GPU SMITH
COMPUTE ENGINEERING

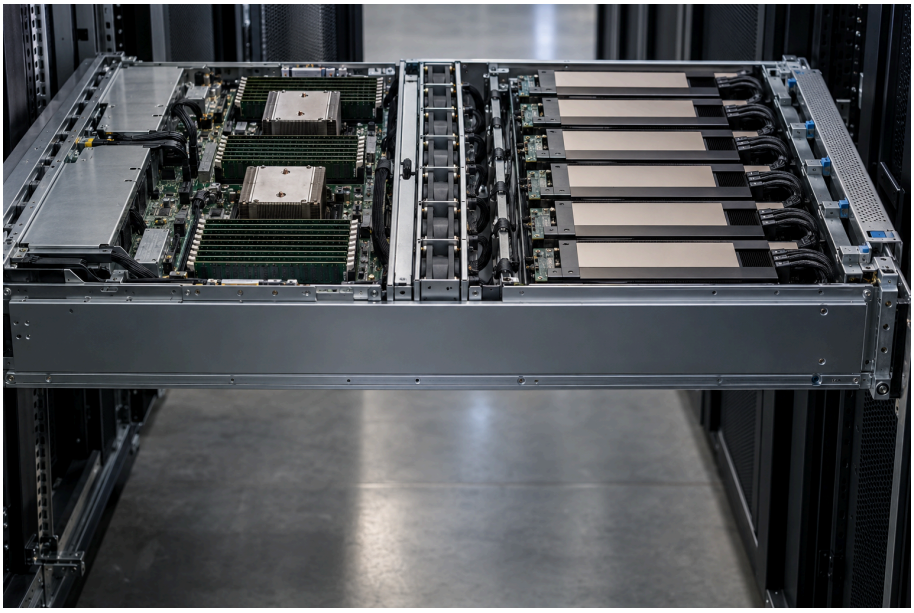
GPU SMITH / RESEARCH & PRACTICAL GUIDANCE

vLLM KV Cache Capacity Planning: Context and Concurrency

> 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 · 2026-10-05 · vllm kv cache capacity planning · vllm kv cache memory calculation · vllm concurrency · kv cache calculator · max model len · max num seqs · fp8 kv cache · inference optimization · gpu memory

Original article: https://gpusmith.com/articles/en/vllm-kv-cache-capacity-planner



In this article

1. Executive Summary
2. Introduction and Background
3. Key Changes in the Capacity-Planning Model
4. From Model Architecture to Concurrent-Token Capacity
5. Implementation Considerations and Process Changes
6. Preemption Monitoring and the Admission Gate
7. Parallelism and Nonstandard Cache Layouts
8. Data Analysis and Evidence
9. Case Studies and Real-World Examples
10. Implications and Future Directions
11. Frequently Asked Questions (FAQs)
12. Conclusion

Executive Summary

vLLM KV cache capacity planning starts with the memory actually assigned to the key-value (KV) cache, rather than the accelerator's advertised capacity. For ordinary full-attention layers, architecture-derived storage per cached token is **2 × layers × KV heads × head dimension × bytes per element**. The factors represent keys and values, all cache-bearing layers, and their stored scalar elements. Hugging Face publishes the corresponding half-precision calculation. (huggingface.co) Multi-query attention (MQA) uses a single KV head, while grouped-query attention (GQA) uses an intermediate count between MQA and multi-head attention (MHA). (arxiv.org) (arxiv.org)

The worked scenarios below are **hypothetical calculations, not benchmarks**. With 32 layers, eight KV heads, head dimension 128, and two-byte cache elements, storage is **128 kibibytes (KiB) per token**. An assumed 24 gibibytes (GiB) cache holds **196,608 tokens** before an analyst-selected 10% reserve and whole-block rounding. A request budget of 8,192 prompt tokens plus 2,048 output tokens then supports a conservative envelope of **17 simultaneous requests**, versus **34** with one-byte, 8-bit floating-point (FP8) storage under otherwise identical assumptions. The architecture equation supplies the calculation, and binary units use the National Institute of Standards and Technology (NIST) definition of GiB. (physics.nist.gov) (docs.pytorch.org) (docs.nvidia.com) These counts describe storage feasibility; they do not predict latency or throughput.

The decisive reconciliation is between the calculated envelope and vLLM's startup report. The documented `GPU KV cache size` is a simultaneous token capacity; its `Maximum concurrency` estimate uses the configured maximum sequence length. (docs.vllm.ai) That estimate is separate from a scheduler sequence cap and an upstream admission limit. Ray Serve, for example, independently bounds ongoing requests per replica and queued requests per deployment handle. (docs.ray.io) (docs.ray.io) A practical admission gate should reserve prompt plus bounded output tokens, round to cache blocks, and retain an explicit local reserve.

Documentation observed on **October 5, 2026** supports a canary process that changes one cache or scheduling control at a time. vLLM documents recomputation after cache-driven preemption. (docs.vllm.ai) Memory profiling must also account for allocator reservation and graph capture; PyTorch distinguishes allocated tensor memory from reserved allocator memory. (docs.pytorch.org) LMCache's calculator provides a useful tokens-to-memory and memory-to-tokens cross-check, but its answer should be reconciled to the deployment's actual graphics processing unit (GPU) allocation. (docs.lmcache.ai) Promote a canary only when its measured memory behavior, preemption deltas, and service-level objective (SLO) results meet locally defined acceptance criteria. Aggregate latency histograms correctly across replicas rather than averaging independent percentiles. (prometheus.io)

Introduction and Background

The admission-control question is specific: how many long-context requests can remain active on a self-hosted model before their cached history exhausts the remaining device memory? KV caching stores previous calculations for reuse during generation. (huggingface.co) It therefore creates a memory obligation that grows with retained history, alongside the model's other allocations. A model that loads successfully has passed an initialization check; the deployment still needs a workload-specific capacity decision.

This report separates **model weights**, **runtime allocations**, and **resident KV state**. A useful planning identity is available executor memory minus measured non-cache memory equals the automatic cache budget. The current vLLM GPU worker implements that allocation path using profiled non-KV memory and an applied CUDA-graph memory estimate. (docs.vllm.ai) The identity is a bookkeeping framework, not a universal percentage allowance. PyTorch's memory profiler also cannot see allocations made directly through CUDA application programming interfaces (APIs), including some communication-library allocations. (docs.pytorch.org)

GPU Smith's first-party description frames cluster sizing around workload models and throughput targets. (gpusmith.com) In this report, that adjacent engineering perspective means the outputs are a reproducible worksheet, an admission envelope, and a validation plan. No consulting delivery, customer outcome, or measured serving performance is inferred from that description.

LMCache explicitly offers a calculator for the memory needed to cache a specified number of tokens. (docs.lmcache.ai) Its utility is transparency at the token-storage boundary. The operator must still identify what memory budget is being entered, which cache layout is assumed, and whether stored state is GPU-resident or reusable from another tier.

The final decision has separate queue and device dimensions. Ray Serve exposes a per-handle queue limit in addition to its replica concurrency limit. (docs.ray.io) A platform should therefore document active request capacity, waiting capacity, and request-length policy separately. The following sections turn those distinctions into formulas and canary evidence, with version-sensitive behavior dated to the documentation actually inspected.

Key Changes in the Capacity-Planning Model

Replace advertised VRAM with a measured cache budget

Video random-access memory (VRAM) is not synonymous with cache bytes. NVIDIA advertises H200 with 141 GB of high-bandwidth memory and 4.8 TB/s memory bandwidth. (www.nvidia.com) Those specifications describe the device, not the memory remaining after a particular model and execution stack initialize. They cannot be substituted directly into a resident-token calculator.

Use a measured allocation boundary. PyTorch provides different measures for tensor allocations and the caching allocator's total reservation. (docs.pytorch.org) Its CUDA documentation also describes private pools for graph capture. (docs.pytorch.org) NVIDIA's graph guidance explains that peak capture memory determines the private pool reservation, including temporary tensors. (docs.nvidia.com) Consequently, a steady-state tensor snapshot alone is an incomplete estimate of the runtime footprint.

The planning record should make the following distinctions explicit:

- **Weights:** record the loaded representation and distribution across ranks.
- **Activations:** profile the intended prefill and batching regime.
- **Graphs:** record the capture configuration and retained memory pools.
- **Workspaces:** include library and communication allocations in the observed footprint.
- **KV pool:** use the resulting cache allocation as the worksheet input.
- **Reserve:** document the operator's remaining margin separately from runtime overhead.

These categories are motivated by documented allocator, graph, and workspace behavior; PyTorch describes retained cuBLAS workspaces per handle-and-stream combination. (docs.pytorch.org) The worksheet should avoid counting the same reservation twice when combining process-level and allocator-level measurements.

Separate allocation, request limits, and reuse

A manual cache allocation is a distinct control. The current vLLM CLI says that a supplied `kv_cache_memory_bytes` ignores `gpu_memory_utilization`. (docs.vllm.ai) Treating both as independently additive budgets would therefore produce a misleading capacity estimate. Record which allocation path the deployment uses before interpreting a flag change.

The request's logical history and its physical cache allocation also differ. Transformers' DynamicCache grows as generation progresses, whereas StaticCache preallocates a configured maximum size. (huggingface.co) (huggingface.co) These are explanations of cache behavior in Transformers, not guarantees about vLLM's allocator. They illustrate why a calculator must state its allocation model rather than assuming every serving stack reserves the same way.

Prefix reuse adds another distinction. vLLM's documented prefix-cache design caches full blocks, while LMCache describes storing reusable state in CPU memory. (docs.vllm.ai) (docs.lmcache.ai) Neither statement establishes a universal multiplier for concurrent GPU-resident requests. The conservative baseline should assume independent histories, with reuse credited only in a separately measured workload scenario.

From Model Architecture to Concurrent-Token Capacity

Derive storage per token

For uniform, conventional full-attention layers, define `L` as cache-bearing layers, `Hkv` as KV heads, `D` as head dimension, and `s` as bytes per stored element. Then `bytes_per_token = 2 × L × Hkv × D × s`. The leading factor accounts for keys and values. Hugging Face's published half-precision expression uses the same dimensions. (huggingface.co)

Query heads are not KV heads. MQA shares keys and values across attention heads, as described in the originating paper. (arxiv.org) GQA instead uses multiple KV heads, fewer than query heads. (arxiv.org) Substituting the query-head count into a GQA calculation can materially overstate cache storage. A model's parameter count or family name is an insufficient input to this equation.

Read the deployed model configuration and retain its revision. As a concrete cross-check, Qwen's official Qwen2.5-7B-Instruct card lists 28 query heads and four KV heads. (huggingface.co) The research also fetched its public configuration. This is an example of checking the architecture, not a claim that this model matches the hypothetical scenarios below.

The input checklist is:

- **Layer count:** include layers that actually allocate the modeled cache.
- **KV heads:** use the key-value count, not the query count.
- **Head dimension:** read the explicit field or verify the architecture's derivation.
- **Cache dtype:** use the stored cache type, independently of weight quantization.
- **Layer exceptions:** identify sliding-window, cross-attention, latent, or hybrid state.
- **Model revision:** preserve the configuration used for the calculation.

For a two-byte scalar, bfloat16 (BF16)'s documented representation is 16 bits. (docs.pytorch.org) NVIDIA describes FP8 E4M3 with a sign bit, four exponent bits, and three mantissa bits, totaling eight bits. (docs.nvidia.com) These representation widths explain the theoretical storage ratio, but actual cache allocation can also include scales, alignment, and implementation-specific state.

Convert bytes into admission reservations

Use exact byte units. NIST defines **1 GiB as 1,073,741,824 bytes**, distinct from decimal GB. (physics.nist.gov) If the supplied cache budget is `B` bytes and block size is `b` tokens, the homogeneous estimate is `T = b × floor(B / (b × bytes_per_token))`. LMCache's reverse calculator addresses the same memory-to-token question. (docs.lmcache.ai)

The recommended worksheet then computes `usable_tokens = b × floor((1 - reserve_fraction) × T / b)`. For request `i`, reserve `b × ceil((prompt_tokens_i + output_budget_i) / b)`. Admit it only when the sum of existing reservations plus this new reservation fits the usable budget. These are analyst-defined conservative rules, not a documented vLLM scheduling algorithm.

Prompt and output percentile budgets are useful planning inputs, but percentile sums are not automatically a joint percentile. Preserve a hard request-length cap or calculate the distribution of combined prompt and output demand. Build separate workload classes when long-document requests and short interactive requests have materially different reservations. Prometheus likewise cautions that independently calculated quantiles cannot be meaningfully averaged. (prometheus.io)

For mixed cache layouts, replace a single global coefficient with verified per-layer or per-group storage. Transformers documents that sliding-window and chunked layers stop growing at their window or chunk limit. (huggingface.co) A linear full-context equation applied indiscriminately to those layers is therefore a scenario assumption, not an exact architecture model.

Implementation Considerations and Process Changes

Treat controls as distinct canary hypotheses

Table 1 summarizes control meanings in documentation observed on **October 5, 2026**. The CLI defines model length as prompt plus output, documents manual cache-budget precedence, and exposes the scheduling controls; the quantized-cache documentation provides the FP8 compatibility caveats. (docs.vllm.ai) (docs.vllm.ai) (docs.vllm.ai)

Control	Capacity question	Recommended canary evidence
`--gpu-memory-utilization`	How much executor memory is available for automatic allocation?	Startup cache allocation, peak runtime memory, and headroom.
`--kv-cache-memory-bytes`	Is the cache budget explicitly fixed per GPU?	Actual allocation and measured non-cache footprint.
`--max-model-len`	What prompt-plus-output length is permitted?	Longest accepted request and startup length feasibility.
`--max-num-seqs`	How many sequences may be scheduled per iteration?	Running population, preemption, and latency under load.
`--max-num-batched-tokens`	How many tokens may be scheduled per iteration?	Prefill pressure, runtime memory, and latency distribution.
`--kv-cache-dtype`	What storage representation and backend are used?	Allocation change, supported kernels, and output-quality comparison.

The table is a set of experiment questions, not recommended universal defaults. In particular, a scheduler sequence cap is not an assurance that every permitted sequence can reach the maximum context simultaneously. The admission worksheet remains responsible for that storage obligation.

The current CLI's defaults can be resolved through engine configuration rather than a single universal value for all workloads. (docs.vllm.ai) Preserve the effective configuration printed by the exact deployed release. Avoid treating an old tuning article's flag defaults as a substitute for the current executable's help and startup output.

Reconcile allocation and logged concurrency

Capture the startup report before generating load. The documented maximum-concurrency estimate uses the configured maximum sequence length, while the token-capacity log describes simultaneous cache capacity. (docs.vllm.ai) For homogeneous full attention, compare logged tokens against byte-derived whole-block capacity. If they differ, investigate units, local rank layout, block rounding, and runtime cache grouping before changing admission limits.

Manual allocation still needs runtime validation. The current GPU worker skips the automatic cache-sizing profile when a manual byte budget is supplied, but still runs a profile step for compilation. (docs.vllm.ai) This is not evidence that the explicit budget will coexist safely with every later workload peak. The canary must exercise the intended prompt and batching regime.

Recommended deployment records include:

- **Build identity:** image digest, package version, and model revision.
- **Effective flags:** resolved settings, rather than remembered defaults.
- **Allocation:** bytes per rank and the startup token-capacity report.
- **Workload:** prompt and output distributions, arrival pattern, and cancellations.
- **Runtime:** device, driver, backend, graph mode, and cache representation.
- **Acceptance:** explicit memory, quality, and latency criteria.

Graph-capture memory can change the allocation boundary, so retain graph configuration beside the cache calculation. (docs.nvidia.com) Kubernetes GPU resource requests and limits, when both are set, must be equal; that pod resource assignment should not be interpreted as a token admission budget. (kubernetes.io)

Evaluate FP8 as a separate change

FP8 capacity gain is theoretical until measured. ROCm's versioned precision documentation notes encoding differences between MI300-native FP8 and NVIDIA H100 formats. (rocm.docs.amd.com) A dtype name alone is therefore not a complete compatibility specification. Verify the target device, attention backend, scale handling, and installed vLLM release together.

Scaling can add state: NVIDIA's Transformer Engine example uses a 32-bit floating-point (FP32) scale factor per tensor. (docs.nvidia.com) That is an illustration of low-precision representation overhead, not a statement of vLLM's exact scaling layout. Compare measured cache capacity and representative output quality before accepting a storage-based request-count increase.

Preemption Monitoring and the Admission Gate

Observe pressure and distinguish metric types

The current vLLM metrics documentation exposes `vllm:kv_cache_usage_perc`, running and waiting request gauges, and preemption metrics. Cache usage equal to **1 means full utilization**. (docs.vllm.ai) It also documents a cumulative `vllm:num_preemptions` counter and a `vllm:request_num_preemptions` histogram

for per-request preemptions. (docs.vllm.ai) Record the actual exported series on the deployed endpoint; registration names and exporter wire-format suffixes should not be assumed interchangeable.

A **counter** accumulates and may reset after restart, while a **gauge** can rise or fall. (prometheus.io) (prometheus.io) For the canary, calculate counter deltas or rates across an identified observation window, handle restarts, and retain denominators such as completed requests. A cumulative count without its window cannot establish whether a configuration change improved pressure.

The current vLLM logger separately describes statistics tracked over its local logging interval. (docs.vllm.ai) Keep those interval values distinct from lifetime cumulative series. Correlate cache occupancy, waiting population, preemption, and request latency rather than treating any single graph as the entire admission decision.

Use the following locally defined gate:

- **Length gate:** reject or reroute requests exceeding the configured length policy.
- **Token gate:** reserve block-rounded prompt and bounded output demand.
- **Sequence gate:** keep the running population within the scheduler's limit.
- **Queue gate:** cap waiting work independently of running work.
- **SLO gate:** retain the measured latency and quality criteria for promotion.
- **Recovery gate:** confirm restart and drain behavior before broad rollout.

For an outer serving layer, Ray documents separate per-replica ongoing and per-handle queued limits. (docs.ray.io) (docs.ray.io) Its queue bound includes the Serve proxy. (docs.ray.io) Translate the token gate into that layer deliberately; no numerical equivalence between a Ray request cap and vLLM cache capacity is established by those APIs.

Establish service-level evidence

Measure **time to first token (TTFT)**, generation latency, throughput, and the waiting population for the same workload window. Choose the SLO thresholds locally; the sources inspected do not supply a universal latency acceptance threshold for the hypothetical deployment. Prometheus supports aggregating histograms before computing a service percentile. (prometheus.io)

Do not average replica-specific percentiles to report service-wide latency. (prometheus.io) Preserve warm-cache and cold-cache conditions separately, together with offered load and completion counts. A run with fewer admitted requests can reduce preemption while worsening queue delay, so promotion should evaluate both the device and the waiting path.

An out-of-memory (OOM) failure requires a different diagnostic from cache pressure. PyTorch distinguishes reserved allocator memory from live tensor memory and documents that releasing unused cached allocations does not free occupied tensor memory. (docs.pytorch.org) Revisit runtime peaks and allocation accounting rather than assuming every OOM is repaired by lowering a logical context cap.

Readiness is a traffic gate: Kubernetes does not send Service traffic to a pod whose containers report not ready. (kubernetes.io) Use it to hold a newly started replica out of rotation until its allocation and health checks complete. Autoscaling may adjust desired scale based on observed metrics, but it is a separate control loop.

(kubernetes.io)

Parallelism and Nonstandard Cache Layouts

Calculate on the limiting rank

Tensor parallelism (TP) distributes model computation across devices; **pipeline parallelism (PP)** partitions layers. vLLM documents that sharding weights can leave more memory for KV cache and that these parallelism strategies introduce their own overheads. (docs.vllm.ai) The worksheet should therefore describe local cache-bearing layers and locally stored KV heads, rather than dividing a single-device answer by device count without checking the implementation.

For an evenly partitioned full-attention model (Hypothetical Example), local storage can be represented as `2 × local_layers × local_KV_heads × D × s`. Equal head sharding and equal layer partitioning are explicit assumptions. Replicated heads, uneven stage assignments, or other layouts require the actual local storage counts. Calculate every rank and let the most constrained rank determine the admission envelope.

Similarly, multiple cache groups require group-aware reconciliation. The current vLLM cache utility rounds request requirements to pages and can use a separate host pool whose smaller concurrency limit applies. (docs.vllm.ai) Consequently, one token-count division may not reconstruct the runtime's estimate for a hybrid layout. Use the runtime report as evidence and retain the architecture worksheet as an explanation of the assumed components.

Bound the applicability of the linear equation

Mistral's original Mistral 7B description combines GQA with sliding-window attention and a rotating cache bounded by the window. (mistral.ai) (mistral.ai) This is a documented architecture example, not a general statement that every model bearing the same family name has identical cache growth.

Speculative decoding also changes the resource model. IBM explains that proposals may come from a smaller draft model or part of the main model itself. (research.ibm.com) Its implementation describes sharing cached history across candidates and notes memory competition with dynamic batching. (research.ibm.com) (research.ibm.com) Do not assume either a duplicate complete cache for every candidate or zero additional state. Reprofile the specific speculation configuration.

Multimodal serving can include additional encoder state. As a version-scoped example, AWS Neuron's vLLM-Neuron encoder-cache design stores vision encoder outputs in preallocated high-bandwidth memory and reserves scratch space. (awsdocs-neuron.readthedocs-hosted.com) (awsdocs-neuron.readthedocs-hosted.com) That evidence concerns Neuron, not CUDA-vLLM byte allocation, but it demonstrates why text-only KV accounting should not be presented as complete multimodal memory sizing.

The implementation checklist is:

- **Sliding windows:** sum full and bounded layers separately.
- **Hybrid models:** identify all runtime cache groups and state types.
- **Latent layouts:** replace the ordinary KV-head equation with verified storage dimensions.
- **Speculation:** include proposal state and the selected backend's allocations.

- **Multimodal inputs:** include encoder caches and supported modality limits.
- **Parallel ranks:** record local storage and the limiting pool.

The worksheet should mark unverified components explicitly. LMCache's embedded calculator exposes additional precision inputs for a specialized cache layout, reinforcing that its model selection is more than a generic parameter-count lookup. (docs.lmcache.ai)

Data Analysis and Evidence

Transparent capacity worksheet (Hypothetical Example)

The following calculations isolate architecture and request demand. They assume **32 uniform full-attention layers**, **head dimension 128**, **24 GiB of available KV cache per GPU**, no sharing, and no TP or PP. Cache block size is an analyst-selected **16 tokens**, with a **10% capacity reserve** applied before another whole-block rounding step. These are scenario inputs, not vendor defaults or measured allocations. The equation is architecture-derived; dtype widths and GiB units come from the cited definitions. (huggingface.co) (docs.pytorch.org) (docs.nvidia.com) (physics.nist.gov)

Table 2 shows exact derived outputs. Prompt and output columns are reservations, not observed workload percentiles. Every number in the table is calculated from the stated hypothetical inputs.

Architecture / cache	KV heads	Bytes per token	Raw tokens/GPU	Usable tokens/GPU	Prompt + output budget	Request envelope
MHA, two-byte	32	524,288	49,152	44,224	8,192 + 2,048	4
GQA, two-byte	8	131,072	196,608	176,944	8,192 + 2,048	17
MQA, two-byte	1	16,384	1,572,864	1,415,568	8,192 + 2,048	138
GQA, one-byte FP8	8	65,536	393,216	353,888	8,192 + 2,048	34
GQA, two-byte, long	8	131,072	196,608	176,944	32,768 + 8,192	4
GQA, one-byte FP8, long	8	65,536	393,216	353,888	32,768 + 8,192	8
GQA, two-byte, extended	8	131,072	196,608	176,944	65,536 + 8,192	2

The comparison holds cache allocation constant to expose the effect of retained history and KV heads. It does not hold compute cost or output quality constant across architectures. In particular, the MQA row is a storage comparison, not a recommendation to replace an existing model; the original MQA paper describes shared keys and values as its architectural change. (arxiv.org)

A spreadsheet can reproduce the table with `B = 24 × 2^30`, `bytes_per_token = 2 × 32 × Hkv × 128 × s`, `raw_tokens = 16 × FLOOR(B / (16 × bytes_per_token))`, and `usable_tokens = 16 × FLOOR(0.90 × raw_tokens / 16)`. Then set `request_tokens = 16 × CEILING((prompt + output) / 16)` and

`request_envelope = FLOOR(usable_tokens / request_tokens)`. These are the report's formulas, using the documented binary-unit convention. (physics.nist.gov)

For a mixed workload, the admission test is the sum of individual reservations, not the number of requests multiplied by an average observed length. Output budgets must remain bounded. LMCache's memory-to-token calculator is a useful independent interface check, but the inspected interface did not expose its JavaScript formula or complete unit assumptions. (docs.lmcache.ai) Retain the worksheet's exact byte inputs instead of silently equating unlike calculator labels.

Canary record and promotion evidence

Table 3 is an unfilled evidence template, dated **October 5, 2026**. No rows contain measured results. Populate the startup columns using vLLM's reported cache tokens and maximum-concurrency estimate. (docs.vllm.ai)

Canary change	Predicted capacity	Startup evidence	Preemption evidence	SLO and quality evidence
Baseline	Worksheet bytes/token and token envelope	Record bytes/rank, tokens, and context denominator	Record start/end counters and observation window	Record offered load, completions, and latency histograms
Allocation control	Recalculate using proposed cache bytes	Compare actual allocation with prediction	Compare normalized deltas with baseline	Check memory peaks and local SLO criteria
Sequence/batch control	Keep storage equation; revise running limits	Confirm whether allocation also changed	Record pressure under the same workload	Check both queue delay and generation latency
Cache dtype	Recalculate scalar width and known overhead	Measure actual token-capacity change	Check pressure at controlled offered load	Compare output quality and latency with baseline

The template prevents predicted and measured capacity from being merged into one result. An outer replica limit remains a separate evidence field when Ray Serve is used. (docs.ray.io) For latency comparisons, aggregate histogram observations consistently rather than comparing mismatched percentile summaries. (prometheus.io) A pass means the local criteria were met under the recorded workload, not that the setting is transferable to another deployment.

Case Studies and Real-World Examples

Ceph's cold and reusable-cache test

Ceph's engineering article provides a neutral example of controlling cache state. Its experiment contrasts computational prefill with a **100% remote-storage cache-hit condition** across context lengths. (www.ceph.com) That comparison concerns TTFT and reused history; it does not establish a general GPU-resident concurrency increase.

The published procedure measures an initial computational run, restarts vLLM to clear GPU and CPU cache effects while retaining remote state, then measures retrieval from that state. Before the next context length, it stops serving and removes the remote cache blocks. (www.ceph.com) Those controls help distinguish a cold

computation path from a reusable-storage path.

Ceph also publishes configuration and command-line material for reproducing its experiment. (www.ceph.com) The inspected page did not provide a complete immutable build identity for every component. Its results should therefore remain attributed to the published setup rather than treated as evidence for a different current release.

For capacity planning, adapt the methodological distinction. First establish independent-history resident capacity. Then measure a separate shared-prefix workload, preserving the cache state at the beginning and end of each test. LMCache's CPU-offloading documentation describes retaining and reusing cache from CPU memory; its connector can store and load KV state. (docs.lmcache.ai) (docs.lmcache.ai) That reusable tier has its own storage and transfer behavior.

The recommended test record includes:

- **Cold state:** specify which device, host, and remote caches were cleared.
- **Warm state:** specify exactly which histories remain reusable.
- **Reuse pattern:** preserve prefix identity and request ordering.
- **Resident pressure:** measure GPU allocation and running population separately.
- **Transfer path:** record the participating host or remote tier.
- **Comparison window:** use matching workload and latency accounting.

This procedure should prevent a warm-start latency result from being interpreted as a larger physical GPU cache. The relevant outcome is a documented operating mode with its own capacity and latency evidence. Ceph's explicit remote-cache clearing step is a useful model for preserving that distinction between repeated test runs. (www.ceph.com)

Implications and Future Directions

Capacity planning should become a versioned operational artifact. Preserve the model configuration, calculation inputs, startup allocation, and canary record together. GPU Smith's description includes KV-cache and scheduler tuning within its serving-optimization scope. (gpusmith.com) The corresponding engineering implication is to make assumptions inspectable, rather than presenting a request count without the workload and runtime conditions that produce it.

The automation boundary should remain explicit. Kubernetes' HorizontalPodAutoscaler is a periodic control loop, with a documented default synchronization interval of **15 seconds**. (kubernetes.io) Autoscaling evidence does not eliminate the need for a bounded queue and an immediate admission policy at existing replicas. Ray's per-handle queue semantics likewise require care when a service has several ingress paths. (docs.ray.io)

Low-precision and speculative execution should have separate refresh triggers. ROCm's precision tables distinguish encodings and device support. (rocm.docs.amd.com) IBM's description shows that speculative decoding can use different proposal structures and compete with batching for memory. (research.ibm.com) (research.ibm.com) A planner that stores only model name and nominal dtype will miss implementation details that matter to the next deployment.

Recommended refresh events are:

- **Model change:** repeat configuration extraction and architecture accounting.
- **Runtime upgrade:** recheck flags, metrics, cache groups, and allocation logs.
- **Backend change:** revalidate kernels, scales, and graph memory.
- **Workload shift:** revisit prompt and output demand distributions.
- **Parallelism change:** recalculate local rank storage and limiting capacity.
- **Reuse change:** rerun cold and warm evidence under controlled cache state.

The report does not establish a single ideal safety reserve or preemption threshold. Those remain local decisions tied to workload risk and SLOs. Service-wide latency must be calculated from appropriate aggregate observations; Prometheus documents why averaging quantiles is statistically unsound. (prometheus.io) Keep that evidence beside capacity results so a storage improvement cannot silently substitute for a service improvement.

Frequently Asked Questions (FAQs)

What determines vLLM maximum concurrent requests?

Use the minimum of storage-feasible reservations, the scheduler's sequence allowance, the upstream admission limit, and the population that meets the local SLO. vLLM's startup concurrency estimate uses maximum sequence length, while Ray's ongoing limit is per replica. (docs.vllm.ai) (docs.ray.io) Neither alone establishes production concurrency.

Does reducing max-model-len automatically create more cache bytes?

Treat it as a request-length policy first. The CLI defines it as prompt plus output; the allocation path is controlled separately. (docs.vllm.ai) Reobserve startup allocation after changing it, since runtime profiling and execution configuration may also change. Distinguish a smaller request denominator from a larger physical cache pool.

Is a KV calculator a complete GPU-memory planner?

Its token-storage answer is a component estimate. LMCache documents memory-required and tokens-from-memory calculator modes. (docs.lmcache.ai) The operator must provide an appropriate cache budget and separately account for weights, runtime pools, and other state. PyTorch's allocator documentation distinguishes live tensors from reserved memory. (docs.pytorch.org)

Can FP8 double the admitted request count?

Halving stored scalar width doubles idealized token capacity when every other assumption remains fixed. Actual promotion requires measured allocation, supported kernels, quality, and SLO evidence. vLLM documents format and backend conditions for quantized KV cache. (docs.vllm.ai) The hypothetical table's arithmetic is a planning estimate, not a performance guarantee.

Conclusion

The useful output of vLLM KV cache capacity planning is an admission rule with a traceable allocation boundary. Start with the deployed architecture, distinguish KV heads from query heads, use the stored cache representation, and retain every assumption that changes local rank storage. The formula explains ordinary full-attention state; the runtime evidence determines whether that explanation matches the actual deployment.

Convert measured cache bytes into block-rounded token capacity, then reserve prompt plus bounded output demand for admitted requests. A worksheet and a memory-to-token calculator can cross-check that boundary. LMCache explicitly supports the reverse calculation from available GPU random-access memory (RAM). (docs.lmcache.ai) Keep safety margin as a documented local decision rather than presenting it as a vendor requirement.

Promotion should depend on a completed canary record. It should connect predicted capacity, startup allocation, pressure metrics, output quality, and correctly aggregated service latency. Prometheus' histogram guidance supports the aggregate latency evidence required for that comparison. (prometheus.io) Separate the active population from waiting work and preserve the workload distribution used to establish the result.

Finally, retain cold-cache and reuse scenarios independently, identify the limiting rank or cache pool, and repeat validation when models or execution settings change. The report's numerical examples show how context and architecture reshape storage demand. Only the deployment's completed evidence record can turn that estimate into an operational concurrency commitment.

Source links

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

1. https://huggingface.co/blog/kv-cache-quantization
2. https://arxiv.org/abs/2305.13245
3. https://physics.nist.gov/cuu/Units/binary.html
4. https://docs.pytorch.org/docs/2.8/tensor_attributes.html
5. https://docs.nvidia.com/deeplearning/transformer-engine/features/low_precision_training/fp8_current_scaling/fp8_current_scaling.html
6. https://docs.vllm.ai/en/latest/serving/parallelism_scaling/
7. https://docs.ray.io/en/latest/serve/api/doc/ray.serve.Deployment.html
8. https://docs.vllm.ai/en/latest/configuration/optimization/
9. https://docs.pytorch.org/docs/2.8/notes/cuda.html
10. https://docs.lmcache.ai/getting_started/kv_cache_calculator.html
11. https://prometheus.io/docs/practices/histograms/
12. https://docs.vllm.ai/en/latest/api/vllm/v1/worker/gpu_worker/
13. https://docs.pytorch.org/docs/2.8/torch_cuda_memory.html
14. https://gpusmith.com/
15. https://gpusmith.com/articles/en/hbm3e-vs-hbm4-ai-gpus

16. https://www.nvidia.com/en-us/data-center/h200/
17. https://docs.nvidia.com/dl-cuda-graph/latest/troubleshooting/memory-issues.html
18. https://docs.vllm.ai/en/latest/cli/serve/
19. https://huggingface.co/docs/transformers/main/en/kv_cache
20. https://docs.vllm.ai/en/latest/design/prefix_caching/
21. https://docs.lmcache.ai/getting_started/quickstart/offload_kv_cache.html
22. https://arxiv.org/abs/1911.02150
23. https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
24. https://docs.vllm.ai/en/latest/features/quantization/quantized_kvcache/
25. https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus/
26. https://rocm.docs.amd.com/en/docs-6.4.0/reference/precision-support.html
27. https://docs.vllm.ai/en/latest/usage/metrics/
28. https://prometheus.io/docs/concepts/metric_types/
29. https://docs.vllm.ai/en/latest/api/vllm/v1/metrics/loggers/
30. https://docs.ray.io/en/latest/serve/production-guide/best-practices.html
31. https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/
32. https://kubernetes.io/docs/concepts/workloads/autoscaling/horizontal-pod-autoscale/
33. https://docs.vllm.ai/en/latest/api/vllm/v1/core/kv_cache_utils/
34. https://mistral.ai/news/announcing-mistral-7b/
35. https://research.ibm.com/blog/speculative-decoding
36. https://awsdocs-neuron.readthedocs-hosted.com/en/v2.32.0/vllm-neuron/docs/design/multimodal/on_device_encoder_cache.html
37. https://docs.lmcache.ai/_static/kv_cache_calculator.html
38. https://www.ceph.com/en/news/blog/2025/vllm-kv-caching/

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.