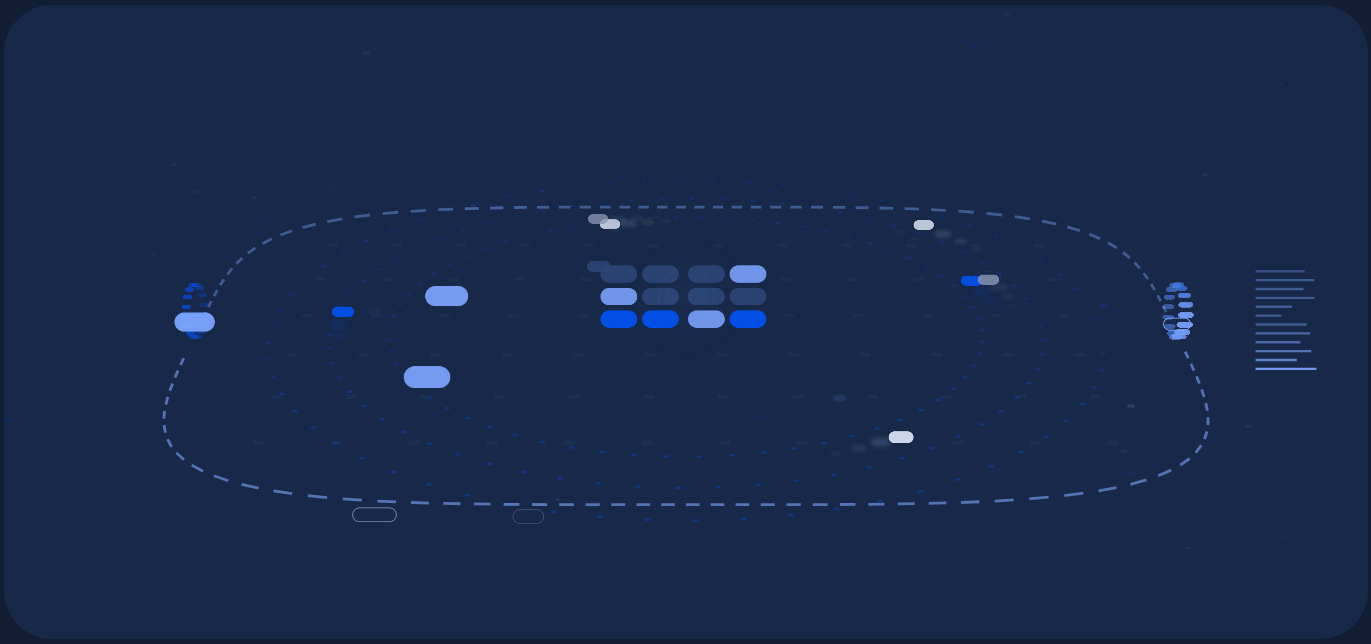
Rillor

METHODS NOTE

How much GPU memory LLM inference needs

LLM inference memory is weights plus a KV cache that grows with context and users, with the arithmetic for a 70B model on 80, 94 and 141 GB GPUs.



7 November 2026 Private SI cloud rillor.com/insights/gpu-memory-for-llm-inference rillor.com

Rillor

Contents

01 Summary

02 Background

03 Method

04 Findings

05 Implications

06 Limits of this analysis

07 Sources

Summary

- How much GPU memory LLM inference needs is set by two terms and a reserve: fixed model weights, a KV cache that grows with every token each active request holds, and room the serving engine keeps for activations and buffers [4, 17, 21, 31].
- Weights are parameters times bytes per parameter. Published 70B checkpoints of one architecture measure 131.42 GiB in BF16, 67.68 GiB in FP8 and 36.81 GiB with 4-bit weights, viewed 24 October 2026 [27, 28, 29].
- KV bytes per token are 2 x layers x KV heads x head size x bytes per element: 320 KiB per token in BF16 for a 70B model with 80 layers and 8 KV heads, or 40 GiB for one sequence of 131,072 tokens [1, 14]. Grouped-query attention makes this 8 times smaller than multi-head attention for this model, and latent attention and sliding windows cut further [3, 5, 7].
- In GiB on both sides, at vLLM's default budget of 0.92 of GPU memory, BF16 weights for a 70B model fit on no single H100 80GB, H100 NVL or H200. FP8 weights fit on each and leave 5.60, 18.42 and 61.49 GiB for activations and the cache [17, 24, 27, 28, 33, 34, 35, 36].
- An FP8 KV cache halves the KV term. With uncalibrated scales, vLLM's April 2026 evaluation recovered about 97 to 98% of baseline for a 70B model on a long-context test, after a kernel fix [15, 30].
- Two published guides, from NVIDIA and Hugging Face, overstate the cache for grouped-query models [31, 32].

Background

A served model holds its weights in high-bandwidth memory for as long as it runs. Each request also stores a key and a value vector for every layer and every token it processes, so later tokens can attend to earlier ones without recomputing them. That store is the KV cache. NVIDIA's 2023 guide to inference names weights and the KV cache as the two main contributors to an LLM's GPU memory requirement [31]. Pope et al., in their study of Transformer inference at scale, focus on the same two groups and note that other tensors in memory are much smaller [4]. TensorRT-LLM's documentation lists weights, internal activations and input and output tensors, with the KV cache the main input and output tensor [21].

The balance depends on load. In the PagedAttention paper's profile of a 13B model on a 40 GB A100, about 65% of memory held weights and close to 30% held the state of requests [9]. Pope et al. found that loading the KV cache dominates time from about 2,048 tokens and batch 512, where the cache of a 500B+ model with multi-head attention reaches 3 TB, three times the size of its parameters [4]. The vLLM project wrote in April 2026 that for standard full-attention decoders the KV cache often dominates GPU memory at 128K tokens and beyond [30].

Memory per GPU has grown more slowly than compute. The PagedAttention authors noted that from the A100 to the H100, FLOPS more than doubled while memory stayed at 80 GB at most [9]. NVIDIA now lists 94 GB for the H100 NVL and 141 GB for the H200 [23, 24]. Whether a model fits on one GPU decides whether a deployment needs a multi-GPU server, and the memory left decides how many users it serves [19].

Method

What was examined. Each term of serving memory: weights, quantized weights, the KV cache and the engine reserve. For the cache: the formula's inputs, attention design, context and concurrency, allocation waste and FP8 storage. For the worked example: the reported memory of three GPUs and the size of three published checkpoints.

Sources. Model papers for architecture inputs [1, 5, 6, 7, 8, 12] and one model family's reference code for head size and cache shape [14]. Attention papers [2, 3, 4], the PagedAttention paper [9] and quantization papers [10, 11, 13]. Engine documentation and source code for vLLM [15, 16, 17, 18, 19, 20], TensorRT-LLM [21] and SGLang [22]. GPU specifications [23, 24, 25, 26]. Checkpoint metadata on Hugging Face [27, 28, 29]. The vLLM FP8 KV evaluation [30]. Two secondary guides [31, 32]. Three pages that show nvidia-smi memory totals [33, 34, 35].

How. The formula comes from the attention table of a latent-attention paper [5] and the cache allocation in the 70B model family's reference code [14]. It was checked against three printed examples [9, 31, 32]. Weight sizes come from file metadata and GPU totals from nvidia-smi output, both viewed 24 October 2026. The fit test uses 0.92, vLLM's default memory fraction in its documentation that day and in its stable documentation on 7 November 2026 [17, 36].

Units. GB is 10^9 bytes, GiB 2^30, MiB 2^20 and KiB 1,024. Sources mix them: GPTQ states one model size "counting in multiples of 1024" [11]. Every comparison here puts both sides in GiB.

Labelling. Figures marked as this note's arithmetic are computed from the cited inputs. None is a measurement. The arithmetic leaves out activations, CUDA graphs and engine buffers unless stated, because no source gives them for this model and engine.

Findings

1. Weights: parameters times bytes per parameter

Weight memory is the parameter count times the bytes stored per parameter. NVIDIA's guide puts a 7B model in 16-bit precision at roughly 14 GB [31]. Hugging Face's release post for one model family, in 8B, 70B and 405B sizes, lists the 70B model at 140 GB in FP16, 70 GB in FP8 and 35 GB in INT4, for loading the checkpoint only, without reserved space for kernels or CUDA graphs [32].

Published files make the figure checkable. The page of the family's 70B instruction-tuned release lists 70,553,706,496 parameters, all BF16, in 30 safetensors files totalling 141,107,497,872 bytes: 141.11 GB, or 131.42 GiB [27].

Mixture-of-experts models are sized by total parameters. One mixture-of-experts model uses 13B active parameters per token, and its authors state that the memory costs for serving it "are proportional to its sparse parameter count, 47B" [12]. The latent-attention model in Finding 4 has 236B total parameters, 21B active per token [5]. Sizing from active parameters understates memory by the ratio of total to active [5, 12].

Weights alone can exceed a server. The family's authors write that in BF16 the 405B model "does not fit in the GPU memory of a single machine with 8 Nvidia H100 GPUs", so they served it on 16 GPUs in two machines [1].

2. Quantized weights

Quantization stores weights in fewer bits. The memory saved is a little below the bits removed, because some layers stay in 16 bits and per-group schemes also store scales [11, 28, 29].

70B checkpoint	Parameters by type	Size of the files	Share of BF16
BF16 checkpoint	70.55 billion BF16	141.11 GB (131.42 GiB)	100%
FP8 checkpoint, weights and activations	68.45 billion FP8, 2.11 billion BF16	72.67 GB (67.68 GiB)	51.5%
4-bit checkpoint, weights in groups of 128	68.45 billion 4-bit, 2.10 billion BF16	39.53 GB (36.81 GiB)	28.0%

Sources: [27, 28, 29], file metadata viewed 24 October 2026. The BF16 checkpoint is the family's 70B instruction-tuned release. The FP8 and 4-bit checkpoints are a third party's quantizations of another 70B instruction-tuned release by the same maker, with the same architecture. Shares are this note's arithmetic.

The publisher of both quantized checkpoints states that only the linear operators within the transformer blocks are quantized [28, 29]. The 4-bit checkpoint uses a symmetric per-group scheme with group size 128 and the GPTQ algorithm [29]. Its file headers store a BF16 scale for each group of 128 weights [29]. That is about 1.07 GB of the 39.53 GB, by this note's arithmetic. Its BF16 remainder of 2,102,665,216 parameters equals two 128,256 x 8,192 matrices (input embedding and output layer) plus the normalization weights, by this note's arithmetic. That also shows the released vocabulary is 128,256 tokens, which Table 3 of the family's paper rounds to 128,000 [1]. GPTQ's authors note the same for a 175B model at 3 bits: about 63 GB "including the embeddings and the output layer, which are kept in full FP16 precision" [11].

The publisher reports MMLU (5-shot) of 81.31 for FP8 and 80.62 for 4-bit weights, against 81.60 for that release unquantized, which is 99.6% and 98.8% recovery [28, 29]. The AWQ paper gives independent figures for the same maker's previous-generation 70B model. There, WikiText-2 perplexity is 3.32 in FP16, and with 4-bit weights in groups of 128 it is 3.46 for round-to-nearest, 3.42 for GPTQ and 3.41 for AWQ [10]. These are previous-generation results.

vLLM's documentation states that FP8 allows a 2x reduction in model memory with minimal impact on accuracy, and that FP8 computation needs NVIDIA compute capability 8.9 or higher [16]. The family's authors add a warning. Their FP8 recipe leaves the self-attention layers and the first and last transformer layers unquantized [1]. They write that standard benchmarks "do not adequately reflect the effects of FP8 quantization", and that with unbounded scaling factors "the model occasionally produces corrupted responses even though the benchmark performance is strong" [1].

3. The KV cache formula and its inputs

The latent-attention paper gives the cache per token for each attention design, in elements [5]. Multi-head attention caches 2 n_h d_h l, with n_h heads, head size d_h and l layers. Grouped-query attention caches 2 n_g d_h l, with n_g KV groups. Multi-query attention caches 2 d_h l [5]. Multiplying by bytes per element gives memory:

KV bytes per token = 2 x layers x KV heads x head size x bytes per element

The factor of 2 covers keys and values. For the family, Table 3 of its paper lists the 70B model with 80 layers, a model dimension of 8,192, 64 attention heads and 8 key/value heads [1]. The family's reference code computes head size as model dimension divided by attention heads, 128 here, and allocates key and value caches sized by KV heads [14].

Model in the family	Layers	KV heads	Head size	KV per token, BF16	128,000 tokens	131,072 tokens
8B	32	8	128	128 KiB	15.63 GiB	16 GiB
70B	80	8	128	320 KiB	39.06 GiB	40 GiB

Model in the family	Layers	KV heads	Head size	KV per token, BF16	128,000 tokens	131,072 tokens
405B	126	8	128	504 KiB	61.52 GiB	63 GiB

This note's arithmetic from Table 3 of [1] and the head-size rule in [14].

The formula reproduces printed examples. For a 13B multi-head model, the PagedAttention paper computes 2 x 5,120 x 40 x 2 bytes and calls it 800 KB per token [9]. NVIDIA's guide puts a 7B model at 4,096 tokens at about 2 GB [31], which is 2.0 GiB exactly by this note's arithmetic. Hugging Face gives the 70B cache in FP16 as 0.313 GB for 1K tokens and 39.06 GB for 128K [32], which matches 1,024 and 128,000 tokens at 320 KiB when "GB" is read as GiB, by the same arithmetic [1, 14].

Two sources go wrong for grouped-query models. NVIDIA's guide writes the formula with heads times head size and states that this is commonly the same as the hidden size [31]. That holds for multi-head attention. For the 70B model it gives 2.5 MiB per token, eight times the figure from its KV heads, by this note's arithmetic. The same guide's section on grouped-query attention says it reduces the key and value heads stored [31]. Hugging Face prints 0.984 GB for 1K tokens and 123.05 GB for 128K for the 405B model [32]. The formula gives 0.49 GiB and 61.52 GiB, half the printed values, by this note's arithmetic.

4. Grouped-query, multi-query and latent attention

The number of KV heads sets the size of the cache. Shazeer's multi-query attention shares keys and values across all heads, greatly reducing those tensors and the memory bandwidth of decoding with "only minor quality degradation" [2]. Ainslie et al. note that multi-query attention can lead to quality degradation and introduce grouped-query attention, with more than one KV head and fewer than the query heads [3]. Uptrained, it "achieves quality close to multi-head attention with comparable speed to MQA" [3]. Moving from H heads to one shrinks the cache by a factor of H [3], and G groups shrink it by H / G.

The family uses 8 key-value heads "to improve inference speed and to reduce the size of key-value caches during decoding" [1], and the 70B model card says all versions use grouped-query attention [27]. With 64 query heads, the 70B cache is one eighth of its multi-head size [1, 3]. Pope et al. show the effect on context. With 30% of memory reserved for the cache, a 540B model on 64 chips supports 1,320 tokens at batch 128 with multi-head attention and 43,000 with optimized multi-query attention, which they summarize as up to 32 times longer contexts [4].

Latent attention compresses further. A 236B mixture-of-experts model with latent attention caches a 512-dimension latent vector and a 64-dimension positional key per layer, across 60 layers [5]. Its paper puts the cache "equal to GQA with only 2.25 groups" and reports that, against the same maker's earlier 67B model, it "reduces the KV cache by 93.3%" [5]. The deployed model also stores each cache element in about 6 bits on average and its parameters in FP8 [5]. The 67B model uses grouped-query attention with 95 layers, 8 KV heads and model dimension 8,192 [6].

Attention design	KV elements per token	Example	KV per token, 16-bit
Multi-head	2 n_h d_h l	The 70B model's shape with 64 KV heads	2.5 MiB
Grouped-query	2 n_g d_h l	The 70B model as built, 8 KV heads	320 KiB
Multi-query	2 d_h l	The 70B model's shape with 1 KV head	40 KiB
Grouped-query	2 n_g d_h l	The earlier 67B model, 95 layers, 8 KV heads	380 KiB
Latent	(d_c + d_h^R) l	The 236B latent-attention model, 60 layers	67.5 KiB

Element formulas from [5]. Sizes are this note's arithmetic from [1, 5, 6].

By this arithmetic, the 67B model caches 389,120 bytes per token at 16 bits. The 236B model's 34,560 latent elements take 69,120 bytes at 16 bits, 82.2% less, or 25,920 bytes at 6 bits, 93.3% less. The paper's figure matches the deployed configuration with 6-bit storage. The latent design alone accounts for about 82% [5, 6].

Sliding windows cap the cache. One 7B model pairs 8 KV heads with a 4,096-token window, and its rolling buffer stops growing once a sequence passes the window. The authors report 8 times less cache memory on a 32K sequence, without impacting model quality [7]. Another model family uses one global layer for every five local layers with a 1,024-token span, to reduce "the KV-cache memory that tends to explode with long context" [8]. In its test on a 2B model with a 32K cache, KV memory overhead falls from 60% for global-only attention to under 15% [8].

5. Context length and concurrent users

The total cache is the per-token size times the tokens held by all active requests, prompt and output so far. Unlike the weights, the cache is unique for each sequence in the batch [4].

Load on the 70B model	BF16 KV cache	FP8 KV cache
32 users at 4,096 tokens	40 GiB	20 GiB
10 users at 8,192 tokens	25 GiB	12.5 GiB
10 users at 32,768 tokens	100 GiB	50 GiB

Load on the 70B model	BF16 KV cache	FP8 KV cache
1 user at 131,072 tokens	40 GiB	20 GiB

This note's arithmetic at 320 KiB per token in BF16 and 160 KiB in FP8 [1, 14].

The 70B model card lists a context length of 128K [27]. At that length one request's BF16 cache is 40 GiB, more than half the FP8 weights, and ten users at 32K tokens hold more than the FP8 weights, by this note's arithmetic [1, 14, 28]. In a long-context deployment the KV term sets the size.

vLLM's startup log reports the cache size in tokens and the maximum concurrency at a given request length. Its documentation shows a cache of 643,232 tokens supporting 15.70 concurrent requests of 40,960 tokens, and advises adding GPUs or nodes if these fall short [19]. With the model length set to auto, vLLM uses the full context if it fits and otherwise shortens it until it fits [17]. When the cache fills, vLLM preempts requests and recomputes them later, which can hurt end-to-end latency [18]. Its documented remedies are a higher memory fraction, fewer concurrent sequences or batched tokens, or more tensor or pipeline parallelism [18].

6. Fragmentation and paged allocation

Allocation decides how much of the cache holds real tokens. The family's reference code reserves cache tensors for the maximum batch size times the maximum sequence length [14]. NVIDIA's guide describes the same pattern: with a 2,048-token maximum, each request reserves 2,048 tokens regardless of its input and output [31].

The PagedAttention paper names three sources of waste: slots reserved for future tokens, internal fragmentation from provisioning for the maximum length, and external fragmentation from the allocator [9]. It found that existing systems used only 20.4% to 38.2% of their KV cache memory to store actual token states, against 96.3% in vLLM [9]. Storing the cache in fixed blocks, 16 tokens by default in the paper, limits waste to one block per request [9]. The paper reports near-zero waste and 2 to 4 times the throughput of earlier systems at the same latency [9]. With paged allocation, the arithmetic in Finding 5 is close to what the engine stores.

7. FP8 KV cache and its accuracy caveats

An FP8 cache halves the KV term: 160 KiB per token for the 70B model, by this note's arithmetic [1, 14]. vLLM's documentation states that it "can significantly reduce its memory footprint", allowing more tokens, higher throughput and longer contexts [15]. It supports fp8_e4m3 on CUDA 11.8 and later and on ROCm, and fp8_e5m2 on CUDA 11.8 and later [15]. The KV type defaults to the model's own, and current options include 8-bit and 4-bit integer formats and NVFP4 [17].

Calibration matters. Without it, vLLM sets all scales to 1.0, and its documentation recommends calibrating with a dataset through llm-compressor [15]. It names sliding-window layers as more sensitive to KV quantization and provides a flag to leave them at the model's native type [15].

The vLLM project's evaluation of 22 April 2026 used per-tensor uncalibrated scales throughout, which its authors treat as a lower bound for accuracy [30]. On the MRCR long-context test to 128K tokens, the other 70B instruction-tuned release with FP8 KV and FP8 attention recovered about 97 to 98% of the baseline AUC, with both the BF16 and the FP8 model [30]. On reasoning tasks it reports at most 1 to 2 points of degradation [30].

The post also records problems. On Hopper GPUs the FP8 Flash Attention 3 kernel lost accumulation precision at long contexts: on a 128K needle-in-a-haystack task, accuracy fell from 91% for BF16 to 13% until a fix restored it near baseline [30]. Some models drift downward with uncalibrated scales, and the post advises calibration when that appears on the real workload [30]. It advises staying with BF16 for contexts under about 7K tokens, for head size 256 where prefill latency matters, where uncalibrated accuracy falls below 95%, and for models with many small sliding-window layers [30]. It concludes that FP8 KV is ready to be "the default starting point for many long-context vLLM deployments" [30].

Lower precisions exist. KVQuant reports under 0.1 perplexity degradation at 3 bits and a context of up to 1 million tokens for a 7B model on one A100-80GB [13], and the 236B latent-attention model deploys about 6 bits per element [5]. These are specific methods with their own kernels. No source read reports calibrated FP8 KV accuracy for the 70B model.

8. Worked examples: a 70B model on three GPUs

Inputs. NVIDIA lists 80GB at 3.35TB/s for the H100 SXM and 94GB at 3.9TB/s for the H100 NVL [23]. Its 188GB figure is for the H100 NVL with NVLink bridge [23], two 94GB cards by this note's arithmetic. It lists 141GB at 4.8TB/s for the H200 SXM and NVL, footnoted "Preliminary specifications. May be subject to change." [24]. The labels are close to binary units:

GPU	NVIDIA label	nvidia-smi total	GiB	GB
H100 SXM (H100 80GB HBM3, 700 W)	80GB	81,559 MiB	79.65	85.52
H100 NVL (400 W)	94GB	95,830 MiB	93.58	100.49
H200 (700 W, the SXM part)	141GB	143,771 MiB	140.40	150.75

Labels from [23, 24]. Totals from [33, 34, 35]. GiB and GB are this note's arithmetic. The H100 NVL total is one user's report. No H200 NVL total was found.

Mixing units misleads. A 141.11 GB checkpoint beside a "141GB" GPU looks like a near fit. In GiB it is 131.42 against 140.40 [27, 35].

Budget. vLLM's stable documentation gives 0.92 of GPU memory per instance as the default [36]. Its development documentation, viewed 7 November 2026, gives 1.0 when the extensible KV cache, sized from memory measured after warmup, is on, and 0.92 otherwise [17]. The test takes 0.92 of the reported total as the budget for weights, activations, CUDA graphs and cache. Other engines use other rules. TensorRT-LLM gives the KV cache 90% of the memory free after loading [21]. SGLang sets the static fraction for weights and the KV pool to GPU memory minus a reserve, over GPU memory, and uses 0.88 when it cannot detect GPU memory [22].

GPU	Budget at 0.92	Left after BF16 weights (131.42 GiB)	Left after FP8 weights (67.68 GiB)	Left after 4-bit weights (36.81 GiB)
H100 SXM 80GB	73.28 GiB	Does not fit	5.60 GiB	36.46 GiB
H100 NVL 94GB	86.10 GiB	Does not fit	18.42 GiB	49.29 GiB
H200 141GB	129.17 GiB	2.25 GiB short	61.49 GiB	92.36 GiB

This note's arithmetic from [17, 27, 28, 29, 33, 34, 35].

BF16 weights fit on none of the three. On an H200, 100% of reported memory leaves 8.98 GiB after BF16 weights, before activations or cache, by this note's arithmetic [27, 35]. FP8 weights fit on all three.

KV capacity. The space left also holds activations and CUDA graphs, so these token counts are ceilings.

GPU, FP8 weights	BF16 KV tokens	FP8 KV tokens	8K sequences, BF16 KV	32K sequences, BF16 KV
H100 SXM 80GB	about 18,300	about 36,700	2.2	0.6
H100 NVL 94GB	about 60,400	about 120,700	7.4	1.8
H200 141GB	about 201,500	about 403,000	24.6	6.1

This note's arithmetic at 320 KiB per token in BF16 and 160 KiB in FP8 [1, 14, 17, 28, 33, 34, 35].

With 4-bit weights, BF16 KV ceilings rise to about 119,500, 161,500 and 302,600 tokens, or 14.6, 19.7 and 36.9 sequences of 8K, by the same arithmetic [17, 29, 33, 34, 35]. NVIDIA's H200 page footnotes a throughput test of a 70B model with batch size 8 on one H100 SXM and 32 on one H200 SXM, without stating the precision [24].

Activations. No source read gives activation memory for the 70B model under a given engine and settings. TensorRT-LLM states that activation size is close to the maximum number of input tokens times a constant that depends on the network and its optimizations, and lists a known issue in which FP8 GEMMs use more [21]. vLLM profiles memory use and estimates CUDA graph memory at startup [18]. Setting the cache size by hand skips that step: too small a value caps concurrency, and too large a value fails at allocation [18].

9. Splitting the model across GPUs

When the weights do not fit, or the cache needs room, the model is split. TensorRT-LLM states that with 8-way tensor or pipeline parallelism each rank typically holds 1/8 of the weights, and that tensor parallelism normally uses memory more efficiently [21]. vLLM's guide describes tensor parallelism as freeing memory for the cache on each GPU at a cost in synchronization, and pipeline parallelism as splitting layers at a possible cost in latency [18].

The cache splits with the KV heads, up to a limit. In vLLM's QKV layer, "the key/value head may be replicated while the query heads are partitioned" when KV heads are fewer than query heads [20]. Once the tensor-parallel size reaches the KV head count, each GPU holds one KV head and further GPUs hold copies [20]. The GQA paper notes the same waste in standard sharding of multi-query models [3].

The 70B model in BF16	Weights per GPU	Left per GPU at 0.92	KV per token per GPU	KV tokens for the group
2 x H100 SXM 80GB	65.71 GiB	7.57 GiB	160 KiB	about 49,600
4 x H100 SXM 80GB	32.85 GiB	40.42 GiB	80 KiB	about 530,000
8 x H100 SXM 80GB	16.43 GiB	56.85 GiB	40 KiB	about 1.49 million
2 x H100 NVL 94GB	65.71 GiB	20.39 GiB	160 KiB	about 133,600
2 x H200 141GB	65.71 GiB	63.46 GiB	160 KiB	about 415,900

This note's arithmetic from [1, 17, 20, 27, 33, 34, 35], with weights split evenly and 8 KV heads divided among the GPUs.

By this note's arithmetic, two H100 80GB GPUs hold the BF16 weights and about six 8K sequences, and four hold about four 128K sequences. Past eight GPUs, per-GPU KV per token stays at 40 KiB [20].

vLLM's guidance runs in order: one GPU if the model fits, tensor parallelism within a node if it does not, and tensor plus pipeline parallelism across nodes [19]. For uneven splits, or GPUs without NVLink such as the L40S, it advises pipeline parallelism [19]. NVIDIA lists 900GB/s NVLink for the H100 SXM, 600GB/s for the H100 NVL and 900GB/s per GPU for the H200 NVL with a 2-way or 4-way bridge [23, 24].

Implications

For ML platform engineers. Compute the KV term from the model's architecture table, with KV heads and head size, and check it against one printed example. Do the fit test in GiB against the driver's total, then set context length and concurrency limits so the ceiling in Finding 8 holds with room for activations. Read the engine's startup log for its cache size and concurrency estimate [19]. Test an FP8 KV cache on the real workload with calibrated scales, and skip sliding-window layers on hybrid models [15, 30].

For infrastructure buyers. Per-GPU memory decides whether a model fits on one GPU and how much context and concurrency each GPU carries. In this arithmetic, a 70B model in BF16 needs two GPUs of the 80 to 141 GB class, and in FP8 one, with the cache then setting users per GPU. Newer systems carry more. NVIDIA lists 1,440 GB for the eight GPUs of a DGX B200, 180 GB each by division [25], and AMD lists 288 GB for the Instinct MI355X [26]. The same test applies with the reported total in place of the label. Sizing the hardware, platform and models is the Design step on Rillor's private superintelligence (SI) cloud (https://rillor.com/compute/private-si) page, where Rillor deploys and operates dedicated SI environments on infrastructure the customer controls.

For readers of sizing guides. A guide that multiplies by hidden size, or sizes a mixture-of-experts model by active parameters, can be wrong several times over. Check its formula on one model with a public architecture table, as Finding 3 does.

Limits of this analysis

- The worked examples are arithmetic from cited specifications and file sizes. No configuration was run or measured, and no throughput or latency is reported.
- No source read gives activation, CUDA graph or buffer memory for the 70B model. The token counts are ceilings that include that space.
- GPU totals come from nvidia-smi output on third-party pages, one a single forum post. No source read states how much of the reported total CUDA leaves usable. The H200 NVL total was not observed.
- NVIDIA marks the H200 specifications as preliminary [24].
- Checkpoint accuracy figures come from their publisher. The AWQ figures are for the previous-generation 70B model. The FP8 KV evidence for the other 70B release uses uncalibrated scales on one long-context test.
- Draft models for speculative decoding, adapters, multimodal encoders, prefix caching and CPU or KV offload change memory in ways this note does not quantify [17].
- Engine defaults change between releases. On 7 November 2026, vLLM's stable documentation gave 0.92 [36], and its development documentation gave 1.0 with the extensible KV cache on and 0.92 otherwise [17]. This note does not test the 1.0 setting.
- The note covers memory only. It covers no GPU price or availability and recommends no purchase.

Sources

1. The Llama 3 Herd of Models. Llama Team, AI @ Meta, arXiv:2407.21783v3, 23 November 2024. https://arxiv.org/html/2407.21783v3

2. Fast Transformer Decoding: One Write-Head is All You Need. Noam Shazeer, arXiv:1911.02150, 6 November 2019. https://arxiv.org/abs/1911.02150

3. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. Ainslie et al., Google Research, EMNLP 2023, arXiv:2305.13245v3, 23 December 2023. https://arxiv.org/abs/2305.13245

4. Efficiently Scaling Transformer Inference. Pope et al., Google, arXiv:2211.05102, 9 November 2022. https://arxiv.org/abs/2211.05102

5. DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model. DeepSeek-AI, arXiv:2405.04434v5, 19 June 2024. https://arxiv.org/html/2405.04434v5

6. DeepSeek LLM: Scaling Open-Source Language Models with Longtermism. DeepSeek-AI, arXiv:2401.02954, 5 January 2024. https://arxiv.org/abs/2401.02954

7. Mistral 7B. Jiang et al., Mistral AI, arXiv:2310.06825, 10 October 2023. https://arxiv.org/abs/2310.06825

8. Gemma 3 Technical Report. Gemma Team, Google DeepMind, arXiv:2503.19786, 25 March 2025. https://arxiv.org/abs/2503.19786

9. Efficient Memory Management for Large Language Model Serving with PagedAttention. Kwon et al., SOSP 2023, arXiv:2309.06180, 12 September 2023. https://arxiv.org/abs/2309.06180

10. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. Lin et al., MLSys 2024, arXiv:2306.00978v6, 25 April 2026. https://arxiv.org/abs/2306.00978

11. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. Frantar et al., arXiv:2210.17323v2, 22 March 2023. https://arxiv.org/abs/2210.17323

12. Mixtral of Experts. Jiang et al., Mistral AI, arXiv:2401.04088, 8 January 2024. https://arxiv.org/abs/2401.04088

13. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. Hooper et al., NeurIPS 2024, arXiv:2401.18079v6, 28 May 2025. https://arxiv.org/abs/2401.18079

14. Llama reference implementation, models/llama3/model.py. Meta, GitHub repository meta-llama/llama-models, viewed 24 October 2026. https://github.com/meta-llama/llama-models/blob/main/models/llama3/model.py

15. Quantized KV Cache. vLLM documentation, updated 3 August 2026. https://docs.vllm.ai/en/latest/features/quantization/quantized_kvcache.html

16. FP8 W8A8. vLLM documentation, updated 24 September 2026.

https://docs.vllm.ai/en/latest/features/quantization/llm_compressor/fp8/

17. Engine Arguments. vLLM documentation, updated 7 November 2026, viewed 7 November 2026. https://docs.vllm.ai/en/latest/configuration/engine_args.html

18. Optimization and Tuning. vLLM documentation, updated 20 August 2026. https://docs.vllm.ai/en/latest/configuration/optimization.html

19. Parallelism and Scaling. vLLM documentation, updated 15 October 2026. https://docs.vllm.ai/en/latest/serving/parallelism_scaling.html

20. vllm/model_executor/layers/linear.py (QKVParallelLinear), commit cba669d. vLLM project, GitHub, 24 October 2026. https://github.com/vllm-project/vllm/blob/cba669d45d127f56ed3e634074b620d2d16d05cd/vllm/model_executor/layers/linear.py

21. Memory Usage of TensorRT-LLM. NVIDIA, TensorRT-LLM documentation, viewed 24 October 2026. https://nvidia.github.io/TensorRT-LLM/reference/memory.html

22. Server Arguments. SGLang documentation, viewed 24 October 2026. https://docs.sglang.io/docs/advanced_features/server_arguments

23. NVIDIA H100 Tensor Core GPU. NVIDIA, viewed 24 October 2026. https://www.nvidia.com/en-us/data-center/h100/

24. NVIDIA H200 Tensor Core GPU. NVIDIA, viewed 24 October 2026. https://www.nvidia.com/en-us/data-center/h200/

25. NVIDIA DGX B200. NVIDIA, viewed 24 October 2026. https://www.nvidia.com/en-us/data-center/dgx-b200/

26. AMD Instinct MI355X GPUs. AMD, viewed 24 October 2026. https://www.amd.com/en/products/accelerators/instinct/mi350/mi355x.html

27. meta-llama/Llama-3.1-70B-Instruct, model card and file metadata. Meta, on Hugging Face, last modified 15 December 2024, viewed 24 October 2026. https://huggingface.co/meta-llama/Llama-3.1-70B-Instruct

28. RedHatAI/Llama-3.3-70B-Instruct-FP8-dynamic, model card and file metadata. Red Hat AI, on Hugging Face, last modified 9 June 2026, viewed 24 October 2026. https://huggingface.co/RedHatAI/Llama-3.3-70B-Instruct-FP8-dynamic

29. RedHatAI/Llama-3.3-70B-Instruct-quantized.w4a16, model card and file metadata. Red Hat AI, on Hugging Face, last modified 5 August 2026, viewed 24 October 2026. https://huggingface.co/RedHatAI/Llama-3.3-70B-Instruct-quantized.w4a16

30. The State of FP8 KV-Cache and Attention Quantization in vLLM. vLLM Blog, 22 April 2026. https://vllm.ai/blog/2026-04-22-fp8-kvcache

31. Mastering LLM Techniques: Inference Optimization. NVIDIA Technical Blog, 17 November 2023. https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization/

32. Llama 3.1 - 405B, 70B & 8B with multilinguality and long context. Hugging Face blog, 23 July 2024. https://huggingface.co/blog/llama31

33. Get started: Serverless GPU compute with H100 GPUs. Databricks documentation, updated 11 September 2026. https://docs.databricks.com/aws/en/machine-learning/ai-runtime/examples/tutorials/sgc-api-h100-starter

34. NVIDIA guest driver on PVE 8 + H100 94GB vGPU. Proxmox Support Forum, 24 October 2024. https://forum.proxmox.com/threads/nvidia-guest-driver-on-pve-8-h100-94gb-vgpu.156429/

35. Job Resource Utilization. Tufts University, TTS Research Technology Guides, viewed 24 October 2026. https://rtguides.it.tufts.edu/hpc/slurm/utilization.html

36. Engine Arguments, stable documentation. vLLM documentation, updated 1 November 2026, viewed 7 November 2026. https://docs.vllm.ai/en/stable/configuration/engine_args/

About this report

Published by Rillor on 7 November 2026. Superintelligence (SI) tools assist research and drafting, and every figure cites its source. Rillor builds and secures SI systems, supplies the data they learn from and delivers the infrastructure they run on. We apply SI to financial and compute markets.

Cite as: Rillor. How much GPU memory LLM inference needs. 7 November 2026. https://rillor.com/insights/gpu-memory-for-llm-inference

Notices

Rillor is not a registered investment adviser or commodity trading advisor and does not provide investment or trading advice. Research, forecasts, data and software described on this site are for research and engineering use. Nothing here is an offer or recommendation to buy or sell any security, commodity interest or digital asset. Past or simulated results do not indicate future results.

NVIDIA, AMD, Ceph, Kubernetes and related product names are trademarks of their owners and are used only to identify products. No affiliation or endorsement is implied.

Superintelligence (SI) outputs can be wrong. Important decisions should include human review.

Work with Rillor

Send us the workloads you want to run, the data they use, who uses them and where your hardware sits today. Contact us at rillor.com/contact?type=private-si or contact@rillor.com. More on private SI cloud: rillor.com/compute/private-si