



TensorRT-LLM for Private Deployments: trtllm-serve Guide

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 · tensorrt-llm · trtllm-serve · tensorrt-llm vs vllm · tensorrt-llm pytorch backend · tensorrt-llm fp8 nvfp4 quantization · tensorrt-llm kv cache free_gpu_memory_fraction

Original article: <https://gpusmith.com/articles/en/tensorrt-llm-private-deployment-guide>



In this article

1. What TensorRT-LLM is in October 2026
 2. Where it fits in a private stack
 3. The PyTorch-native architecture and what was removed
 4. Serving with trtllm-serve
 5. Telemetry: the first thing to switch off on a private cluster
 6. Hardware and precision support matrix
 7. Quantization recipes and when to pick each
 8. KV cache configuration
 9. In-flight batching, overlap scheduler and feature combinations
 10. Observability and operations
 11. TensorRT-LLM vs vLLM vs SGLang
 12. Adoption checklist and rollback
 13. Sources
-

What TensorRT-LLM is in October 2026

TensorRT-LLM (now styled "TensorRT LLM") is NVIDIA's Apache 2.0 open-source library for LLM inference on NVIDIA GPUs ([Overview](#); [LICENSE](#)). Current TensorRT-LLM is a PyTorch-native runtime. It loads Hugging Face checkpoints directly and serves them through trtllm-serve, an OpenAI-compatible HTTP server. The old "convert checkpoint, then trtllm-build an engine" workflow is gone from the development line but still present in the last stable release. Choosing between them is your first decision.

As of 2026-10-28 the release picture is:

Line	Latest tag	Published	Status
1.2	v1.2.1	2026-04-20	Latest non-prerelease (GitHub release v1.2.1)
1.3	v1.3.0rc29	2026-09-29	Release candidates only; no final 1.3.0 tag (GitHub releases)
1.4	v1.4.0rc2	2026-10-28	Pre-release (GitHub release v1.4.0rc2); main already reports 1.4.0rc3 (version.py)

The public docs track the development tree; on 2026-10-28 they were generated from the 1.4.0rc2 source ([Installation Guide](#)). Flags and defaults below come from those docs, with v1.2.1 differences checked against the tagged source.

Where it fits in a private stack

TensorRT-LLM is the runtime: execution, batching, KV cache, parallelism and quantized kernels. NVIDIA positions its LLM API as integrating with Dynamo and Triton Inference Server ([Overview](#)), which gives three ways to run it:

1. **trtllm-serve directly**, one server process (or one Slurm/MPI job) per model replica, behind your own gateway. This is the focus here.
2. **Inside Dynamo**, for disaggregated prefill/decode and KV-aware routing; the docs include a Dynamo Kubernetes example.
3. **Behind Triton**. Check packaging first: the 26.08 release notes say the TensorRT-LLM backend container "is not included in this release", and NGC lists no `-trtllm-` Triton tag after 26.07 ([Triton 26.08 release notes](#); [runbook](#)).

Lock-in is mostly hardware. The supported-hardware list contains only NVIDIA parts: Blackwell (B200, GB200, B300, GB300, DGX Spark), Hopper (H100, H200, GH200), Ada Lovelace (L20, L40/L40S) and Ampere (A100) ([Supported Hardware](#)). Clients see an OpenAI-compatible API, so switching costs sit in runtime configuration and quantized checkpoints.

The PyTorch-native architecture and what was removed

TensorRT-LLM 1.0 made "the PyTorch-based architecture ... stable and the default experience" ([Release Notes](#)). The development line has since removed the engine path: "PyTorch is now the sole execution backend" ([Migration Guide: TensorRT Backend Removed](#)). `trtllm-build`, `trtllm-refit`, `trtllm-prune` and per-model `convert_checkpoint.py` scripts are gone. `LLM(backend="tensorrt")` raises `ValueError`, and the `tensorrt` pip dependency is no longer installed (same source). Two-model speculative decoding was removed too; only one-model Eagle3/MTP paths remain ([Release Notes](#)).

When the removal happened matters for version pinning. The release-notes page lists the removal under its "Release 1.2" heading, but the tagged source disagrees. `v1.2.1`'s `setup.py` still registers `trtllm-build`, `trtllm-prune` and `trtllm-refit` as console scripts ([setup.py at v1.2.1](#)). Its `serve` command still accepts `--backend tensorrt` ([serve.py at v1.2.1](#)). We checked `setup.py` at each tag: `trtllm-build` is still present at `v1.3.0rc20` and absent from `v1.3.0rc21` (published 2026-07-15) onward ([setup.py at v1.3.0rc21](#)). Prebuilt engines and CI jobs calling `trtllm-build` are pinned to `v1.2.x`; moving to `1.3/1.4` means re-validating from Hugging Face or ModelOpt checkpoints.

Since 1.0, deprecated APIs get a 3-month migration period with runtime warnings before removal ([README](#), [Deprecation Policy](#)), so read the release notes on every bump.

Serving with trtllm-serve

Endpoints

`trtllm-serve` "starts an OpenAI compatible server" exposing `/v1/models`, `/v1/completions` and `/v1/chat/completions`, plus a Responses API at `/v1/responses`. It also serves `/health`, `/metrics` and `/version`, prototype `/start_profile` and `/stop_profile` endpoints, and render endpoints that are off unless `TRTLLM_ENABLE_RENDER_ENDPOINTS=1` ([trtllm-serve](#)).

A minimal pinned launch (image tag and run flags from NVIDIA's installation guide; the model is a local directory, as an air-gapped site needs):

```
docker run --rm --ipc host --gpus all \
  --ulimit memlock=-1 --ulimit stack=67108864 \
  -e TRTLLM_NO_USAGE_STATS=1 \
  -v /models:/models -p 8000:8000 \
  nvcr.io/nvidia/tensorrt-llm/release:1.4.0rc2 \
  trtllm-serve /models/Llama-3.1-8B-Instruct-FP8 \
  --host 0.0.0.0 --port 8000 \
  --served_model_name llama-3.1-8b-fp8 \
  --config /models/serve.yaml --no-telemetry
```

--host defaults to localhost and --port to 8000 in both v1.2.1 and v1.4.0rc2 ([serve.py at v1.2.1](#), [serve.py at v1.4.0rc2](#)). In a container, bind 0.0.0.0 deliberately; NVIDIA notes --ipc=host avoids "Bus error (core dumped)" ([Container Images](#)).

Flags that matter in production (1.4.0rc2 docs)

Flag	What it does	Notes
--tp_size, --pp_size, --ep_size, --cp_size	Tensor, pipeline, expert, context parallelism	--cluster_size is deprecated
--max_batch_size, --max_num_tokens, --max_seq_len	Scheduler capacity	LlmArgs defaults: 2048 and 8192; max_seq_len deduced from the model (llm_args.py at v1.4.0rc2)
--free_gpu_memory_fraction (alias --kv_cache_free_gpu_memory_fraction)	Share of free memory given to KV cache after weights and buffers	Default 0.9
--kv_cache_dtype auto fp8 nvfp4	Override KV dtype	Prototype; not present in v1.2.1
--config (alias --extra_llm_api_options)	YAML file mapped onto LlmArgs	See precedence below
--set PATH=VALUE	Override one LlmArgs field	Prototype; new in the 1.3/1.4 line
--served_model_name	Name exposed in the API	Not present in v1.2.1
--tool_parser, --reasoning_parser	Tool-call and reasoning extraction	Must match the checkpoint's chat template
--enable_chunked_prefill, --enable_attention_dp	Scheduling features	Chunked prefill is off by default in LlmArgs
--otlp_traces_endpoint	OpenTelemetry traces target	Prototype
--telemetry/--no-telemetry	Usage telemetry	Defaults to on
--backend	Execution backend	Deprecated; only pytorch remains

Sources: [trtllm-serve CLI reference](#); v1.2.1 comparison from [serve.py at v1.2.1](#); defaults from [llm_args.py at v1.4.0rc2](#). Prototype-tagged flags can change between release candidates.

Configuration precedence changed

This is the most common migration trap. Current docs: "Precedence is fixed, regardless of command-line order: LlmArgs defaults < --config < dedicated CLI flags < --set" ([trtllm-serve](#)). Previously "YAML overrode CLI" ([Release Notes](#)); v1.2.1's --config help still reads "Path to a YAML file that overwrites the parameters specified by trtllm-serve" ([serve.py at v1.2.1](#)). A script with --max_batch_size 64 and max_batch_size: 256 in YAML runs at 256 on 1.2.1 and 64 on 1.4. Keep each setting in one place and diff the effective config after every upgrade.

Do not pass secrets through --set (they can appear in process listings and logs). JSON schemas for validating YAML ship in release containers at /app/tensorrt_llm/schemas/ (same source).

No built-in authentication on the API

The CLI reference lists no API-key or TLS option for the OpenAI endpoints; examples pass a placeholder key ([trtllm-serve](#)). In the 1.4.0rc2 source, API keys exist only for optional control endpoints: --enable_sleep_mode requires TRTLLM_RUNTIME_CONTROL_API_KEY ([serve.py at v1.4.0rc2](#), [openai_server.py at v1.4.0rc2](#)). Put an authenticating, TLS-terminating gateway in front of every replica and keep /metrics, profiling and kv_cache_events on the cluster network. vLLM has --api-key, but its docs warn it covers only some path prefixes and should not be relied on alone ([vLLM serve CLI](#)).

Telemetry: the first thing to switch off on a private cluster

From the 1.3 release-candidate line onward, the README states that "TensorRT-LLM collects anonymous telemetry data by default" ([README](#), [Telemetry Data Collection](#)). The v1.2.1 README has no such section. NVIDIA says the payload excludes prompts, outputs, weights and model paths but includes GPU SKU and count, CUDA version, parallelism, quantization, KV dtype and selected configuration values; the telemetry page lists 334 captured TorchLlmArgs fields ([Telemetry](#)). It is auto-disabled only in CI and test environments.

The documented opt-outs are: TRTLLM_NO_USAGE_STATS=1, DO_NOT_TRACK=1 or TELEMETRY_DISABLED=true; the file ~/.config/trtllm/do_not_track; TelemetryConfig(disabled=True) in Python; or --no-telemetry on trtllm-serve, trtllm-bench and trtllm-eval ([README](#)). On a regulated or air-gapped cluster, set the variable in the image, pass the flag too, and confirm with egress logs. Security review should record the on-by-default behavior.

Hardware and precision support matrix

The quantization page lists ten recipes: FP4, FP8 per-tensor, FP8 block scaling, FP8 rowwise, FP8 KV cache, NVFP4 KV cache, and W4A16/W4A8 in both GPTQ and AWQ form. It states that "the default PyTorch backend supports FP4 and FP8 quantization on the latest Blackwell and Hopper GPUs" ([Quantization](#)). Its hardware matrix, transcribed exactly ("Y" = supported, "." = not listed):

Architecture	NVF P4	MXF P4	FP8 per-tensor	FP8 block	FP8 rowwise	FP8 KV	NVFP 4 KV	W4A8 AWQ	W4A16 AWQ	W4A8 GPTQ	W4A16 GPTQ
Blackwell (sm120)	Y	Y	Y	.	.	Y	.	.	.	.	.
Blackwell (sm100/103)	Y	Y	Y	Y	.	Y	Y	Y	Y	Y	Y
Rubin (sm107)	Y	Y	Y	Y	.	Y	.	Y	Y	Y	Y
Hopper	.	.	Y	Y	Y	Y	.	Y	Y	Y	Y
Ada Lovelace	.	.	Y	.	.	Y	.	Y	Y	Y	Y
Ampere	.	.	.	.	.	Y	.	.	Y	.	Y

Source: [Quantization, Hardware Support Matrix](#), as of 2026-10-28.

- **NVFP4 and MXFP4 weights need Blackwell (or Rubin) in this matrix.** Hopper is "." for both. Model guides document narrower fallbacks (for example MXFP4 GPT-OSS on Hopper), tracked in our [NVFP4 vs MXFP4 vs FP8 runtime compatibility](#) article.
- **FP8 block scaling is not the same recipe across generations.** NVIDIA notes that on sm100/103/107 the FP8 block-wise GEMMs use the MXFP8 recipe (E4M3 values with UE8M0 scales). SM90 uses E4M3 with FP32 scales ([Quantization](#)). Re-run accuracy gates when moving a block-scaled FP8 model from Hopper to Blackwell.
- **Blackwell sm120 parts get a much smaller menu** than data-center sm100/103 parts: no block or rowwise FP8 and no AWQ/GPTQ. Check your GPU's compute capability (`nvidia-smi --query-gpu=compute_cap --format=csv`) before choosing a checkpoint.
- **Ampere (A100) is limited to W4A16 AWQ/GPTQ and FP8 KV cache** in the matrix. On A100s the precision argument for TensorRT-LLM is weak.
- **The model matrix also gates you.** NVFP4 KV cache, for example, is marked only for LLaMA-v2, LLaMA 3 and Qwen-3 among the listed families (same page).

Quantization recipes and when to pick each

TensorRT-LLM runs pre-quantized checkpoints produced by NVIDIA Model Optimizer (ModelOpt). Where none exists, you quantize offline with ModelOpt's `huggingface_example.sh --quant fp8` (or other `--quant` values) ([Quantization](#)). The checkpoint metadata now carries the deployment decision. NVIDIA's Llama-3.1-8B-Instruct-FP8 checkpoint, for example, declares "quant_algo": "FP8" and "kv_cache_quant_algo": "FP8", with `lm_head` excluded ([hf_quant_config.json](#), [nvidia/Llama-3.1-8B-Instruct-FP8](#)). With the default auto KV dtype, it gets an FP8 KV cache without anyone asking.

A pragmatic selection order for private fleets:

Situation	Start with	Why	Gate
Hopper, quality-sensitive	FP8 per-tensor	Broadest Hopper support in both matrices	Golden-prompt diff vs BF16
Hopper, DeepSeek-style models shipped block-scaled	FP8 block scaling	Matches the published checkpoint	Re-gate if moved to Blackwell (MXFP8 recipe)
Blackwell sm100/103, memory-bound	NVFP4	Native FP4 path per the overview	Task-level eval, not perplexity only
Ada or Ampere, memory-bound	W4A16 AWQ or GPTQ	Only 4-bit recipes listed for Ampere	Check the model is "Y" in the model matrix
Long contexts, many sessions	FP8 KV cache	Halves KV bytes vs 16-bit	Long-context retrieval eval

NVIDIA's overview claims FP8 "can double performance and halve memory consumption compared to 16-bit floating point, with minimal impact on model accuracy" ([Overview](#)). Treat that as a vendor claim to test on your own models, not a planning number. For the cost side of precision choices, see [FP8 and INT4 inference costs](#).

NVFP4 KV cache has extra constraints. It requires ModelOpt offline quantization, and currently TensorRT-LLM "only supports FP8 weight/activation quantization when NVFP4 KV cache is enabled" ([Quantization](#)).

KV cache configuration

Defaults

Setting	Default	Source
free_gpu_memory_fraction	0.9	KV Cache System
max_tokens	unset; if set, the lesser of the two limits wins	same
enable_block_reuse (cross-request prefix reuse)	true	same
enable_partial_reuse	true	same
host_cache_size (host offload)	0 (no offload)	same
tokens_per_block	32	llm_args.py at v1.4.0rc2
dtype	auto, inferred from model config	KV Cache System

The fraction applies to memory that is *free after weights and buffers are allocated*, not to total HBM ([trtllm-serve](#)). Eviction is prioritized LRU with priorities 0 to 100 (default 35). With host offload enabled, evicted blocks move to host memory and remain reusable until evicted from there ([KV Cache System](#)). Two KV cache managers exist; `use_kv_cache_manager_v2`: `auto` lets the model choose (same source).

Worked example: KV capacity for an 8B FP8 model on one 80 GB GPU

Model facts, from the checkpoint config: 32 layers, 8 KV heads, head dimension 128 ([config.json](#), [nvidia/Llama-3.1-8B-Instruct-FP8](#)). Its quant config selects FP8 KV (1 byte per element).

1. KV bytes per token = 2 (K and V) x 32 layers x 8 heads x 128 dims x 1 byte = 65,536 bytes = 64 KiB.
With a BF16 KV cache it would be 128 KiB.
2. **Assumption:** after weights, activations and runtime buffers, the startup profiler finds 60 GiB free. This is illustrative; read the real figure from your startup log.
3. KV pool = 0.9 x 60 GiB = 54 GiB = 56,623,104 KiB.
4. Token capacity = 56,623,104 / 64 = 884,736 tokens, which is 27,648 blocks of 32 tokens.
5. At 8,192 tokens of context per request, that is 884,736 / 8,192 = 108 fully grown sequences resident at once. With BF16 KV it would be 54.

The 6 GiB the 0.9 default leaves is headroom; raise the fraction only after a peak-concurrency soak test. NVIDIA's own multi-node DeepSeek example uses 0.95 ([trtllm-serve](#)). The same arithmetic, applied to vLLM's allocator, is in our [vLLM KV cache capacity planner](#).

Multi-tenant caution

Block reuse is on by default and matches prefixes across requests ([KV Cache System](#)). Shared system prompts therefore reuse blocks across tenants. If tenants on one replica must not share cache state, separate them by replica or disable reuse (`kv_cache_config.enable_block_reuse: false`), and measure the throughput cost. Multimodal models currently require block reuse to be disabled anyway ([trtllm-serve](#)).

In-flight batching, overlap scheduler and feature combinations

In-flight (continuous) batching processes context-phase and generation-phase sequences together in one packed batch. `max_num_tokens` caps the batched tokens per iteration and defaults to 8192. NVIDIA recommends tuning it: high enough to saturate GPU math, low enough to hold TTFT and TPOT targets ([Paged Attention, IFB, and Request Scheduling](#)). Chunked prefill splits long contexts so they can batch with decode work. Each chunk except the last must be a multiple of the KV block size (same source).

The overlap scheduler is enabled by default. It launches GPU work for step $n+1$ while the CPU finishes step n . NVIDIA states the tradeoff: it "introduces one extra decoding step but significantly improves throughput". Disable it with `disable_overlap_scheduler: true` ([Overlap Scheduler](#)).

Not every feature combines. Pairs from the [Feature Combination Matrix](#) worth knowing before you design a config (as of 2026-10-28):

Combination	Status
Overlap scheduler + CUDA graph + TP/PP/EP + chunked prefill + KV cache reuse	Yes
Linear speculative decoding + pipeline parallelism	No
Speculative decoding (linear, dynamic trees or legacy NGram) + Helix parallelism	No

Combination	Status
Logits post-processor + disaggregated serving	No
Logits post-processor + any speculative decoding	No
Chunked prefill + pipeline parallelism	Untested
LoRA + expert parallelism, attention DP, disaggregated serving, speculative decoding or guided decoding	Untested
Helix parallelism + attention DP	Known issues

Treat "Untested" as a pilot item. Model-specific limits sit on top; DeepSeek-V4.1's CSA2 attention, for example, does not support PP or CP (same source).

Observability and operations

There are two metrics surfaces, and they behave differently:

- `/metrics` returns JSON iteration statistics: GPU memory, KV block counts, cache hit rate and active requests. It is populated only when `enable_iter_perf_stats: true`, which "can slightly impact performance". Statistics are removed from an internal queue once read and are returned to only one caller. NVIDIA advises polling "from a single client" ([trtllm-serve](#), [Metrics Endpoint](#)).
- `/prometheus/metrics` serves Prometheus exposition with the `trtllm_` prefix: request success counts, end-to-end latency, TTFT, queue time and KV cache hit rate, reuse and utilization ([Prometheus Metrics example](#)). In the 1.4.0rc2 server, this route is mounted only when `return_perf_metrics` is true ([openai_server.py at v1.4.0rc2](#)), and that field defaults to false ([llm_args.py at v1.4.0rc2](#)). Histogram buckets are configurable through `PrometheusMetricsConfig` ([PrometheusMetricsConfig](#)).

A baseline YAML for a private replica, using only fields documented on the pages cited above:

```
# serve.yaml (validate against /app/tensorrt_llm/schemas/ for your version)
max_batch_size: 256
max_num_tokens: 8192
enable_chunked_prefill: true
return_perf_metrics: true      # mounts /prometheus/metrics
enable_iter_perf_stats: false  # turn on only for profiling windows
kv_cache_config:
  free_gpu_memory_fraction: 0.9
  enable_block_reuse: true
  dtype: auto                  # follows the checkpoint's KV quant config
```

`max_batch_size: 256` is an example value, not a recommendation; tune it under load. For version control, record the container tag *and* `TRT_LLM_GIT_COMMIT`, which the installation guide says is recorded in the image ([Installation Guide](#)). `/version` gives the runtime view. NGC tags include both regular and `.dev` nightly builds, and NVIDIA says nightly quality "is not guaranteed" (same source). Never let a floating tag reach production.

Upgrade gates: read [BREAKING CHANGE] and [DEPRECATION] notes, diff the effective config, validate YAML against the new schema, re-run accuracy and latency gates, re-check the telemetry opt-out.

TensorRT-LLM vs vLLM vs SGLang

No speedup figures here; published benchmarks rarely match a private workload. The comparison is structural:

Criterion	TensorRT-LLM	vLLM	SGLang
Latest release (2026-10-28)	v1.2.1 stable; v1.4.0rc2 pre-release	v0.32.0 (2026-10-18)	v0.5.22 (2026-10-19)
Hardware	NVIDIA only	NVIDIA, AMD ROCm, Intel XPU, CPUs and plugins (vLLM install docs)	NVIDIA, AMD Instinct, Google TPU and others (SGLang README)
API auth built in	None for OpenAI routes	--api-key with documented gaps	--api-key, plus --admin-api-key for management endpoints
Telemetry default	On (1.3+ line)	Anonymous usage stats on; off with VLLM_NO_USAGE_STATS=1 or DO_NOT_TRACK=1	No usage reporting in v0.5.22 source; tracing off unless --enable-trace
Strongest fit	NVIDIA-only fleets using ModelOpt NVFP4/FP8 checkpoints and NVIDIA model recipes	Mixed or changing hardware	Mixed hardware, including TPU

Choose TensorRT-LLM when the fleet is NVIDIA-only for good, target models have NVIDIA deployment guides and ModelOpt checkpoints, and the team can absorb a fast release cadence. Choose vLLM or SGLang when hardware diversity or recent features in a non-pre-release build matter more.

A fair benchmark protocol:

1. Same checkpoint precision, GPU and TP size on both engines.
2. Replay recorded production prompts, including prefix sharing.
3. Sweep concurrency; report throughput at fixed p95 TTFT and TPOT targets.
4. Golden-prompt accuracy diff against a BF16 reference on both.
5. Measure after warmup.

Adoption checklist and rollback

Before the pilot

- [] Choose the line: v1.2.1 (stable, still has trtllm-build, older precedence rules) or a pinned 1.4 release candidate (current features, no engine path). Write the reason down.
- [] Confirm every target model and precision is "Y" in both the model and hardware matrices for your exact GPU architecture.
- [] Mirror the container image and checkpoints internally; point MODEL at local paths.

- [] Disable telemetry (environment variable plus `--no-telemetry`) and verify with egress logs.
- [] Put an authenticating gateway in front; restrict `/metrics`, profiling and control endpoints to the cluster network.

During the pilot

- [] Record startup KV pool size and token capacity; compare with your own arithmetic.
- [] Golden-prompt accuracy gate vs BF16 for each quantized checkpoint.
- [] Load test at target concurrency with the overlap scheduler, chunked prefill and block reuse in their intended states.
- [] Check every feature pair you enable against the combination matrix.

Rollback

- [] Keep the previous engine (vLLM or the prior TensorRT-LLM tag) deployed on standby with the same served model name, so the gateway can switch upstreams without client changes.
- [] Keep BF16 or the previous quantized checkpoint available; a precision change and an engine change should never ship in the same release.
- [] Define the rollback trigger in advance: accuracy-gate failure, p95 TTFT regression beyond your SLO, or KV-capacity shortfall at peak.

GPU Smith helps teams plan and validate private inference stacks on NVIDIA hardware; [contact us](#) if you are running this evaluation.

Sources

1. Overview, TensorRT LLM documentation, NVIDIA, page updated 2026-10-26.
<https://nvidia.github.io/TensorRT-LLM/overview.html>
2. trtllm-serve CLI reference, NVIDIA, page updated 2026-10-26. <https://nvidia.github.io/TensorRT-LLM/commands/trtllm-serve/trtllm-serve.html>
3. Migration Guide: TensorRT Backend Removed, NVIDIA. <https://nvidia.github.io/TensorRT-LLM/legacy/tensorrt-backend-removal.html>
4. Release Notes, TensorRT LLM, NVIDIA. <https://nvidia.github.io/TensorRT-LLM/release-notes.html>
5. NVIDIA/TensorRT-LLM releases, GitHub (v1.2.1 2026-04-20; v1.4.0rc2 2026-10-28).
<https://github.com/NVIDIA/TensorRT-LLM/releases>
6. Release v1.2.1, NVIDIA/TensorRT-LLM, GitHub, 2026-04-20. <https://github.com/NVIDIA/TensorRT-LLM/releases/tag/v1.2.1>
7. Release v1.4.0rc2, NVIDIA/TensorRT-LLM, GitHub, 2026-10-28. <https://github.com/NVIDIA/TensorRT-LLM/releases/tag/v1.4.0rc2>
8. README (Deprecation Policy, Telemetry Data Collection), NVIDIA/TensorRT-LLM, GitHub.
<https://github.com/NVIDIA/TensorRT-LLM/blob/main/README.md>
9. LICENSE, NVIDIA/TensorRT-LLM, GitHub. <https://github.com/NVIDIA/TensorRT-LLM/blob/main/LICENSE>

10. setup.py at v1.2.1, NVIDIA/TensorRT-LLM, GitHub. <https://github.com/NVIDIA/TensorRT-LLM/blob/v1.2.1/setup.py>
11. setup.py at v1.3.0rc21, NVIDIA/TensorRT-LLM, GitHub, 2026-07-15. <https://github.com/NVIDIA/TensorRT-LLM/blob/v1.3.0rc21/setup.py>
12. tensorrt_llm/commands/serve.py at v1.2.1, NVIDIA/TensorRT-LLM, GitHub. https://github.com/NVIDIA/TensorRT-LLM/blob/v1.2.1/tensorrt_llm/commands/serve.py
13. tensorrt_llm/commands/serve.py at v1.4.0rc2, NVIDIA/TensorRT-LLM, GitHub. https://github.com/NVIDIA/TensorRT-LLM/blob/v1.4.0rc2/tensorrt_llm/commands/serve.py
14. tensorrt_llm/serve/openai_server.py at v1.4.0rc2, NVIDIA/TensorRT-LLM, GitHub. https://github.com/NVIDIA/TensorRT-LLM/blob/v1.4.0rc2/tensorrt_llm/serve/openai_server.py
15. tensorrt_llm/llmapi/llm_args.py at v1.4.0rc2, NVIDIA/TensorRT-LLM, GitHub. https://github.com/NVIDIA/TensorRT-LLM/blob/v1.4.0rc2/tensorrt_llm/llmapi/llm_args.py
16. Telemetry, TensorRT LLM developer guide, NVIDIA. <https://nvidia.github.io/TensorRT-LLM/developer-guide/telemetry.html>
17. Supported Hardware, TensorRT LLM, NVIDIA. <https://nvidia.github.io/TensorRT-LLM/supported-hardware.html>
18. Installation Guide, TensorRT LLM, NVIDIA. <https://nvidia.github.io/TensorRT-LLM/installation/installation-guide.html>
19. Container Images, TensorRT LLM, NVIDIA. <https://nvidia.github.io/TensorRT-LLM/installation/containers.html>
20. Quantization, TensorRT LLM, NVIDIA. <https://nvidia.github.io/TensorRT-LLM/features/quantization.html>
21. KV Cache System, TensorRT LLM, NVIDIA. <https://nvidia.github.io/TensorRT-LLM/features/kvcache.html>
22. Paged Attention, IFB, and Request Scheduling, TensorRT LLM, NVIDIA. <https://nvidia.github.io/TensorRT-LLM/features/paged-attention-ifb-scheduler.html>
23. Overlap Scheduler, TensorRT LLM, NVIDIA. <https://nvidia.github.io/TensorRT-LLM/features/overlap-scheduler.html>
24. Feature Combination Matrix, TensorRT LLM, NVIDIA. <https://nvidia.github.io/TensorRT-LLM/features/feature-combination-matrix.html>
25. Prometheus Metrics example, TensorRT LLM, NVIDIA. https://nvidia.github.io/TensorRT-LLM/examples/prometheus_metrics.html
26. PrometheusMetricsConfig API reference, TensorRT LLM, NVIDIA. <https://nvidia.github.io/TensorRT-LLM/llm-api/reference/PrometheusMetricsConfig.html>
27. nvidia/Llama-3.1-8B-Instruct-FP8 model card, config.json and hf_quant_config.json, Hugging Face. <https://huggingface.co/nvidia/Llama-3.1-8B-Instruct-FP8>
28. vllm serve CLI reference (--api-key), vLLM. <https://docs.vllm.ai/en/latest/cli/serve/>
29. Installation (supported hardware), vLLM. https://docs.vllm.ai/en/latest/getting_started/installation/
30. Release v0.32.0, vllm-project/vllm, GitHub, 2026-10-18. <https://github.com/vllm-project/vllm/releases/tag/v0.32.0>
31. SGLang README (Supported Hardware), sgl-project/sglang, GitHub. <https://github.com/sgl-project/sglang>

32. Release v0.5.22, sgl-project/sglang, GitHub, 2026-10-19. <https://github.com/sgl-project/sglang/releases/tag/v0.5.22>; serving and observability args at the tag: https://github.com/sgl-project/sglang/blob/v0.5.22/python/sglang/srt/arg_groups/fields/serving.py
33. tensorrt_llm/version.py on main (1.4.0rc3), NVIDIA/TensorRT-LLM, GitHub, checked 2026-10-28. https://github.com/NVIDIA/TensorRT-LLM/blob/main/tensorrt_llm/version.py
34. vllm/usage/usage_lib.py at v0.32.0, vllm-project/vllm, GitHub. https://github.com/vllm-project/vllm/blob/v0.32.0/vllm/usage/usage_lib.py
35. Triton Inference Server Release 26.08 notes, NVIDIA, updated 2026-10-01. <https://docs.nvidia.com/deeplearning/triton-inference-server/release-notes/rel-26-08.html>

Source links

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

1. <https://nvidia.github.io/TensorRT-LLM/overview.html>
2. <https://github.com/NVIDIA/TensorRT-LLM/blob/main/LICENSE>
3. <https://github.com/NVIDIA/TensorRT-LLM/releases/tag/v1.2.1>
4. <https://github.com/NVIDIA/TensorRT-LLM/releases>
5. <https://github.com/NVIDIA/TensorRT-LLM/releases/tag/v1.4.0rc2>
6. https://github.com/NVIDIA/TensorRT-LLM/blob/main/tensorrt_llm/version.py
7. <https://nvidia.github.io/TensorRT-LLM/installation/installation-guide.html>
8. <https://docs.nvidia.com/deeplearning/triton-inference-server/release-notes/rel-26-08.html>
9. <https://nvidia.github.io/TensorRT-LLM/supported-hardware.html>
10. <https://nvidia.github.io/TensorRT-LLM/release-notes.html>
11. <https://nvidia.github.io/TensorRT-LLM/legacy/tensorrt-backend-removal.html>
12. <https://github.com/NVIDIA/TensorRT-LLM/blob/v1.2.1/setup.py>
13. https://github.com/NVIDIA/TensorRT-LLM/blob/v1.2.1/tensorrt_llm/commands/serve.py
14. <https://github.com/NVIDIA/TensorRT-LLM/blob/v1.3.0rc21/setup.py>
15. <https://github.com/NVIDIA/TensorRT-LLM/blob/main/README.md>
16. <https://nvidia.github.io/TensorRT-LLM/commands/trtllm-serve/trtllm-serve.html>
17. https://github.com/NVIDIA/TensorRT-LLM/blob/v1.4.0rc2/tensorrt_llm/commands/serve.py
18. <https://nvidia.github.io/TensorRT-LLM/installation/containers.html>
19. https://github.com/NVIDIA/TensorRT-LLM/blob/v1.4.0rc2/tensorrt_llm/llmapi/llm_args.py
20. https://github.com/NVIDIA/TensorRT-LLM/blob/v1.4.0rc2/tensorrt_llm/serve/openai_server.py
21. <https://docs.vllm.ai/en/latest/cli/serve>
22. <https://nvidia.github.io/TensorRT-LLM/developer-guide/telemetry.html>
23. <https://nvidia.github.io/TensorRT-LLM/features/quantization.html>
24. <https://huggingface.co/nvidia/Llama-3.1-8B-Instruct-FP8>
25. <https://nvidia.github.io/TensorRT-LLM/features/kvcache.html>

- 26. <https://nvidia.github.io/TensorRT-LLM/features/paged-attention-ibf-scheduler.html>
- 27. <https://nvidia.github.io/TensorRT-LLM/features/overlap-scheduler.html>
- 28. <https://nvidia.github.io/TensorRT-LLM/features/feature-combination-matrix.html>
- 29. https://nvidia.github.io/TensorRT-LLM/examples/prometheus_metrics.html
- 30. <https://nvidia.github.io/TensorRT-LLM/llm-api/reference/PrometheusMetricsConfig.html>
- 31. <https://github.com/vllm-project/vllm/releases/tag/v0.32.0>
- 32. <https://github.com/sgl-project/sglang/releases/tag/v0.5.22>
- 33. https://docs.vllm.ai/en/latest/getting_started/installation
- 34. <https://github.com/sgl-project/sglang>
- 35. https://github.com/sgl-project/sglang/blob/v0.5.22/python/sglang/srt/arg_groups/fields/serving.py
- 36. https://github.com/vllm-project/vllm/blob/v0.32.0/vllm/usage/usage_lib.py

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.