GPU SMITH
COMPUTE ENGINEERING

GPU SMITH / RESEARCH & PRACTICAL GUIDANCE

Private RAG GPU Sizing: Embedding, Reranking, Latency

> 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 · 2026-11-12 · private rag gpu sizing · rag gpu requirements · embedding gpu throughput · reranker gpu requirements · text embeddings inference · tei max batch tokens · rag latency budget · reranker top-k latency · on-premise rag infrastructure · gpu co-location

Original article: https://gpusmith.com/articles/en/private-rag-gpu-sizing-embedding-reranker-latency



In this article

1. Executive Summary
2. Introduction and Background
3. Key Changes: From One GPU Pool to Three Retrieval Workloads
4. Implementation Considerations and Process Changes
5. Data Analysis and Evidence
6. Case Studies and Real-World Examples
7. Implications and Future Directions
8. Frequently Asked Questions (FAQs)
9. Conclusion

Executive Summary

Private retrieval-augmented generation (RAG) stacks run three unlike retrieval workloads next to the answer model: offline document embedding and re-indexing, online query embedding, and query-passage reranking. Each has its own rate unit, length distribution and failure mode, so a single "GPUs per user" figure cannot size them. This report, published by GPU Smith as an **independent advisor** (not a chip maker or cloud provider), treats published ratios as examples of structure and not as transferable capacity.

The sizing arithmetic is simple and should be done per stage. **Ingest tokens per second** equal total tokenized chunk tokens divided by the seconds left after setup and validation. **Reranker pairs per second** equal query arrivals per second times the observed mean pairs per query, counted after de-duplication across multi-query expansion. **Replica count** is the ceiling of required rate over measured, SLO-qualified per-replica capacity, plus an explicit failure scenario, because capacity planning "should be coupled with performance testing" (sre.google). **Hugging Face Text Embeddings Inference (TEI)** states that it cannot infer the batch-token setting because "the actual memory overhead depends on the model implementation" (huggingface.co), so every number must come from a sweep on the pinned model, runtime and GPU.

The published evidence shows why the stages scale differently. NVIDIA's reference guide gives its Scale 1X baseline as **2.75 GPUs**, with a half-GPU embedder and a full-GPU reranker (docs.nvidia.com). Its own pages state the embedding ratio as 1:40 in some places and 1:50 in others (docs.nvidia.com), which shows why the exact configuration and section must travel with any ratio. On NVIDIA's benchmark pages, a 1B reranker scores 40 passages of 512 tokens per request at **483.7 passages per second** on one H100 at FP8, versus **181.4** on an L40S (docs.nvidia.com), so a top-K change or a GPU change moves capacity by multiples. Sentence-Transformers warns that an embedding model sharing a GPU with a generative model "can contend for memory and compute" (sbert.net), and NVIDIA's guide says to avoid co-locating its LLM service with other GPU workloads (docs.nvidia.com). A cached vector hit or a small weight file proves nothing about the end-to-end 95th percentile (p95), because fan-out amplifies tails: with a one-second p99 per server and 100 servers, "63% of user requests will take more than one second" (cacm.acm.org).

The recommended path is: separate the three workloads on paper, pin the model contract, sweep token batch, concurrency and length, compute replicas per stage, then run a co-location test with generation and ingestion on and off. Split pools only when that evidence requires it. Because "Switching models requires re-

embedding all vectors in your collection" (qdrant.tech), keep the old index and model contract until the new one passes quality and latency gates.

Introduction and Background

Enterprise teams that host their own RAG stack, often in air-gapped or regulated environments, usually buy GPUs for the answer model first. Embedding and reranking arrive later as "small models" that are expected to fit in the leftover memory. That assumption deserves a test, because the three retrieval workloads are not small in aggregate. A re-index touches every chunk in the corpus. A query embedder sits on the critical path of every request. A reranker multiplies each request into tens or hundreds of scored pairs.

The question is a recurring one. A **GitHub issue** opened on **August 29, 2024** against the TEI repository has a user stating that they are trying to estimate how much GPU equipment is needed (github.com) for a reranker, and reports that their own load test observed about "5.4 TPS" on a partial A100 slice (github.com). That is an individual user's question and measurement, not a benchmark, and the page shows no maintainer answer. It is useful as evidence of the shape of the problem: capacity is asked for in documents per query and requests per second, and the answer depends on lengths and hardware.

This report is a standard-analysis piece. It does not walk through generic RAG architecture and does not compare answer-model token prices. It concentrates on retrieval service capacity while keeping generation and vector-database latency visible, because the buyer's real constraint is the response time of the whole pipeline. As of **November 2026**, the report relies on vendor documentation, preprints and peer-reviewed work, each cited for its own configuration. Where a source gives a number, the configuration travels with it. Where no public benchmark exists for a given model, runtime and GPU combination, the report says so and specifies the test instead.

The decision the report supports is concrete. Should query embedding, corpus (re-)embedding and reranking share the generation GPUs, run on CPU capacity, or use separate replicas? And what capacity is needed to meet an ingestion deadline plus online latency and quality requirements? The answer is a worksheet and a gate, not a table of GPU counts.

Key Changes: From One GPU Pool to Three Retrieval Workloads

Inventory the stages and fix the targets first

The first change is to stop describing "the embedding model" as one thing. The same weights can serve three workloads whose demand is independent. Table 1 summarizes the stage ledger that the rest of this report fills in. A team should complete every cell with measured, dated values before choosing hardware.

Table 1 below lays out the stage ledger: the rate unit, the length distribution to record, the capacity measurement that qualifies a replica, and who owns the number.

Stage	Rate unit	Length distribution to record	Measured accepted capacity (per replica, at SLO)	Owner
Offline ingestion and re-index	Tokens per second, chunks per second	Actual tokenized chunk tokens after de-duplication, including any document-side prompt prefix	Tokens per second at the batch-token and concurrency setting that keeps memory stable and finishes inside the window	Data platform
Online query embedding	Queries per second at p95	Query tokens plus instruction template, percentile length mix, cache hit state	Queries per second with p95 under the stage budget, with ingestion on and off	Search or RAG service team
Reranking	Query-passage pairs per second	Query plus passage tokens per pair, pairs per request after de-duplication	Pairs per second with p95 under the stage budget at the agreed top-K	Search or RAG service team
Vector search	Queries per second, recall at the agreed depth	Dimensions, quantization, filter selectivity	Latency at the working-set size actually resident in memory	Vector database owner
Generation	Time to first token, inter-token latency	Prompt tokens after context assembly, output tokens	Tail latency under the full retrieval load	Model serving team

The ledger is deliberately unit-specific. NVIDIA's own guide measures stage latency as a share of time to first token (TTFT), which NVIDIA's benchmarking documentation describes as including "request queuing time, prefill time, and network latency" (docs.nvidia.com). For capacity planning, shares are a diagnostic. The planning quantities are rates measured per stage.

The ledger notes that each row's capacity must be SLO-qualified, meaning measured with the p95 target holding. A throughput number taken at saturation with queueing is a different quantity from one taken at the latency target. The Google SRE book makes the same point about overload, noting that "queued requests consume memory and increase latency" (sre.google), which is what happens when a batch embedder is driven past its token budget.

Ingestion and re-index: a worklist, not a corpus size

Corpus re-embedding is a batch job with a deadline. The inputs are the exact chunk count and token-length distribution, the deduplicated changed content, the tokenizer and model revision, the retry policy and the processing window. The formula is:

- **Required ingest tokens per second** = total actual tokenized chunk tokens / available seconds after measured setup and validation.
- **Setup and validation** must be measured, not assumed, and include index build and a quality check on a held-out query set.
- **Retries** inflate the token total; count them in the worklist.

Vendor data shows how strongly length and precision move the answer. On NVIDIA's Embedding NIM performance page, a 1B embedding model on an H100 at FP8, with 300-token passages, batch 64 and concurrency 5, reaches **814.5 inputs per second**. At FP16 with the same lengths and concurrency it is **608.3**, and at FP8 with 512-token passages it falls to **504.3** (docs.nvidia.com). Those are NVIDIA's measurements for one model, one GPU and one set of settings. They illustrate that a 1.7 times longer passage and a precision change each move re-index time by tens of percent, which is why the length distribution is the first input to record.

Batching internals matter as much as the model. Hugging Face documents that padding to "the longest sequence in the batch" is the default behavior (huggingface.co), so one long input inflates the padded length of every item beside it. **Sentence-Transformers** documents that, where flash attention with variable-length support is present, it "skips padding for text-only inputs by concatenating them into a single sequence" (sbert.net), and that padding everything to a large fixed length "makes every call process the full length and is slower" (sbert.net). Length-sorted or token-budgeted batches are therefore part of the ingestion design.

NVIDIA's ingestion guide adds a pipeline observation: in its wiki bulk-ingest test each embedding instance "saturates at approximately CR=8 concurrent threads" (docs.nvidia.com). Its guide also recommends pre-computing embeddings off the serving path for its multimodal data. These are results for NVIDIA's stack, but the design lesson transfers: ingestion should be scheduled as its own pool or window, not as an unbounded neighbor of the online path.

Online query embedding: arrival traces, not corpus statistics

Query embedding is a latency-sensitive service with short inputs. Record these inputs before any sweep:

- **Arrival trace**, including bursts and the ratio of peak to mean.
- **Language and instruction template**, since both change token counts.
- **Cache state**, because a cached query vector never reaches the GPU.
- **Percentile length mix**, not only the mean.

The template is part of the model contract. The Qwen3-Embedding model card says queries use a task instruction while "No need to add instruction for retrieval documents" (huggingface.co), and the technical report states that the instruction is concatenated for the query "while leaving the document unchanged before processing" (arxiv.org). Changing the query instruction changes only query vectors, but it changes retrieval behavior and must be treated as a versioned change. TEI exposes a related control that prepends a default prompt to every input, so those tokens count toward batch tokens.

On NVIDIA's performance page a 20-token query on one H100 at FP8, batch 1, reaches **340.0 inputs per second** at concurrency 13 (docs.nvidia.com). NVIDIA's reference guide also reports that latency overhead for reranking, embedding and the vector database "increases as concurrency increases" (docs.nvidia.com). The planning consequence: measure the query embedder at the concurrency your gateway will actually present, with and without ingestion running, and take the p95 from that run.

Reranking fan-out: pairs, not queries

A cross-encoder or LLM reranker scores each query-passage pair separately. Sentence-Transformers describes the mechanism: "The query and a possible document is passed simultaneously to transformer network" (sbert.net), and warns that "Scoring thousands or millions of (query, document)-pairs would be rather slow" (sbert.net). The original monoBERT work scores "each passage independently" (arxiv.org) and caps query, passage and separator tokens at 512 (arxiv.org), so cost grows linearly with candidate count and with pair length. A shallow cross-encoder study states the resulting bound directly: scoring K candidates needs K model inferences (arxiv.org), so a latency window caps K.

The planning formula is:

- **Reranker pairs per second** = query arrivals per second x observed mean pairs per query.
- With multi-query expansion, count actual pairs after de-duplication, not queries x top-K.
- Token demand per pair is query tokens plus passage tokens plus any template tokens.

A production-style study of hybrid retrieval with query rewriting describes this fan-out: each fusion query's retrieved candidates are reranked by the same cross-encoder before rank fusion (arxiv.org), and the study reports added latency from "query rewriting and larger candidate" sets without matching downstream gains (arxiv.org). The relevance-quality sweep belongs in the same test, because K and the number of rewritten queries are quality parameters as well as capacity parameters.

NVIDIA's reranker pages give configuration-specific reference points. For a 1B reranker at 512 tokens, 40 passages per request and concurrency 5, an H100 at FP8 scores 483.7 passages per second (docs.nvidia.com), at FP16 it scores **347.2**, and an L40S at FP8 scores 181.4. A newer 8B reranker on the same kind of request (a query of about 20 tokens and 40 passages of about 512 tokens) is reported at **2933.4 ms** average latency on an H100 (docs.nvidia.com). The 8B figure is a latency for one request shape, while the 1B figures are throughput, so the two are not directly comparable. Model choice, not only top-K, changes rerank cost, and p95 for the chosen model must be measured.

Pin the model, runtime and hardware contract

Runtime support is a separate question from model-card availability. The **Qwen3-Reranker-0.6B** card loads the model as a causal language model, using AutoModelForCausalLM (huggingface.co), and the technical report frames reranking as point-wise scoring from the likelihood of a yes or no next token (arxiv.org). TEI's reranker path is documented for sequence-classification models with a single class (huggingface.co), and its supported-models page says it "currently supports CamemBERT, and XLM-RoBERTa Sequence Classification models" (huggingface.co) (the page's example table lists other architectures, so the prose and table differ). A TEI issue opened in June 2025 asks for Qwen3 reranker support, stating that TEI "does not support Qwen3ForSequenceClassification" (github.com). The status of that issue and the TEI release notes may have changed since it was filed, and the OpenSearch figures below come from the 2.9 documentation, so check the current versions before relying on either. Verify the exact version and image for your hardware instead of assuming support from a model card.

Table 2 below summarizes the contract fields to pin per model, with what the cited sources say.

Contract field	What to pin	Evidence and caveat
Embedding dimension and index width	Output dimension per index	Qwen3-Embedding-0.6B supports output dimensions "ranging from 32 to 1024" (huggingface.co); the 4B model is 2560 (arxiv.org) and the 8B model is 4096 (arxiv.org). A truncated dimension is a different index.
Instruction and prompt	Query template and document template	Query-side instruction only; BGE-M3 "no longer requires adding instructions to the queries" (huggingface.co).
Pooling and normalization	Pooling mode and normalization	TEI reads pooling from the model unless overridden: "the pooling configuration will be parsed from the model" (huggingface.co); Qwen3 embeds from the final end-of-sequence hidden state (arxiv.org).
Library version	Transformers or runtime version	Qwen3 needs Transformers 4.51.0 or later: earlier versions raise an error (huggingface.co).
Truncation and length rejection	Auto-truncate setting, max length	With auto-truncate off, "the server will refuse to start with an error" if model length exceeds the batch-token budget (huggingface.co).
Architecture and image	Image tag per GPU generation	TEI publishes per-architecture images (for example a Hopper-specific tag), and Flash Attention is off by default on the Turing image (huggingface.co).
Reranker architecture	Cross-encoder or causal reranker, and runtime	BGE reranker is a cross-encoder that takes "question and document as input and directly output similarity" (huggingface.co); Qwen3-Reranker uses a default instruction of "Given a web search query, retrieve relevant passages that answer the query" (huggingface.co).

The table's practical point is that a change in any embedding-side field changes the vectors, and the Qdrant migration guidance states plainly that "Switching models requires re-embedding all vectors in your collection" (qdrant.tech). Reranker changes do not touch the index, which is one reason to version the two components independently. The contract also decides precision: Sentence-Transformers measured plain FP16 at a median 2.92 times the throughput of FP32 on its test models and hardware (sbert.net), but precision changes can shift quality and must be re-tested on the relevance set.

Vector storage and the vector database stay visible

Vector storage is a lower bound plus overheads (see also the vector index RAM sizing guide). **Qdrant's** documentation gives the dense size as "dense_size = base * dimensions * bytes_per_dim" (qdrant.tech), where the base multiplies points by "replication_factor" (qdrant.tech) and float32 is "4 bytes (default)". Graph and bookkeeping are separate lines: "hnsw_size = base * m * 2 * 4 bytes * 1.2" (qdrant.tech) and "id_tracker_size = base * 52 bytes" (qdrant.tech), with headroom added "on top of your final RAM and disk totals" (qdrant.tech). Other engines use different formulas, for example OpenSearch's 2.9 documentation (an older release; current documentation may differ) gives "1.1 * (4 * dimension + 8 * M)" bytes per vector (docs.opensearch.org); pgvector stores a float32 vector in 4 x dimensions + 8 bytes (github.com); OpenSearch notes that "Having a replica doubles the total number of vectors" (docs.opensearch.org), and Elastic notes that "Each shard replica holds a full copy" (www.elastic.co). Elastic also states that the HNSW graph "must be fully loaded in memory for efficient search" (www.elastic.co), and the Faiss wiki notes that "all Faiss indexes are stored in RAM" (github.com). Quantization shrinks the footprint, with Elastic citing reductions of "4x, 8x, or as

much as 32x" (www.elastic.co) and Qdrant describing 4x scalar compression with "minimal accuracy loss" (qdrant.tech), but Qdrant also recommends binary quantization only "with rescoring enabled" (qdrant.tech), so quality must be re-measured. Compare the lower bound with measured index memory on your engine, version and settings, and add import workspace separately.

Latency lives here too. Elastic warns that "Once the working set no longer fits in cache, queries start reading from disk and latency climbs sharply" (www.elastic.co). A retrieval stage that benchmarks well on a warm cache can miss its budget after a re-index changes the working set.

Implementation Considerations and Process Changes

Sweep, then calculate replicas

Run sweeps on the pinned model and image, per stage. For embedding, vary the batch-token budget, concurrency and the length mix. TEI describes its maximum batch tokens as the critical control: "Overall this number should be the largest possible until the model is compute bound" (huggingface.co), but because memory overhead is model-dependent the safe value is found by test, with peak memory (including scratch) observed. TEI's token-based dynamic batching (huggingface.co) makes the token budget, not the request count, the unit that fills the GPU. Its concurrency limit is also a design lever, since a low limit refuses client requests, which is backpressure rather than silent queueing. Sentence-Transformers advises the same empirical approach: benchmark a few batch sizes on a small subset of your data (sbert.net).

For reranking, sweep K, pair length and concurrency, and record quality (for example a held-out relevance set) next to p95. The Pareto table to produce has columns top-K, queries expanded, mean pairs after de-duplication, tokens per pair, pairs per second at SLO, p95 for the stage and relevance score. Beyond a point, extra depth costs capacity without improving relevance; the hybrid-retrieval query-rewriting study above reports larger candidate sets and query rewriting adding latency without matching gains (arxiv.org); that is one study's data, not a general rule.

Then compute replicas from accepted throughput, not peak:

- **Replica count** = ceiling(required workload rate / measured SLO-qualified per-replica capacity).
- **Failure scenario:** add replicas so the stage still meets the SLO with the largest failure domain down. Google's SRE guidance describes N+2 as provisioning so peak traffic is handled "without making the user experience unacceptable" during a simultaneous planned and unplanned outage (sre.google).
- **Do not assume linear scaling.** NVIDIA's guide reports that stage latency overhead rises with concurrency, and shared bottlenecks (CPU tokenizers, the vector database, the network) appear as replicas are added. TEI's tokenization workers default to the number of CPU cores on the machine (huggingface.co), so host CPU is part of the capacity of a GPU embedder.

A worked deadline worksheet (Hypothetical Example)

The numbers below are invented inputs to show the arithmetic. They are not benchmarks.

(Hypothetical Example) A corpus has 12 million chunks, with a mean of 420 tokens per chunk after tokenization and prompt prefix, giving 5.04 billion tokens. The window is 72 hours, with 6 hours reserved for setup, index build and validation, leaving 66 hours or 237,600 seconds. Required ingest rate is 5.04 billion / 237,600, about **21,200 tokens per second**. If a measured, memory-stable, SLO-qualified rate for one replica is 9,000 tokens per second, the ceiling of 21,200 / 9,000 is 3 replicas, but losing one leaves 18,000 tokens per second, below the requirement, so the plan needs **4 replicas** or a longer window.

(Hypothetical Example) For reranking, assume a peak of 12 queries per second, 3 expanded queries with 25 candidates each (75 raw pairs) and 55 unique pairs after de-duplication, so the load is **660 pairs per second**. At 560 tokens per pair that is 369,600 tokens per second. If measurement gives 400 SLO-qualified pairs per second per replica, the ceiling of 660 / 400 is 2, and a failure scenario raises it to **3**.

(Hypothetical Example) For storage, 12 million chunks at 1024 dimensions in float32 is 12,000,000 x 1024 x 4 bytes = **49.15 GB** before replicas; the same corpus at 2560 dimensions is **122.9 GB** and at 4096 dimensions is **196.6 GB**. With Qdrant's published graph formula at m = 16, the graph is about 1.84 GB and the identifier tracker, at 52 bytes per point, about 0.62 GB. Replication, payload, headroom and import workspace are added on top and then compared with measured resident memory.

The co-location gate

Co-location is decided by a measured gate, not by weight size. The gate has four parts.

- **Isolation mechanism.** Decide how memory and fault isolation will be provided. MPS gives each client fully isolated GPU address spaces, but NVIDIA's MIG guide says MPS "does not offer error isolation between" clients (docs.nvidia.com). **MIG** is built "to provide a defined quality of service (QoS) with fault isolation for different clients" (docs.nvidia.com). **Time-slicing** "trades the memory and fault-isolation that is provided by MIG" (docs.nvidia.com).
- **Memory budget.** vLLM's memory fraction is a per-instance limit that applies only to the current vLLM instance (docs.vllm.ai), and its documentation says to set it explicitly when other processes share the GPU, including ones that allocate after it starts (docs.vllm.ai). Lowering it "takes memory away from the KV cache" (docs.vllm.ai), which shows how a co-located embedder reduces generation capacity. PyTorch's caching allocator keeps unused memory reserved, so it still appears used (docs.pytorch.org), and varying batch sizes can cause fragmentation (docs.pytorch.org).
- **Test matrix.** Run low, peak and burst demand with generation on, ingestion on and off, and include model reload and cold start. Record p95 for each stage and for the whole pipeline.
- **Decision rule.** Split pools only when the matrix shows the shared configuration missing a stage or end-to-end target, or when failure isolation requirements cannot be met.

A compatibility note matters here. **NVIDIA's reference guide** allocates the embedder 0.5 GPU "with Run:ai or MIG" (docs.nvidia.com), while the support matrix of its Embedding NIM 2.3 states that MIG mode clusters are "not supported" (docs.nvidia.com). These are different documents for different product versions, and the point is only that a partitioning plan must be validated against the runtime's own support statement.

Index migration, readiness and rollback

Treat a new embedding model, instruction, tokenizer, dimension or normalization as a new index contract:

- **Dual-run:** keep the old index and model serving until the new one passes retrieval quality and online SLO checks.
- **Cutover and rollback:** switch traffic, then retain the old index for rollback for an agreed period.
- **Readiness check:** three conditions must hold.
  - **Quality** on the relevance set at or above the baseline.
  - **Latency**: p95 per stage and end-to-end within budget with ingestion off.
  - **Headroom** for incremental updates and a failed replica.

Rerankers can change independently, but their input length and pair count feed capacity. Document each contract version and the ingestion run that produced the index.

Data Analysis and Evidence

The published evidence is useful for structure and weak for transfer. *Table 3* below collects configuration-specific reference points. Every row is a vendor-reported result for its own model, runtime, GPU and settings, and none is a general ratio.

Source and section	Configuration	Reported result	How to use it
NVIDIA Embedding NIM performance page	llama-nemotron-embed-1b-v2, H100, FP8, 300-token passages, batch 64, concurrency 5	814.5 inputs per second; FP16 608.3; 512-token passages 504.3 (docs.nvidia.com)	Shows length and precision sensitivity of re-index rate
NVIDIA Embedding NIM performance page	Same model, H100, FP8, 20-token query, batch 1, concurrency 13	340.0 inputs per second	Shows query embedding scales with concurrency, not with corpus size
NVIDIA Reranking NIM 1.11.0 performance page	1B reranker, 512 tokens, 40 passages per request, concurrency 5, H100 FP8 and FP16, L40S FP8	483.7, 347.2 and 181.4 passages per second	Shows GPU and precision change pairs per second by multiples
NVIDIA reranker 8B performance page	8B reranker, 40 passages of about 512 tokens, H100 FP16	2933.4 ms average per request	Reports latency only for the 8B reranker; the 1B rows are throughput, so the two are not directly comparable (different model generations)
Qdrant capacity planning	Float32 dense vectors	dense size = points x replication x dimensions x bytes	Lower bound, then measure
Sentence-Transformers efficiency page	RTX 3090 GPU and i7-13700K CPU, listed models	FP16 median 2.92x FP32; hardware: RTX 3090 and i7-13700K (sbert.net)	Direction of precision effect, not your throughput

The table is read as a pattern library. The same vendor reports 814.5 inputs per second for 300-token passages and 504.3 for 512-token passages on one GPU, a roughly 38% drop for a 1.7 times longer input. Precision moves the 1B reranker from 347.2 to 483.7 passages per second, about 39%, and the GPU class

moves it from 483.7 to 181.4. None of this transfers to another model. A newer 8B model at 40 passages per request is reported at 2933.4 ms, a reminder that p95 for a rerank call must be tested for the chosen model.

Discrepancies inside one published guide

NVIDIA's retrieval guide is a good example of why section and configuration must be preserved. The scaling page lists the embedding ratio as "1:40 (Embed:RAG)" (docs.nvidia.com), while the performance-results page gives a 1:50 ratio, and the summary page contains a 1:40 narrative and a table row of 1:50. The accuracy page gives reranker ratios of 1:2 or 1:4, while the scaling page gives 1:8 for chat and 1:4 for summarization (docs.nvidia.com), and the embedding share of TTFT is given as 3% to 9% in one table and 3% to 12% in another. The reasonable reading is that each figure belongs to a specific table and workload definition. A reader should cite the section, not a single ratio.

What the published data does not show

- **No universal ratio.** NVIDIA's ratios are tied to its Nemotron stack, chat and summarization workloads, and named GPUs; the guide does not give TEI, Qwen or BGE ratios.
- **Retrieval depth moves generation load.** Raising vector top-K from 4 to 10 increases the context sent to the model, and NVIDIA reports that each extra 1,000 tokens of context reduce output throughput by up to 25% in its low-latency range (docs.nvidia.com). Rerank depth is therefore a generation-capacity lever too.
- **Older latency data is dated.** BEIR reported dense retrievers "20-30x faster" than reranking models on 2021 hardware (arxiv.org), and re-ranking models achieve the best zero-shot results "at high computational costs" (arxiv.org); the ColBERT paper makes the same cost argument for sending "each query-document pair through a massive neural network" (arxiv.org).
- **Latency budgets cap depth.** In a 2024 shallow cross-encoder study, a full-scale BERT model at a 25 ms per-query limit reached NDCG@10 of only 0.431 on TREC DL 2019 (arxiv.org). That is one dataset and one model class, but it shows a latency window limits K.
- **CPU is hardware-dependent.** Sentence-Transformers reports that the best CPU backend differs by machine: OpenVINO INT8 led on a desktop CPU and llama.cpp on a cloud instance (sbert.net), and ONNX Runtime defaults its intra-op threads to physical cores (onnxruntime.ai). Intel reports INT8 BGE accuracy loss within 1% relative on its MTEB STS test (intel.com). CPU is a valid candidate for query embedding where measured, not a default.
- **Public speed tables are dated.** Sentence-Transformers' cross-encoder speed table states that runtime was computed on a V100 GPU with Transformers v4 (sbert.net), so absolute figures predate current GPUs and runtimes.
- **Generation can dominate multi-query pipelines.** In the original RAG-Fusion study, the extra latency of multi-query retrieval was attributed mostly to the second large language model call (arxiv.org), which is why retrieval fan-out must be traced next to generation.
- **Benchmarks differ in purpose.** The MTEB benchmark's speed figure was run on a single A100 80GB for STS15 (arxiv.org), which is quality context and not production throughput.

Case Studies and Real-World Examples

A published single-GPU implementation

An **EMNLP 2024** Industry Track paper on a power-plant maintenance RAG framework serves its embedding component with Hugging Face TEI, whose toolkit provides "functionalities such as a web server, load balancer," and a worker pool (aclanthology.org). The reported experiments ran on "an NVIDIA RTX A6000 48GB" (aclanthology.org). The paper is a neutral implementation reference. It shows a small private deployment using TEI for the embedding stage; it does not provide transferable throughput numbers for other corpora, lengths or models.

A user-reported sizing question

The TEI issue discussed earlier describes a reranker sized by documents per request and a target transaction rate, and the user's observed rate on their slice of an A100. It illustrates the missing inputs: chunk length distribution, request concurrency, batch settings and the latency target. This is why the sweep in this report records memory and utilization with every run.

Implications and Future Directions

Three implications follow. First, retrieval capacity planning should have its own owner and artifacts. The generation team's GPU budget will not reveal that a re-index consumed the embedder's memory headroom or that a top-K increase doubled rerank work. Second, end-to-end claims need per-request traces. **vLLM** exposes time to first token, inter-token latency and prefill as separate metrics, with time per output token computed as "(end-to-end latency - TTFT) / (number of output tokens - 1)" (docs.vllm.ai) and prefill as an interval (docs.vllm.ai); retrieval stages need the same granularity, with the trace joined per request. Adding stage p95 values does not produce the end-to-end p95, because tails do not add. The **Tail at Scale** paper shows that for a service with a one-second p99 on each of 100 servers, "63% of user requests will take more than one second", and that in a measured service the p99 for all fan-out requests to finish was "140ms" against a much smaller single-request p99 (cacm.acm.org). Rerank fan-out and multi-query retrieval are fan-out in exactly this sense. The paper also notes that "Eliminating all sources of latency variability in large-scale systems is impractical" (cacm.acm.org), and it reports that hedged requests cut a p99.9 from 1,800 ms to 74 ms in one measured case (cacm.acm.org); whether hedging is worth the extra load is a measured question for a GPU-bound reranker.

Third, queueing theory is a sanity check, not a substitute for measurement. Little's relationship between average concurrency, arrival rate and time in the system is credited in the MIT queueing text to J. D. C. Little as first proving it (web.mit.edu), and the SRE book applies it, noting that a frontend at 1,000 queries per second with 0.1 seconds of latency has about 100 worker threads busy (sre.google). Capacity should be measured in resources consumed, not just request counts: the SRE book recommends to "measure capacity directly in available resources" (sre.google). For retrieval that means tokens and pairs.

Operationally, burst handling and overload behavior matter more than average utilization. The SRE book calls overload "the most common cause of cascading failures" (sre.google) and warns that "Depending upon this slack CPU as your safety net is dangerous" (sre.google), which applies to borrowing idle generation GPU time

for ingestion. Looking forward, runtime coverage will keep changing: TEI lists Qwen3 among supported embedding architectures (huggingface.co) while reranker architectures are documented separately, so teams should re-check support with each version bump. A sound procurement sequence follows from this: set the targets, pin the contract, measure, then buy.

Frequently Asked Questions (FAQs)

How many GPUs do I need for RAG?

There is no universal count. NVIDIA's reference baseline is **2.75 GPUs** for a named stack, but that includes an answer-model GPU and specific lengths. Size generation, query embedding, reranking and ingestion separately: compute required tokens per second or pairs per second, divide by measured SLO-qualified per-replica capacity, round up and add a failure scenario.

What are the on-premise RAG GPU requirements?

Requirements start with the runtime. TEI needs drivers compatible with CUDA 12.2 or higher (huggingface.co). Then memory: model weights, batch-token scratch and the runtime's own overhead, which TEI says depends on the model implementation. Add host CPU for tokenization and index memory for the vector database.

How do I size embedding model GPU throughput?

Measure tokens per second on the actual chunk length distribution with the batch-token and concurrency settings swept, then divide the ingest token total by the available window. Published figures such as 814.5 inputs per second for one model at 300 tokens are references for method, not targets.

What are reranker GPU requirements?

The unit is pairs per second. Each pair is a full forward pass, so requirements scale with top-K, queries expanded and pair length. TEI's reranker path is for sequence-classification models, so verify runtime support for a causal reranker.

How does top-K change reranker latency?

Cost is linear in candidates in the cross-encoder design: K candidates need K inferences. Sentence-Transformers uses 100 candidates as an example depth (sbert.net). Measure p95 at each K together with relevance, and choose the knee of the curve.

What is a good end-to-end RAG latency budget?

Budget from the product requirement, then trace each stage per request. NVIDIA's guide proposes stage shares of TTFT for its stack: the LLM at 80% or more, reranking below 10%, embedding below 5% and vector search below 2% (docs.nvidia.com). Those thresholds are for that stack and workload.

How do I size self-hosted RAG infrastructure for batch embedding?

Use the deadline worksheet: tokens over seconds, after measured setup and validation. Run ingestion in a separate window or pool, and use the co-location gate before sharing it with online serving. Retain the old index until the new one passes.

Conclusion

Private RAG GPU sizing is three calculations that share a model family but not a unit. Ingestion is a deadline problem measured in tokens per second. Query embedding is a latency problem measured at the concurrency the gateway presents. Reranking is a fan-out problem measured in pairs per second, where top-K, query expansion and pair length are both capacity and quality parameters. The vector database and the answer model sit in the same trace, and their latency must stay visible, because a fast embedder or a warm vector cache does not prove that the p95 of the whole pipeline is met.

The method is consistent: pin model, instruction, dimension, normalization, truncation and image; sweep token batch, concurrency and length; compute replicas from measured, SLO-qualified capacity with an explicit failure scenario; run the co-location test with generation and ingestion on and off at low, peak and burst demand; and migrate indexes with the old contract retained for rollback. Published vendor figures remain their own configurations, and no number in this report should replace a measurement on the target hardware.

Source links

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

1. https://sre.google/sre-book/addressing-cascading-failures
2. https://huggingface.co/docs/text-embeddings-inference/en/cli_arguments
3. https://docs.nvidia.com/enterprise-reference-architectures/enterprise-rag-retrieval-scaling-and-sizing-guide/latest/rag-scaling-guidelines.html
4. https://docs.nvidia.com/enterprise-reference-architectures/enterprise-rag-retrieval-scaling-and-sizing-guide/latest/rag-retrieval-performance-results.html
5. https://docs.nvidia.com/nim/nemo-retriever/text-reranking/1.11.0/performance.html
6. https://sbert.net/docs/sentence_transformer/usage/efficiency.html
7. https://docs.nvidia.com/enterprise-reference-architectures/enterprise-rag-retrieval-scaling-and-sizing-guide/latest/summary.html
8. https://cacm.acm.org/research/the-tail-at-scale
9. https://qdrant.tech/documentation/tutorials-operations/embedding-model-migration
10. https://github.com/huggingface/text-embeddings-inference/issues/392
11. https://docs.nvidia.com/nim/benchmarking/llm/latest/metrics.html
12. https://docs.nvidia.com/nim/nemo-retriever/text-embedding/latest/performance.html
13. https://huggingface.co/docs/transformers/main/en/pad_truncation
14. https://docs.nvidia.com/enterprise-reference-architectures/enterprise-rag-ingestion-scaling-guide/latest/summary.html
15. https://huggingface.co/Qwen/Qwen3-Embedding-0.6B

16. https://arxiv.org/pdf/2506.05176
17. https://sbert.net/examples/sentence_transformer/applications/retrieve_rerank/README.html
18. https://arxiv.org/pdf/1901.04085
19. https://arxiv.org/pdf/2403.20222
20. https://arxiv.org/pdf/2603.02153
21. https://docs.nvidia.com/nim/nemo-retriever/reranking/latest/performance.html
22. https://huggingface.co/Qwen/Qwen3-Reranker-0.6B
23. https://huggingface.co/docs/text-embeddings-inference/quick_tour
24. https://huggingface.co/docs/text-embeddings-inference/en/supported_models
25. https://github.com/huggingface/text-embeddings-inference/issues/643
26. https://huggingface.co/BAAI/bge-m3
27. https://huggingface.co/BAAI/bge-reranker-v2-m3
28. https://qdrant.tech/documentation/capacity-planning
29. https://docs.opensearch.org/2.9/search-plugins/knn/knn-index
30. https://github.com/pgvector/pgvector
31. https://www.elastic.co/docs/deploy-manage/production-guidance/optimize-performance/approximate-knn-search
32. https://github.com/facebookresearch/faiss/wiki/Guidelines-to-choose-an-index
33. https://qdrant.tech/documentation/manage-data/quantization
34. https://huggingface.co/docs/text-embeddings-inference/en/index
35. https://sbert.net/docs/package_reference/sentence_transformer/SentenceTransformer.html
36. https://sre.google/sre-book/service-best-practices
37. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/concepts.html
38. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/introduction.html
39. https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/gpu-sharing.html
40. https://docs.vllm.ai/en/latest/configuration/engine_args
41. https://docs.pytorch.org/docs/stable/notes/cuda.html
42. https://docs.nvidia.com/nim/nemo-retriever/embedding/2.3/support-matrix.html
43. https://docs.nvidia.com/enterprise-reference-architectures/enterprise-rag-retrieval-scaling-and-sizing-guide/latest/rag-retrieval-accuracy-performance.html
44. https://arxiv.org/html/2104.08663v4
45. https://arxiv.org/abs/2104.08663
46. https://arxiv.org/abs/2004.12832
47. https://onnxruntime.ai/docs/performance/tune-performance/threading.html
48. https://intel.com/content/www/us/en/developer/articles/technical/efficient-natural-language-embedding-models.html
49. https://sbert.net/docs/pretrained-models/ce-msmarco.html
50. https://arxiv.org/pdf/2402.03367
51. https://arxiv.org/pdf/2210.07316

52. https://aclanthology.org/2024.emnlp-industry.61.pdf
53. https://docs.vllm.ai/en/stable/design/metrics
54. https://web.mit.edu/course/other/urban_or_book/OldFiles/www/book/chapter4/4.4.html
55. https://sre.google/sre-book/handling-overload

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.