an independently measured figure, so it is an **estimate**, not a benchmark measurement, unless otherwise noted.

Model	Layers	Attention heads	KV heads	Head dim	KV-cache estimate or architecture note
Llama 3 8B	32	32	8 (huggingface.co)	128	~16 GiB (calculated); Meta/Hugging Face's own published table lists 15.62 GB at 128K tokens (huggingface.co)
Llama 3 70B	80	64	8 (huggingface.co)	128	~40 GiB (calculated); matches NVIDIA's own published figure of "about 40 GB" for Llama 3 70B at 128K context (developer.nvidia.com)
Mistral 7B	32	32	8 (arxiv.org)	128	Uses a rolling/sliding window instead of full-length caching, which Mistral's own paper reports cuts cache memory by 8x at a 32K window (detailed in *Table 3*)
Qwen2-7B	28	28	4 (arxiv.org)	128	Half the KV heads of Llama 3 8B at a similar layer count, roughly halving the per-token cache cost for a comparable hidden size
DeepSeek-V2 (MLA)	not applicable (latent compression)	not applicable	not applicable	not applicable	DeepSeek reports a 93.3% KV cache reduction versus its own prior-generation dense model, describing the compressed cache as equivalent to GQA with only 2.25 groups (arxiv.org)

The pattern in *Table 1* is that KV cache size is driven almost entirely by **KV head count** and **layer depth**, not by total parameter count. Qwen2-7B, at roughly the same parameter class as Llama 3 8B, uses only 4 KV heads against Llama 3's 8, which by the formula above roughly halves its per-token cache cost. Independent confirmation of the Llama 3 numbers comes from two different primary sources computed two different ways: Hugging Face and

Meta's joint Llama 3.1 announcement publishes a table showing FP16 KV cache size growing from 0.125 GB at 1,000 tokens to 15.62 GB at 128,000 tokens for the 8B model, and from 0.313 GB to 39.06 GB for the 70B model over the same range (huggingface.co), while NVIDIA's independently published engineering blog states the 70B model's 128K-context cache "consumes about 40 GB of memory," a figure consistent with Hugging Face's 39.06 GB within normal rounding. Loading the 70B model's weights alone in FP16 requires approximately 140 GB before any KV cache is added, according to the same NVIDIA blog post, which is already more than a single 80 GB H100 or even a single 141 GB H200 can hold (developer.nvidia.com).

Concurrency compounds the problem. Because KV cache scales linearly with batch size, running the same 70B model for 10 concurrent long-context users at 128K tokens each does not add 40 GB once, it adds roughly 400 GB, a multiple of what any single accelerator provides. This is why production serving stacks manage KV cache allocation as a first-class scheduling resource rather than an afterthought, a point developed further in the sections below.

GPU Memory Budgets and the Cost of Long-Context Inference

Because model weights and GPU-resident KV cache share the same GPU memory pool, an accelerator's HBM capacity limits the context and concurrency that can be served with the cache resident on that GPU. CPU-memory KV-cache offload or unified-memory systems can extend available capacity, although cache reuse must justify transfer overhead (developer.nvidia.com) (developer.nvidia.com). *Table 2* below summarizes memory, bandwidth, and dated provider-published price examples for accelerators used for long-context LLM inference.

Table 2 pairs GPU specifications from vendor datasheets with dated, provider-published on-demand price examples, so that the memory ceiling and illustrative rental cost can be read side by side.

GPU	HBM capacity	Memory bandwidth	On-demand price example (per GPU/hour; provider-stated date)
NVIDIA A100 (80GB)	80 GB (nvidia.com)	over 2 TB/s (nvidia.com)	Not directly quoted by the providers checked for this article; generally priced below H100
NVIDIA H100 SXM	80 GB (nvidia.com)	3.35 TB/s (nvidia.com)	Together AI HGX H100: $3.99 per GPU/hour, on-demand; price page states July 2026 (accessed September 5, 2026) (together.ai)
NVIDIA H200	141 GB (nvidia.com)	4.8 TB/s (nvidia.com)	Together AI HGX H200: $5.99 per GPU/hour, on-demand; price page states July 2026 (accessed September 5, 2026) (together.ai)
NVIDIA GB200 NVL72 (per Grace Blackwell Superchip)	372 GB per superchip; 13.4 TB across the full 72-GPU rack (nvidia.com) (nvidia.com)	up to 576 TB/s aggregate across the rack	Not separately listed by the providers checked for this article; the standalone Blackwell-generation **B200** GPU (a component of the NVL72 rack) rents for $6.69 per GPU/hour on Lambda (lambda.ai)

The practical reading of *Table 2* is that a single **H100** or **A100 80GB** GPU cannot hold both the weights and a full 128K-context KV cache for a 70B-class model at FP16, since the weights alone need roughly 140 GB (developer.nvidia.com); such a deployment requires multi-GPU tensor parallelism, which is precisely why AWS's own P5 instance family bundles 8 H100 GPUs for a combined 640 GB of HBM3 memory per instance

(aws.amazon.com), and why its newer P5e and P5en instances bundle 8 H200 GPUs for up to 1,128 GB per instance (aws.amazon.com). Splitting a model's weights across GPUs with tensor parallelism has a useful side effect for KV cache budgeting: vLLM's own optimization documentation notes that sharding weights across more GPUs leaves more of each GPU's memory available for the KV cache (docs.vllm.ai).

Price movements matter as much as specifications for cost estimation. AWS announced a pricing and usage-model change for its GPU-backed EC2 instances effective June 2025 that included **up to a 45% reduction** in On-Demand rates for GPU instance families, a substantial shift for anyone budgeting long-context inference costs against list prices from even a year earlier (aws.amazon.com). Because GPU cloud pricing changes quickly, any cost model for long-context serving should be re-validated against the provider's current page rather than carried forward from an older estimate; *Table 2* reports each provider's stated price date and the article's observation date.

When a request's context and batch size would require more KV cache than physically fits, operators have three practical levers, all drawn from official inference-server documentation rather than from vendor marketing: reduce the maximum number of concurrent sequences or batched tokens the scheduler will admit at once (docs.vllm.ai), let the KV cache manager claim a smaller, explicitly configured fraction of free GPU memory rather than TensorRT-LLM's default of 90% (nvidia.github.io), or apply one of the memory-reduction techniques covered in the next section. Each lever trades either concurrency, context length, or engineering complexity for memory headroom, and the correct choice depends on whether the deployment is latency-sensitive, throughput-sensitive, or context-length-sensitive.

Architectural and Software Techniques to Reduce KV Cache Memory

Because KV cache memory is the binding constraint on long-context serving, the field has produced a wide range of techniques to shrink it, at the architecture level, the numerical-precision level, and the memory-management level. *Table 3* below summarizes the main approaches, their mechanisms, and the reported impacts attributed to their cited sources.

Table 3 below groups these techniques into architectural changes made at training time, precision changes applied at serving time, and memory-management changes made by the inference server, since each category is adopted independently and they compose with one another.

Technique	Mechanism	Reported impact (claim source)
Grouped-query attention (GQA)	Shares KV projections across groups of query heads instead of one set per head	Reaches accuracy close to full multi-head attention at decoding speed comparable to multi-query attention, per the original paper (arxiv.org)
Multi-query attention (MQA)	Shares a single KV projection across all heads	"Greatly reduc[es]" cached tensor size, per Shazeer's original 2019 paper (see above); more aggressive than GQA but with a larger quality tradeoff
Multi-head latent attention (MLA)	Compresses K/V into a low-rank latent vector before caching (used in DeepSeek-V2/V3)	DeepSeek's own paper reports a 93.3% KV cache reduction and 5.76x higher maximum generation throughput versus its prior dense architecture (arxiv.org)

Technique	Mechanism	Reported impact (claim source)
Cross-layer attention (CLA)	Shares KV projections across adjacent transformer layers, not just across heads	The paper introducing CLA reports a further 2x cache reduction on top of MQA at nearly unchanged accuracy (arxiv.org)
Sliding-window / local attention	Bounds the cache to a fixed recent window instead of the full sequence (Mistral 7B, Gemma 2)	Mistral's paper reports an 8x cache-memory reduction at a 32K sequence length with a rolling buffer cache (arxiv.org); Gemma 2 interleaves local and global attention layers (arxiv.org)
Attention sinks / StreamingLLM	Retains a small number of initial "sink" tokens plus a recent window, discarding the rest	The StreamingLLM paper reports stable language modeling up to 4 million tokens using a bounded cache, without fine-tuning (arxiv.org)
KV cache eviction (H2O)	Identifies and retains "Heavy Hitter" tokens that dominate attention scores, evicting the rest	H2O's paper reports up to 29x higher throughput than DeepSpeed Zero-Inference and Hugging Face Accelerate at a 20% cache retention ratio (arxiv.org)
FP8 / INT8 KV cache quantization	Stores cached K/V tensors in 8-bit rather than 16-bit precision	NVIDIA's TensorRT-LLM documentation reports FP8 KV cache enables a 2 to 3x larger batch size on H100 for GPT-J-class models, for roughly a 1.5x performance benefit, and recommends FP8 over INT8 for lower accuracy impact (nvidia.github.io) (nvidia.github.io)
Paged KV cache management (PagedAttention / vLLM)	Allocates KV cache in fixed-size, non-contiguous blocks, like OS virtual memory paging	The vLLM project reports the technique reduces memory waste to under 4%, versus 60 to 80% wasted to fragmentation in prior systems, while improving throughput 2 to 4x at equal latency (vllm.ai) (arxiv.org)

Two of these techniques deserve additional context because they represent genuinely different design philosophies. **PagedAttention**, introduced in the paper behind the vLLM inference engine, borrows directly from operating-system memory management: rather than pre-allocating one large contiguous memory block per request (which forces over-reservation for a sequence that might grow to its maximum length but often does not), it allocates cache in small fixed-size blocks that can be assigned non-contiguously, the same way an OS pages virtual memory (arxiv.org). The paper's own instrumentation of an unmodified serving system found that **65% of GPU memory was allocated to model weights** and roughly 30% to the dynamic KV cache when serving a 13B parameter model on a 40 GB A100, with much of that KV cache allocation going to waste under naive contiguous allocation (arxiv.org).

Quantization, by contrast, does not change how memory is allocated; it changes how many bytes each cached value consumes. Because the KV cache formula above scales linearly with `precision_in_bytes`, moving from FP16 (2 bytes) to FP8 or INT8 (1 byte) mechanically halves KV cache memory for a fixed context length and batch size, which is consistent with NVIDIA's reported 2 to 3x batch-size increase (batch size headroom is not purely linear with cache savings, since freed memory also has to accommodate activation growth) (nvidia.github.io). vLLM's own documentation confirms FP8 KV cache quantization is supported in both a simpler per-tensor scheme and a more accurate per-attention-head scheme that requires the FlashAttention backend and calibration through the `llm-compressor` toolkit (github.com) (github.com).

The GQA, MQA, MLA, CLA, sliding-window, StreamingLLM, and H2O figures in this table come from the papers that introduced each technique and should be read as source-attributed reported impacts.

Data Analysis and Evidence

This section assembles the quantitative record on how long context actually behaves in production-representative benchmarks, independent of any single vendor's claims.

Memory waste under naive allocation is large and well documented. Before PagedAttention, the vLLM team's own measurement of existing LLM-serving systems found they wasted **60% to 80% of allocated GPU memory** to fragmentation and over-reservation of KV cache space, because systems pre-allocated a contiguous block sized for a request's maximum possible length regardless of how long the request actually turned out to be (vllm.ai). The paged, block-based allocator described in the same post reduced that waste to under 4% and delivered up to 24x higher throughput compared to a naive Hugging Face Transformers serving baseline in the project's own benchmarks (vllm.ai). A separate serving system, **SGLang**, which introduces a KV-cache reuse technique called RadixAttention (sharing cached prefixes across requests via a radix tree), reports up to 5x higher throughput compared to existing systems including vLLM in its own benchmark (lmsys.org). Because both figures are self-reported by the systems' own authors under their own chosen workloads, they should be read as evidence that memory-management architecture materially affects throughput, not as a precise, workload-independent multiplier a reader can expect to reproduce.

Advertised context length and usable context length are not the same thing. The RULER benchmark, an independent long-context evaluation, tested 17 long-context LLMs (15 open-source models plus the closed-source Gemini-1.5-Pro and GPT-4) against their own claimed context windows, which ranged from 32K to 1M tokens. RULER's headline finding is that only about half of the evaluated models could maintain satisfactory performance even at a 32K-token length, well short of many vendors' advertised maximums (arxiv.org). This matters directly for the cost question this article addresses: a KV cache sized for a 128K or 1M-token context is a real, billable memory allocation regardless of whether the model can actually use that much context effectively, so provisioning for advertised context length without validating usable context length risks paying for capacity that does not translate into better answers.

Frontier API context windows, as officially documented in September 2026, illustrate how far "long context" has moved beyond the 32K to 128K range only a few years ago. Anthropic's current-generation Claude models are documented as offering a 1-million-token context window, with Anthropic's own documentation naming the accuracy tradeoff of very long contexts "**context rot**": as token count grows, accuracy and recall degrade even when the tokens technically fit in the window (platform.claude.com) (platform.claude.com). OpenAI's GPT-5 API model is documented with a 400,000-token context window (developers.openai.com), and Google's Gemini 2.5 Pro documentation states an input token limit of 1,048,576 tokens (ai.google.dev). All three figures are official, vendor-documented maximums as of the access dates in this article's citations; they describe the size of the context window the API will accept, not a guarantee that the model reasons equally well at every point within it, per the RULER findings and Anthropic's own "context rot" framing above.

Cost mitigation through caching is already a commercial reality, not only a research topic. IntuitionLabs' own prior analysis of DeepSeek's inference pricing describes a mechanism, prompt-prefix caching, that is conceptually adjacent to the KV cache problem discussed here: caching the neural activations for repeated prompt prefixes so

they do not need to be recomputed on every request (intuitionlabs.ai), which that article reports carries roughly a 90% price discount for cache-hit tokens versus the standard per-token rate (intuitionlabs.ai). This is a different mechanism from the request-local KV cache covered in this article (cross-request prefix caching versus within-request key-value caching), but it draws on the same underlying insight: recomputing attention state that has already been computed once is expensive, and eliminating that recomputation is where much of the industry's long-context cost engineering effort is currently concentrated.

Implications and Future Directions

The formula, hardware, and evidence above point to a few durable implications for teams evaluating or budgeting long-context LLM deployments. First, **architecture choice at the model-selection stage has a larger effect on serving cost than most infrastructure tuning that happens afterward**: the difference between 8 KV heads and 4 KV heads, or between standard attention and a latent-compression scheme like MLA, changes the KV cache bill by a multiple before a single quantization or paging optimization is applied, as *Table 1* and *Table 3* above illustrate. Teams selecting a model primarily for a long-context use case should treat published KV head counts and attention variant as a cost parameter, not just an accuracy parameter.

Second, **the gap between advertised and usable context length, documented by RULER, is itself a cost consideration**: provisioning GPU memory for a 128K or 1M-token KV cache that the model cannot reliably use at that length is a real infrastructure cost with no corresponding accuracy benefit, as the RULER results above illustrate. Anthropic's public acknowledgment of "context rot" suggests vendors are increasingly transparent about this tradeoff, which should inform how conservatively a team provisions for maximum context versus typical context (platform.claude.com).

Third, the techniques in *Table 3* are not mutually exclusive; production systems increasingly combine them (for example, GQA or MLA at the architecture level, FP8 quantization at the precision level, and paged allocation at the memory-management level, stacked together). For organizations without in-house infrastructure teams to run that kind of evaluation, this is the kind of architecture decision where an experienced technical advisor, brought in before infrastructure is committed rather than after, can prevent an expensive mismatch between a chosen model's cache profile and the hardware budgeted to serve it; IntuitionLabs' own published methodology for AI information-layer projects treats performance and cost as inseparable from quality, noting plainly that "users abandon slow tools or work around limits" when that tradeoff is mismanaged, and frames the choice between a natively hosted model, a managed retrieval layer, or a privately hosted model as a deliberate architectural decision rather than a default (intuitionlabs.ai) (intuitionlabs.ai).

Fourth, hardware trends are easing but not eliminating the constraint: the move from A100's 80 GB to H200's 141 GB and toward the GB200 NVL72 rack's 372 GB per superchip increases per-GPU headroom for KV cache, but the GB200 NVL72's 372 GB HBM3E specification applies to a Grace Blackwell Superchip containing two Blackwell GPUs—approximately 186 GB per GPU (nvidia.com) (nvidia.com) (nvidia.com). Because model context windows have grown from the tens of thousands to over a million tokens over a comparable period, memory pressure from long context is unlikely to disappear on hardware growth alone; it will continue to be managed jointly through hardware, architecture, and serving-software choices for the foreseeable future.

Frequently Asked Questions (FAQs)

What is the formula for calculating KV cache memory in a transformer model? The standard formula, as published by NVIDIA, is: total KV cache bytes = batch_size x sequence_length x 2 x num_layers x hidden_size x bytes_per_parameter, where the factor of 2 accounts for storing both keys and values (developer.nvidia.com). For models using grouped-query or multi-query attention, replace `num_attention_heads x head_dim` with `num_kv_heads x head_dim`, since only KV projections are cached.

How much memory does the KV cache actually use for a real model? It depends heavily on model architecture and context length. Hugging Face's own published figures show Llama 3 8B needs 0.125 GB of KV cache at 1,000 tokens but 15.62 GB at 128,000 tokens, while Llama 3 70B needs 0.313 GB at 1,000 tokens and 39.06 GB at 128,000 tokens (huggingface.co), a figure NVIDIA independently corroborates at roughly 40 GB for the 70B model at that context length (developer.nvidia.com).

What drives the cost of GPU memory for long-context inference? The cost driver is that KV cache memory and model weights share the same finite GPU HBM pool, and cache size scales linearly with context length, batch size, and layer/head count. Current data-center GPUs range from 80 GB (H100, A100) to 141 GB (H200) to 372 GB per superchip (GB200 NVL72) (nvidia.com) (nvidia.com) (nvidia.com), and on-demand hourly pricing for these GPUs ranged from about $3.99 to $5.99 per GPU-hour on the cloud providers checked for this article as of September 2026 (lambda.ai) (together.ai).

What are the main techniques for reducing KV cache memory usage? The main categories are architectural (grouped-query attention, multi-query attention, multi-head latent attention, cross-layer attention, sliding-window attention), precision-based (FP8 or INT8 KV cache quantization), and memory-management-based (paged allocation as in vLLM's PagedAttention, and token eviction or attention-sink methods such as H2O and StreamingLLM), each summarized in *Table 3* above.

What KV cache quantization techniques are available today? NVIDIA's TensorRT-LLM supports both FP8 and INT8 KV cache quantization and recommends FP8 for lower accuracy impact in most tested cases, reporting it enables a 2 to 3x larger batch size on H100 GPUs (nvidia.github.io) (nvidia.github.io). vLLM supports FP8 KV cache quantization in both a per-tensor and a more accurate per-attention-head scheme (github.com).

Does a longer advertised context window mean the model actually performs well at that length? Not necessarily. As the RULER findings above show, only about half of evaluated long-context models maintained satisfactory performance even at a 32K-token length, despite many claiming context windows up to 1M tokens, and Anthropic's own documentation names the degradation of accuracy at very long context "context rot" (platform.claude.com).

Conclusion

The KV cache is the mechanism that makes autoregressive LLM decoding computationally tractable, and it is also the single largest variable in the memory and dollar cost of serving long-context inference. Its size is governed by a simple, well-documented formula: twice the product of layer count, KV head count, head dimension, sequence length, batch size, and bytes-per-parameter, and every term in that formula is a lever an architecture or infrastructure decision can pull. Real published model specifications show the consequences directly: a 70B-

parameter model with grouped-query attention needs roughly 40 GB of cache for a single 128,000-token request, a number independently corroborated across two primary sources, while an architecture change like DeepSeek's multi-head latent attention can cut that cost by more than 90% for a comparable model. GPU hardware has grown from 80 GB per A100-class GPU to approximately 186 GB per GPU in the GB200 NVL72; the rack's 372 GB figure applies to a two-GPU Grace Blackwell Superchip, and cloud pricing for that hardware has itself moved by tens of percentage points within a single year, which means any cost estimate for long-context serving has a shelf life measured in months, not years. Software techniques, from paged memory allocation to precision quantization to token eviction, can each independently cut effective KV cache cost by a meaningful multiple. For organizations building or buying long-context LLM capability, the KV cache is not an implementation detail to leave to a vendor's defaults; it is a budget line that architecture, precision, and serving-software choices can each move by an order of magnitude.
