

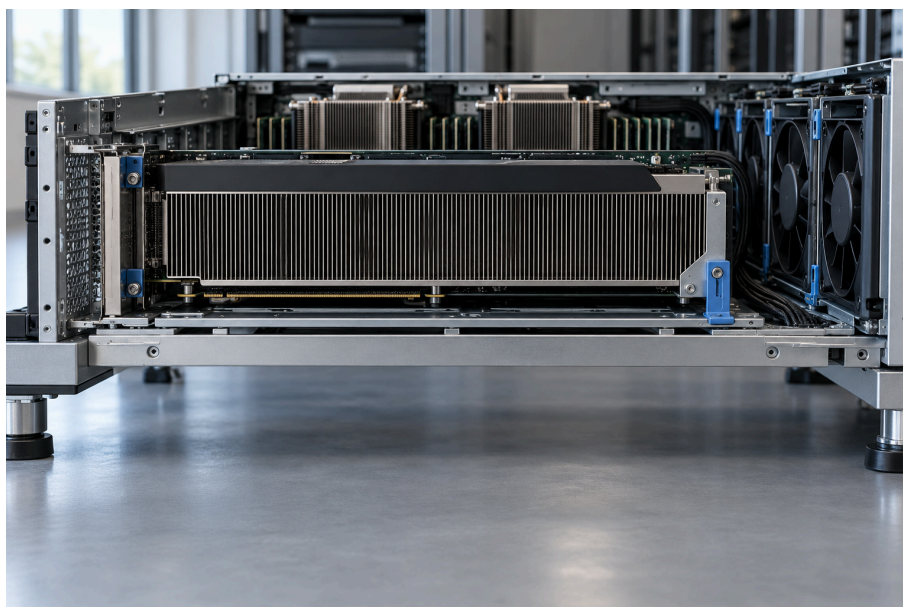


FP8 KV Cache Quantization: Capacity Gains and Accuracy Risk

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 · fp8 kv cache vllm · kv cache quantization accuracy long context · --kv-cache-dtype fp8_e4m3 · fp8 kv cache calibration llm-compressor · nvfp4 vs fp8 · tensorrt-llm fp8 kv cache

Original article: <https://gpusmith.com/articles/en/fp8-kv-cache-quantization-capacity-accuracy>



In this article

1. The short answer
 2. Why the KV cache sets long-context capacity
 3. What FP8 KV changes, and what it does not
 4. Flags per engine (verified 2026-10-28)
 5. Scales and calibration
 6. Attention backend interactions
 7. Accuracy risks to test for
 8. Research baselines below 8 bits
 9. Worked capacity example: Qwen3-32B on one H200
 10. Evaluation protocol and ship gate
 11. Rollout and rollback
 12. Sources
-

The short answer

Storing the KV cache in FP8 instead of BF16 halves the bytes per cached token, so the same KV budget holds about twice as many tokens. In the worked example below, that takes one H200 serving Qwen3-32B from 6 to 13 full 32K-token sequences. On the validated paths, the accuracy cost is small. The vLLM team's April 2026 study used uncalibrated per-tensor scales, the worst case. It reported that Llama-3.3-70B-Instruct recovered 97-98% of baseline AUC on the openai/mrcr long-context benchmark up to 128K tokens, and that Qwen3.5-27B lost at most 0.7 points on reasoning benchmarks ([vLLM blog, 2026-04-22](#)).

That depends on the attention kernel, scales, layer types and context length. The same study found a Hopper FP8 attention kernel bug that dropped a 128K needle-in-a-haystack score from 91% (BF16) to 13% before it was fixed ([vLLM blog](#)). So treat FP8 KV as a capacity lever that ships only after you test it on your own model, engine version and context lengths. Below: flags as of 2026-10-28, scale handling, capacity arithmetic and an evaluation protocol.

Why the KV cache sets long-context capacity

Weights are a fixed cost. The KV cache grows with every token of every active sequence. The PagedAttention paper put it this way: "the key-value cache (KV cache) memory for each request is huge and grows and shrinks dynamically" ([Kwon et al., 2023](#)). For a standard decoder with grouped-query attention, the per-token size is:

```
bytes_per_token = 2 (K and V) x num_layers x num_kv_heads x head_dim x bytes_per_element
```

Concurrency is then the KV budget divided by bytes per token, rounded down to the engine's block granularity. Our [KV cache memory calculator](#) and [vLLM KV cache capacity planner](#) walk through that sizing in BF16. Here we cover what changes when `bytes_per_element` drops from 2 to 1.

What FP8 KV changes, and what it does not

Capacity. With per-tensor scales, each K or V element takes 1 byte instead of 2. The scales are one value per tensor per layer, so their overhead does not grow with tokens. Token capacity roughly doubles. That is arithmetic, not a vendor claim. SGLang's documentation reaches the same ratio from the other direction: it gives NVFP4 about 3.56x the tokens of BF16 and about 1.78x the tokens of FP8, which implies FP8 holds 2x the tokens of BF16 ([SGLang Quantized KV Cache](#)).

Bandwidth. Each decode step reads the whole cache of each active sequence, so halving its size can lower inter-token latency (ITL) at long contexts. vLLM measured this for Llama-3.1-8B on one H100. The ITL slope fell to 54% of BF16, and FP8 began to beat BF16 on ITL at about 7K tokens. Under load (150 requests, concurrency 8, about 20K input and 2K output tokens), output throughput rose 14.9% ([vLLM blog](#)).

What it does not change. Weight memory, activation memory and the prefill compute budget stay the same. FP8 KV is independent of FP8 weights, and you can combine them (see [FP8 and INT4 inference costs](#) for weight formats).

E4M3 vs E5M2. E4M3 has 3 mantissa bits and a maximum normal value of 448 (SGLang's docs quote ± 240 , the maximum of the E4M3FNUZ variant used on AMD MI300-series GPUs). E5M2 has 2 mantissa bits and a maximum of 57,344 ([Micikevicius et al., FP8 Formats for Deep Learning, 2022](#)). With scaling, the extra mantissa bit usually matters more than range. SGLang recommends fp8_e4m3 "for better accuracy" ([SGLang](#)), and the vLLM study used E4M3 throughout.

Flags per engine (verified 2026-10-28)

The option names below come from each project's official documentation as published on 2026-10-28. The versions are vLLM stable docs (the latest PyPI release is 0.32.0, 2026-10-18), the SGLang docs (latest PyPI release 0.5.22), and the TensorRT-LLM docs build labeled 1.4.0rc2, cross-checked against v1.2.1 source (the latest stable, 2026-04-20). Check flags against the version you pin.

Engine	How to enable FP8 KV	Accepted KV dtype values (relevant subset)	Scale source
vLLM	<code>--kv-cache-dtype fp8</code> (Python: <code>kv_cache_dtype="fp8"</code>)	auto, bfloat16, float16, fp8 (= fp8_e4m3 on CUDA), fp8_e4m3, fp8_e5m2, fp8_inc (Intel Gaudi), plus nvfp4, fp8_ds_mla, per-token-head and TurboQuant modes (v0.32.0 source)	Checkpoint scales from llm-compressor calibration, otherwise 1.0
SGLang	<code>--kv-cache-dtype fp8_e4m3</code>	auto, fp8_e5m2, fp8_e4m3, bf16, bfloat16, nvfp4, fp4_mx_block16	k_scale/v_scale in the checkpoint, or JSON via <code>--quantization-param-path</code> , otherwise 1.0

Engine	How to enable FP8 KV	Accepted KV dtype values (relevant subset)	Scale source
TensorRT-LLM	KvCacheConfig(dtype='fp8'), or kv_cache_config: dtype: fp8 in the YAML passed to trtllm-serve --config, as in NVIDIA's DeepSeek-R1 deployment guide (1.4.0rc2 container)	auto, fp8, nvfp4, prototype mixed_nvfp4_bfloat16 / _fp8 / _float16 (v1.2.1 maps only auto and fp8)	FP8: fixed 1.0 in the default attention backend; checkpoint scales are loaded for NVFP4 KV

Sources: [vLLM engine arguments](#), [vLLM Quantized KV Cache](#), [SGLang server arguments](#), [SGLang Quantized KV Cache](#), [TensorRT-LLM Quantization](#), [TensorRT-LLM KV cache](#), [trtllm-serve](#).

Engine-specific details that matter:

- **vLLM** documents fp8_e4m3 as supported on CUDA 11.8+ and ROCm, and fp8_e5m2 on CUDA 11.8+ only. ROCm accepts fp8 (= fp8_e4m3) ([engine arguments](#)). vLLM also has --kv-cache-dtype-skip-layers, which keeps chosen layers at the model's native dtype. It accepts layer indices or a layer type such as sliding_window ([vLLM docs](#)). The engine-arguments help also notes that some models, "namely DeepSeekV3.2", default to FP8 KV.
- **SGLang** lists --quantization-param-path as the "Path to the JSON file containing the KV cache scaling factors", and warns that without it the scales default to 1.0, "which may cause accuracy issues" ([server arguments](#)). The JSON is keyed by tensor-parallel rank, then by layer index. SGLang currently supports only per-tensor (scalar) FP8 scales ([SGLang](#)).
- **TensorRT-LLM** says you can "enable the FP8 KV cache manually, even for checkpoints that do not have it enabled by default". NVFP4 KV, by contrast, "requires" offline quantization with ModelOpt ([TensorRT-LLM Quantization](#)). The docs do not say where FP8 scales come from; the source does: layer FP8 KV scales "Stay at 1.0 (no-op) for the PyTorch backend" ([trtllm.py at v1.4.0rc2](#)). Its hardware matrix marks FP8 KV as supported on Ampere, Ada Lovelace, Hopper, Blackwell (sm100/103 and sm120) and Rubin (sm107). NVFP4 KV is marked only for Blackwell sm100/103.
- **ModelOpt option names differ by script**. SGLang's guide documents hf_ptq.py --kv_cache_qformat (default fp8) and notes that the default "may not be optimal for all use cases" ([SGLang Quantization](#)). TensorRT-LLM's guide uses scripts/huggingface_example.sh ... --quant fp8 --kv_cache_quant nvfp4 ([TensorRT-LLM Quantization](#)).

Scales and calibration

FP8 has limited range, so each K and V tensor is divided by a scale before it is cast. The scale choice is the main accuracy decision you control.

vLLM offers two scale granularities ([vLLM docs](#)):

- *Per-tensor*: one scale each for Q, K and V.
- *Per-attention-head*: k/v_scale = [num_kv_heads] and q_scale = [num_heads]. This "is currently available only with the Flash Attention backend and requires the calibration pathway provided by llm-compressor."

Without calibration, "All quantization scales are set to 1.0." vLLM labels dataset calibration with llm-compressor as "[Recommended]" for "the highest-quality quantization". Its example calibrates on 512 samples of HuggingFaceH4/ultrachat_200k at a maximum sequence length of 2,048. It sets kv_cache_scheme with strategy tensor or attn_head and saves a compressed checkpoint that carries the scales.

How much does calibration buy? The vLLM study's headline results all used scale = 1.0, which the authors call "the worst-case scenario for accuracy". Their advice is to start uncalibrated and calibrate if you see "persistent downward shift" across context lengths. Their example was Kimi-K2.5 on the FlashMLA backend on H200: the aggregate AUC drop was modest, but it was consistently negative across sequence-length buckets. Their stated threshold for considering calibration is uncalibrated accuracy below 95% of baseline on your workload ([vLLM blog](#)).

Calibration data should look like production traffic. The vLLM example caps calibration sequences at 2,048 tokens; if you serve 64K-token prompts, measure whether those scales hold. No source reviewed here quantifies it.

Attention backend interactions

FP8 KV only helps if the attention kernel reads FP8 directly. SGLang warns that if dequantization is not fused into the attention kernel, "performance can be extremely slow", possibly "negating the memory benefits" ([SGLang](#)).

In vLLM, the backend table ([attention backend feature support](#)) shows:

vLLM backend	FP8 KV dtypes listed	Compute capability
FLASH_ATTN (FA2)	none (auto, float16, bfloat16 only)	8.0 and above
FLASH_ATTN (FA3)	fp8, fp8_e4m3	9.x
FLASH_ATTN (FA4)	fp8, fp8_e4m3	10.0 and above
FLASHINFER (Native)	fp8, fp8_e4m3, fp8_e5m2	8.x-9.x
FLASHINFER (trtllm-gen)	fp8, fp8_e4m3, fp8_e5m2, nvfp4	10.x
TRITON_ATTN	fp8, fp8_e4m3, fp8_e5m2	any
ROCM_AITER_FA	fp8, fp8_e4m3, fp8_e5m2	N/A (ROCm)

Two consequences. First, fp8_e5m2 is not available on the FlashAttention backends, so asking for it can move you onto a different kernel. Second, with FlashAttention 3, "attention operations are also performed in the quantized (FP8) domain" and queries are quantized too ([vLLM docs](#)). You are testing FP8 attention math as well as FP8 storage.

That FP8 math is where the Hopper bug sat. Imprecise FP32 accumulation in FP8 Tensor Cores hurts when the contraction dimension (the context length, in the softmax-times-V product) reaches about 100K. vLLM added two-level accumulation, which brought needle accuracy back to 89%. It costs prefill speed, and for

head_dim = 256 (gemma-4-E2B) the TTFT quadratic coefficient was about 1.6x BF16's. The authors report that B200 with FlashInfer does not need the workaround ([vLLM blog](#)). The study compared vLLM v0.10.2 with v0.19.1. Older vLLM builds are therefore not evidence for or against FP8 KV quality today.

Accuracy risks to test for

Sliding-window and hybrid layers. vLLM's docs say "some attention layer types (e.g. sliding-window) are more sensitive to KV-cache quantization." For gpt-oss-20b (128-token windows), leaving those layers in BF16 was also faster: the ITL slope was 71% of BF16 with skip-layers versus 80% with full FP8 ([vLLM blog](#)). TensorRT-LLM's mixed NVFP4 prototype likewise keeps sliding-window layers native ([TensorRT-LLM KV cache](#)).

Long-context drift. The study's MoE results were less tight than its dense ones. Qwen3-30B-A3B-Instruct-2507 recovered about 94% of AUC@256K (BF16 weights) on H100/FA3 and about 93% on B200/FlashInfer, with the gap concentrated in the longest buckets ([vLLM blog](#)). SGLang notes that error accumulation "may become significant" in long-context use ([SGLang](#)).

Hard reasoning tasks vs easy ones. SGLang's published preliminary table puts FP8 E4M3 KV close to BF16 on GSM8K. Harder sets move more:

Model	Benchmark	BF16 KV	FP8 E4M3 KV
Qwen3-235B-A22B	gsm8k	0.9168	0.9181
Qwen3-235B-A22B	aime25	0.7733	0.7333
Qwen3-235B-A22B	gpqa_diamond	0.7010	0.6899
DeepSeek-R1-0528	aime25	0.5067	0.4934
GPT-OSS-120B	gpqa_diamond	0.5081	0.5434

Source: [SGLang Quantized KV Cache](#), citing its PRs #10078 and #12612. The AIME25 drop for Qwen3-235B-A22B (4 points) and the GPQA gain for GPT-OSS-120B (3.5 points) show how noisy small reasoning sets are. You need repeated runs before you can call a difference real.

Research baselines below 8 bits

Research methods go further than FP8. Treat these as background: they evaluate their own methods on older models.

- **KIVI** (2-bit, keys quantized per channel, values per token) reports "almost the same quality" on Llama, Falcon and Mistral, with 2.6x less peak memory including weights, up to 4x larger batch size and 2.35x-3.47x throughput ([Liu et al., 2024](#)).
- **KVQuant** (3-bit, pre-RoPE per-channel keys, non-uniform datatypes) reports under 0.1 perplexity degradation on Wikitext-2 and C4 for LLaMA, Llama-2, Llama-3 and Mistral. It also reports serving LLaMA-7B at up to 1 million tokens of context on one A100-80GB ([Hooper et al., 2024](#)).

In production engines, the next step below FP8 is NVFP4 KV. NVIDIA reports under 1% accuracy loss against BF16 and FP8 baselines on LiveCodeBench, MMLU-PRO, MBPP and Ruler 64K for Qwen3-Coder-480B-A35B. It also reports 5% higher MMLU for NVFP4 than for MXFP4 KV on Llama 3.3 70B ([NVIDIA, 2025-12-08](#)). SGLang's own table shows FP4 KV dropping GPT-OSS-120B on AIME25 from 0.7533 (BF16) to 0.3533. On "nvfp4 vs fp8" for KV: NVFP4 costs 0.5625 bytes per element including block scales, against FP8's 1 byte, so it holds about 1.78x FP8's tokens. The accuracy risk is visibly higher on smaller models and hard tasks. For weight and activation formats, see [NVFP4 vs MXFP4 vs FP8](#).

TensorRT-LLM also offers a middle option. It keeps the GPU cache in its normal dtype and compresses pages to NVFP4 only when they are offloaded to host or disk ("cold pages"). NVIDIA reports AIME25 unchanged at 90.0% and 90.2% for Qwen3.5-397B-A17B, with 28% and 61% of reused pages compressed ([TensorRT-LLM blog, 2026-10-14](#)).

Worked capacity example: Qwen3-32B on one H200

Model inputs (published config). Qwen3-32B's `config.json` lists `num_hidden_layers`: 64, `num_key_value_heads`: 8, `head_dim`: 128, `torch_dtype`: `bfloat16`, and no sliding window ([config.json](#)). The model card gives 32.8B parameters and 32,768 tokens of native context ([Qwen3-32B](#)). Every layer is full attention, so the simple formula applies.

Step 1, elements per token: $2 \times 64 \times 8 \times 128 = 131,072$ elements.

Step 2, bytes per token:

- BF16: $131,072 \times 2 = 262,144$ bytes (256 KiB)
- FP8: $131,072 \times 1 = 131,072$ bytes (128 KiB). Per-tensor scales add a fixed few hundred bytes per model, not per token.

Step 3, bytes per 32,768-token sequence:

- BF16: $32,768 \times 262,144 = 8.59$ GB
- FP8: 4.29 GB

Step 4, KV budget. These are assumptions, labeled as such:

- GPU: one H200 with 141 GB of HBM3e ([NVIDIA H200](#)). The runtime sizes from the memory the device reports, which can differ from the marketing figure.
- vLLM's default `--gpu-memory-utilization` of 0.92 ([engine arguments](#)): $141 \times 0.92 = 129.72$ GB.
- Weights: the BF16 checkpoint's safetensors total 65.52 GB (Hugging Face file listing).
- Activations, CUDA graphs and other runtime overhead: 6 GB. This is an assumption; read the real figure from your engine's startup log.
- KV budget: $129.72 - 65.52 - 6 = 58.20$ GB.

Step 5, capacity:

Weights	KV dtype	Tokens in budget	Full 32K sequences
BF16 (65.52 GB)	BF16	$58.20e9 / 262,144 = 222,015$	6 (6.78)
BF16 (65.52 GB)	FP8	$58.20e9 / 131,072 = 444,030$	13 (13.55)
FP8 (34.32 GB, Qwen3-32B-FP8)	BF16	$89.40e9 / 262,144 = 341,033$	10 (10.41)
FP8 (34.32 GB)	FP8	$89.40e9 / 131,072 = 682,067$	20 (20.82)

The FP8 weight row uses $129.72 - 34.32 - 6 = 89.40$ GB. Real engines round down to whole cache blocks and reserve some headroom, so the measured number will be somewhat lower. Real concurrency depends on the length distribution, not the cap. With tensor parallelism, KV heads are split across ranks (8 KV heads at TP=2 means 4 per GPU), and per-GPU bytes per token halve again. Compute the limiting rank as the [capacity planner](#) describes.

Evaluation protocol and ship gate

This is a suggested protocol, based on the failure modes documented above. It is not a vendor standard.

1. **Pin everything.** Use the same engine version, attention backend (log the one the engine actually selects), weights checkpoint and sampling parameters for the baseline and the candidate. Change only the KV dtype and the scale source.
2. **Golden prompts.** Run a few hundred real production prompts at temperature 0. Diff the outputs and flag structural breaks: invalid JSON, failed tool calls, truncation.
3. **Long-context retrieval at your real lengths.** Run needle-in-a-haystack and a multi-turn retrieval set such as openai/mrcr in length buckets up to your maximum. Report per-bucket scores and an aggregate AUC. Look for a systematic downward slope, not single noisy buckets.
4. **Task evals with repeats.** Use the benchmarks closest to your workload, repeated enough times to separate signal from noise (the vLLM study used 5-10 repetitions per benchmark).
5. **Perplexity delta (optional).** Compare perplexity on held-out in-domain text as a cheap early-warning check. It does not replace task evals.
6. **Gate.** For example: no golden-prompt structural regressions, and long-context AUC and task scores at 95% or more of baseline (matching vLLM's stated threshold for considering calibration). If uncalibrated scales fail, calibrate and retest. If calibrated scales still fail, skip the sensitive layers or stay on BF16.

Rollout and rollback

Ship FP8 KV per model, not fleet-wide, and canary it. Watch TTFT and ITL percentiles (FP8 can slow prefill for head_dim = 256 or on unfused backends), KV utilization and preemptions, and quality proxies: tool-call failure rate, structured-output validity, user regenerations. Rolling back means restarting the server with `--kv-cache-dtype auto` (vLLM, SGLang) or `dtype: auto` (TensorRT-LLM). Capacity drops by half when you do, so confirm the BF16 replica count can carry the load before you need it. Recheck after every engine upgrade, because kernels, defaults and dtype lists changed several times between the releases cited here.

GPU Smith helps teams size and validate long-context inference clusters, including KV precision decisions; [contact us](#).

Sources

1. "Quantized KV Cache", vLLM documentation (stable), page updated 2026-08-03.
https://docs.vllm.ai/en/stable/features/quantization/quantized_kvcache.html
2. "Engine Arguments", vLLM documentation (stable), accessed 2026-10-28.
https://docs.vllm.ai/en/stable/configuration/engine_args.html
3. "Attention Backend Feature Support", vLLM documentation (stable), accessed 2026-10-28.
https://docs.vllm.ai/en/stable/design/attention_backends.html
4. Kübler, Kurtić et al., "The State of FP8 KV-Cache and Attention Quantization in vLLM", vLLM Blog, 2026-04-22. <https://vllm.ai/blog/2026-04-22-fp8-kvcache>
5. "Quantized KV Cache", SGLang documentation, accessed 2026-10-28.
https://docs.sglang.io/docs/advanced_features/quantized_kv_cache
6. "Server Arguments", SGLang documentation, accessed 2026-10-28.
https://docs.sglang.io/advanced_features/server_arguments.html
7. "Quantization", SGLang documentation, accessed 2026-10-28.
https://docs.sglang.io/advanced_features/quantization.html
8. "KV Cache System", TensorRT-LLM documentation (1.4.0rc2 build), accessed 2026-10-28.
<https://nvidia.github.io/TensorRT-LLM/features/kvcache.html>
9. "Quantization", TensorRT-LLM documentation (1.4.0rc2 build), accessed 2026-10-28.
<https://nvidia.github.io/TensorRT-LLM/features/quantization.html>
10. "trtllm-serve", TensorRT-LLM documentation, accessed 2026-10-28. <https://nvidia.github.io/TensorRT-LLM/commands/trtllm-serve/trtllm-serve.html>
11. "KV Cache Compression in TensorRT LLM", NVIDIA TensorRT-LLM tech blog (GitHub), 2026-10-14.
https://github.com/NVIDIA/TensorRT-LLM/blob/main/docs/source/blogs/tech_blog/blog30_KV_Cache_Compression_in_TensorRT-LLM.md
12. "Optimizing Inference for Long Context and Large Batch Sizes with NVFP4 KV Cache", NVIDIA Technical Blog, 2025-12-08. <https://developer.nvidia.com/blog/optimizing-inference-for-long-context-and-large-batch-sizes-with-nvfp4-kv-cache/>
13. Micikevicius et al., "FP8 Formats for Deep Learning", arXiv:2209.05433, 2022.
<https://arxiv.org/abs/2209.05433>
14. Kwon et al., "Efficient Memory Management for Large Language Model Serving with PagedAttention", arXiv:2309.06180, 2023. <https://arxiv.org/abs/2309.06180>
15. Hooper et al., "KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization", arXiv:2401.18079, 2024. <https://arxiv.org/abs/2401.18079>
16. Liu et al., "KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache", arXiv:2402.02750, 2024.
<https://arxiv.org/abs/2402.02750>
17. Qwen3-32B config.json, Qwen on Hugging Face. <https://huggingface.co/Qwen/Qwen3-32B/blob/main/config.json>
18. Qwen3-32B model card, Qwen on Hugging Face. <https://huggingface.co/Qwen/Qwen3-32B>

19. Qwen3-32B-FP8 model card, Qwen on Hugging Face. <https://huggingface.co/Qwen/Qwen3-32B-FP8>
20. "NVIDIA H200 Tensor Core GPU", NVIDIA. <https://www.nvidia.com/en-us/data-center/h200/>
21. Deployment Guide for DeepSeek R1 on TensorRT-LLM, NVIDIA TensorRT-LLM docs (1.4.0rc2 container), accessed 2026-10-28. <https://nvidia.github.io/TensorRT-LLM/deployment-guide/deployment-guide-for-deepseek-r1-on-trtllm.html>
22. `tensorrt_llm/_torch/attention/backends/trtllm.py` at v1.4.0rc2, NVIDIA/TensorRT-LLM, GitHub. https://github.com/NVIDIA/TensorRT-LLM/blob/v1.4.0rc2/tensorrt_llm/_torch/attention/backends/trtllm.py

Source links

Links cited in this article, in order of first appearance. Full addresses are provided for readers using extracted text.

1. <https://vllm.ai/blog/2026-04-22-fp8-kvcache>
2. <https://arxiv.org/abs/2309.06180>
3. https://docs.sglang.io/docs/advanced_features/quantized_kv_cache
4. <https://arxiv.org/abs/2209.05433>
5. <https://nvidia.github.io/TensorRT-LLM/deployment-guide/deployment-guide-for-deepseek-r1-on-trtllm.html>
6. https://docs.vllm.ai/en/stable/configuration/engine_args.html
7. https://docs.vllm.ai/en/stable/features/quantization/quantized_kvcache.html
8. https://docs.sglang.io/advanced_features/server_arguments.html
9. <https://nvidia.github.io/TensorRT-LLM/features/quantization.html>
10. <https://nvidia.github.io/TensorRT-LLM/features/kvcache.html>
11. <https://nvidia.github.io/TensorRT-LLM/commands/trtllm-serve/trtllm-serve.html>
12. https://github.com/NVIDIA/TensorRT-LLM/blob/v1.4.0rc2/tensorrt_llm/_torch/attention/backends/trtllm.py
13. https://docs.sglang.io/advanced_features/quantization.html
14. https://docs.vllm.ai/en/stable/design/attention_backends.html
15. <https://arxiv.org/abs/2402.02750>
16. <https://arxiv.org/abs/2401.18079>
17. <https://developer.nvidia.com/blog/optimizing-inference-for-long-context-and-large-batch-sizes-with-nvfp4-kv-cache>
18. https://github.com/NVIDIA/TensorRT-LLM/blob/main/docs/source/blogs/tech_blog/blog30_KV_Cache_Compression_in_TensorRT-LLM.md
19. <https://huggingface.co/Qwen/Qwen3-32B/blob/main/config.json>
20. <https://huggingface.co/Qwen/Qwen3-32B>
21. <https://www.nvidia.com/en-us/data-center/h200>
22. <https://huggingface.co/Qwen/Qwen3-32B-FP8>

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.