



vLLM vs SGLang vs TensorRT-LLM: Picking an Inference Server

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 · vllm vs sglang vs tensorrt-llm · sglang vs vllm · tensorrt-llm vs vllm · best llm inference server self-hosted · vllm vs sglang structured output · which inference engine for h100 b200

Original article: <https://gpusmith.com/articles/en/vllm-vs-sglang-vs-tensorrt-llm-selection>



In this article

1. What you are actually choosing
2. Architecture in one page each
3. Hardware support
4. Model coverage
5. Quantization and numeric formats
6. Parallelism, multi-node and disaggregation
7. Structured output and agentic workloads
8. Operations burden
9. Licensing and support
10. A benchmark method you can reproduce
11. Decision matrix
12. FAQ
13. Sources

What you are actually choosing

vLLM, SGLang and TensorRT-LLM are inference engines: the layer that owns the KV cache, schedules batches, runs attention and GEMM kernels and exposes an HTTP API. They are not the whole serving stack. Routers, autoscalers, multi-model gateways and enterprise packaging sit above them. NVIDIA's own documentation says the TensorRT-LLM LLM API "integrates seamlessly with the broader inference ecosystem, including NVIDIA Dynamo and the Triton Inference Server" ([TensorRT-LLM overview](#)), and commercial products wrap the open-source engines: Red Hat AI Inference 3.5.0 (GA 2026-08-27) "packages the upstream vLLM v0.24.0 release" ([Red Hat release notes](#); [life cycle](#)). The NVIDIA NIM LLM 2.0.0 support matrix lists deployment profiles named `vllm-bf16-tp1-pp1` and `vllm-fp8-tp1-pp1` ([NIM support matrix](#)). The page does not name the engine; the `vllm-` prefix on every listed profile is the only first-party indication.

So the question is which engine you will qualify, upgrade and debug on your own hardware for the next 12 to 24 months. This guide uses documented features only, with no speed rankings: those are specific to a model, precision, GPU, engine version and traffic shape, and go stale within a few releases.

Versions as of 2026-10-28

Engine	Latest stable tag	Date	Notes
vLLM	v0.32.0	2026-10-18	v0.29.0 shipped 2026-09-09 (releases)
SGLang	v0.5.22	2026-10-19	v0.5.20 shipped 2026-09-18 (releases)
TensorRT-LLM	v1.2.1	2026-04-20	Newest tag is pre-release v1.4.0rc2 , 2026-10-28

All three projects host "latest" documentation built from their main branches. The vLLM structured-outputs page already announces a deprecation "starting in v0.33.0", one minor version past the newest tag ([vLLM structured outputs](#)). The TensorRT-LLM pages cited here were generated from commit b96ba43, last updated October 26, 2026. Read every feature claim below as "documented on main as of 2026-10-28", and check it against the tag you plan to deploy. We spot-checked the tags: SGLang's eviction policies are in v0.5.22, vLLM's disaggregation page is unchanged in v0.32.0, and TensorRT-LLM v1.2.1 already labels disaggregation "Beta"; the vLLM v0.33.0 deprecation notice is main-only.

Architecture in one page each

vLLM

vLLM started from the PagedAttention paper (SOSP 2023). The paper manages the KV cache in fixed-size blocks, like virtual-memory pages, so memory is allocated on demand rather than reserved per request at maximum length. The authors reported that vLLM "improves the throughput of popular LLMs by 2-4× with the same level of latency" compared with FasterTransformer and Orca ([Kwon et al., arXiv 2309.06180](#)). That is a 2023 result, reported by the authors against 2023 baselines. It is not a current comparison with SGLang or TensorRT-LLM, both of which now use paged KV caches too.

The current docs list continuous batching, chunked prefill, prefix caching, piecewise and full CUDA/HIP graphs, speculative decoding (n-gram, suffix, EAGLE, DFlash), torch.compile-based kernel generation, and "Disaggregated prefill, decode, and encode" ([vLLM docs](#)). Prefix caching is hash-based: each KV block is keyed by its parent's hash plus its tokens, and blocks are evicted in LRU order ([vLLM prefix caching design](#)).

SGLang

SGLang came out of a 2023 paper on executing structured LLM programs. The runtime contribution is RadixAttention, which stores KV for previously seen prefixes in a radix tree so any request that shares a prefix can reuse it, together with "compressed finite state machines for faster structured output decoding". The authors reported "up to 6.4x higher throughput compared to state-of-the-art inference systems" on their own workloads ([Zheng et al., arXiv 2312.07104](#), v1 December 2023, revised June 2024). As with the vLLM paper, treat this as dated and author-reported.

Today's docs describe SGLang as "designed for low-latency, high-throughput inference with RadixAttention, prefix caching, and multi-GPU parallelism" ([SGLang docs](#)). The radix cache has selectable eviction policies: lru (the default), lfu, slru, priority, and tlru. The docs recommend tlru for "Agentic / multi-turn workloads where tail TTFT matters more than raw hit rate" ([SGLang radix eviction policies](#)).

TensorRT-LLM

TensorRT-LLM used to be known for compiling a model into a TensorRT engine before serving it. That is no longer how it works. v1.0.0 (2025-09-24) made PyTorch the default backend in trtllm-serve ([v1.0.0 notes](#)). v1.3.0rc20 (2026-06-30) was announced as "the last one supporting the TensorRT backend" ([v1.3.0rc20 notes](#)). The current migration page says "PyTorch is now the sole execution backend" and "There is no

separate checkpoint-conversion or engine-build step" ([TensorRT backend removal](#)). The overview lists in-flight batching with paged attention, chunked prefill, multi-GPU multi-node inference, speculative decoding (EAGLE, MTP, NGram), LoRA, guided decoding and "Disaggregated Serving (Beta)" ([TensorRT-LLM overview](#)).

There is a catch. As of 2026-10-28, every release since that change is a release candidate. The newest stable tag, v1.2.1, predates the removal. Anyone who wants the architecture the docs describe has to run an RC.

Hardware support

This criterion settles more selections than any other.

	vLLM	SGLang	TensorRT-LLM
NVIDIA GPUs	Yes	Yes	Yes (only vendor)
AMD GPUs	Yes (ROCm)	Yes	No
Intel	XPU; x86 CPUs	XPU; Xeon CPUs with AMX (4th gen+)	No
Google TPU	Hardware plugin	SGLang-TorchTPU (TPU7x), SGL-JAX (v6e, v7)	No
Others	Plugins: Gaudi, Spyre, Ascend, more; ARM/PowerPC CPUs	Ascend NPUs, Moore Threads GPUs, Jetson Orin	No

Sources: [vLLM docs](#); [vLLM installation](#) (plugins "live outside the main vllm repository"); [SGLang hardware platforms](#), [SGLang CPU servers](#), [SGLang TPU](#); [TensorRT-LLM supported hardware](#).

TensorRT-LLM's documented GPU list is Blackwell (B200, GB200, B300, GB300, DGX Spark), Hopper (H100, H200, GH200), Ada Lovelace (L20, L40/L40S) and Ampere (A100) ([supported hardware](#)). If the fleet will include AMD Instinct parts now or after the next procurement cycle, TensorRT-LLM cannot be the only engine.

A plugin entry is a weaker commitment than in-tree support: SGLang's TPU page says each TPU backend "maintains its own runtime, model coverage, and deployment configurations." Qualify the plugin build, not the upstream version.

Model coverage

vLLM and SGLang both fall back to Hugging Face Transformers implementations for architectures they have not ported. vLLM says that "Models do not *need* to be natively supported to be used in vLLM". It also says Transformers-backend performance "should be identical" to a native implementation, which is the project's claim, not a measurement ([vLLM supported models](#)). SGLang's fallback "works for most decoder-style language models" and is selected with `--model-impl transformers` ([SGLang Transformers fallback](#)).

TensorRT-LLM publishes a table of supported architectures for its PyTorch backend, about 50 class names on the page as of 2026-10-26, and the page describes no fallback for unlisted architectures ([TensorRT-LLM supported models](#)). In practice, if you need to run a new open-weight model in the week it is released, check

the TensorRT-LLM table first. With vLLM and SGLang the fallback path gives you a slower but working route.

Quantization and numeric formats

Format	vLLM	SGLang	TensorRT-LLM
FP8	Yes	Yes (NVIDIA, AMD)	Yes (Hopper+; Ada per-tensor)
NVFP4	Yes	modelopt_fp4 (native FP4 on SM100+, Marlin W4A16 on SM80-SM90); petit_nvfp4 on AMD	Yes (Blackwell)
MXFP4	Yes	Yes (NVIDIA; AMD CDNA3/CDNA4)	Yes (Blackwell)
AWQ / GPTQ	Yes	AWQ yes; GPTQ via gptq_marlin on GPUs	W4A16 and W4A8 AWQ/GPTQ
GGUF	Yes	NVIDIA and Ascend, not AMD	Not listed

vLLM lists "FP8, MXFP8/MXFP4, NVFP4, INT8, INT4, GPTQ/AWQ, GGUF, compressed-tensors, ModelOpt, TorchAO, and more" ([vLLM docs](#)). SGLang publishes a per-platform compatibility table covering NVIDIA, AMD MI300X/MI325X/MI350X and Ascend NPUs. On NVIDIA it lists modelopt_fp4 with "SM80-SM90 via Marlin; SM100+ native FP4". Plain gptq is marked "Removed on NVIDIA and AMD GPUs — use gptq_marlin instead" ([SGLang quantization](#)). TensorRT-LLM's matrix gives Blackwell sm100/103 the widest coverage, including NVFP4 and MXFP4 weights and an NVFP4 KV cache. Hopper gets FP8 variants and W4 recipes but no FP4. Ampere gets only FP8 KV cache and W4A16. The expected path is a pre-quantized NVIDIA Model Optimizer checkpoint ([TensorRT-LLM quantization](#)).

A format name on a list does not make two checkpoints interchangeable. Scaling granularity, KV-cache dtype and kernel availability differ by GPU generation. That interaction is covered in [NVFP4 vs MXFP4 vs FP8 runtime compatibility](#).

Parallelism, multi-node and disaggregation

All three engines document multi-GPU and multi-node serving. vLLM lists "Tensor, pipeline, data, expert, and context parallelism" ([vLLM docs](#)). TensorRT-LLM lists multi-GPU multi-node inference with tensor, pipeline and expert parallelism ([overview](#)). SGLang documents expert parallelism, data-parallel attention, prefill and decode context parallelism, pipeline parallelism for long context, and multi-node and Kubernetes deployment guides ([SGLang documentation index](#)).

Prefill/decode disaggregation is where the maturity labels diverge, and the labels matter for change control:

- **vLLM:** "This feature is experimental and subject to change." The page also says plainly that "Disaggregated prefill DOES NOT improve throughput". Its stated goals are separate TTFT and ITL tuning and control of tail ITL. Documented KV connectors include NixlConnector, Mooncake, LMCache and a ROCm-only MoRHIConnector ([vLLM disaggregated prefill](#)).

- **SGLang:** PD disaggregation is documented without an experimental label. "Currently, we support Mooncake and NIXL as the transfer engine." Newer options carry long compatibility lists. For example, the `prefill_complete` decode allocation policy requires Mooncake and is "not a general throughput improvement" ([SGLang PD disaggregation](#)).
- **TensorRT-LLM:** The overview labels it "Beta"; the feature page itself carries no label. It shows the setup (`trtllm-serve disaggregated -c disagg_config.yaml`, NIXL over RDMA or NVLink, with UCX or LIBFABRIC as the NIXL transport) and states that "only decoder-only models are supported" ([TensorRT-LLM disaggregated serving](#)).

If you need disaggregation, what matters most is whether your orchestration layer supports that engine's KV transfer path.

Structured output and agentic workloads

Constrained decoding

	Backends	Constraint types
vLLM	xgrammar, guidance; default auto	choice, regex, JSON schema, grammar, structural_tag
SGLang	XGrammar (default), Outlines, Llguidance	JSON schema, regex, EBNF
TensorRT-LLM	xgrammar, llguidance	JSON schema, regex, EBNF, structural tag (xgrammar only)

Sources: [vLLM structured outputs](#), [SGLang structured outputs](#), [TensorRT-LLM guided decoding](#).

All three use the same grammar libraries, so constrained-decoding capability does not separate them much. What differs is how you turn it on. vLLM accepts `response_format` with `json_schema` plus a `structured_outputs` object in `extra_body`, and its main-branch docs (not the v0.32.0 docs) announce that `lm-format-enforcer` is deprecated from v0.33.0. SGLang allows only one of `json_schema`, `regex` or `ebnf` per request. TensorRT-LLM requires `guided_decoding_backend` to be set in the server's YAML config. Test your actual schemas, including the ones with large enums and deep nesting, because grammar compile time shows up in TTFT.

Prefix-heavy and agentic traffic

All three engines reuse KV for repeated prefixes. vLLM uses hashed blocks. SGLang uses a radix tree. TensorRT-LLM stores filled blocks "in a radix search tree", with reuse "enabled by default" and optional offload to host memory ([TensorRT-LLM KV cache](#)). Prefix reuse is the original design centre of SGLang, and it is the engine with the most tuning surface for it: eviction policies including the multi-turn-oriented `tlru`, a session-aware radix cache, and hierarchical caching (HiCache) ([SGLang documentation index](#)).

Worked example: why prefix reuse dominates agent workloads (illustrative, derived). Assume one agent session has a 6,000-token system prompt plus tool definitions and 10 turns, each adding 500 user/tool tokens and 200 generated tokens.

- Without reuse, turn k re-prefills the whole context: $6,000 + 700 \times (k - 1) + 500$ tokens. Summing over $k = 1$ to 10 gives $10 \times 6,500 + 700 \times 45 = \mathbf{96,500}$ **prefill tokens**.
- With perfect reuse, turn 1 prefills 6,500 tokens and each later turn prefills at most the 700 new tokens: $6,500 + 9 \times 700 = \mathbf{12,800}$ **prefill tokens**.
- Ratio: $96,500 / 12,800 \approx \mathbf{7.5\times}$ less prefill compute per session. The shared 6,000-token prefix can also be reused across concurrent sessions.

Real hit rates are lower: they depend on block granularity (vLLM caches full blocks), eviction under memory pressure, router session affinity, and templates that put timestamps or IDs at the start of the prompt. Measure the hit rate: vLLM exports `vllm:prefix_cache_hits` ([vLLM metrics](#)), and SGLang exposes Prometheus metrics with `--enable-metrics` ([SGLang observability](#)).

Operations burden

Release cadence (derived from GitHub release dates). vLLM went from v0.23.0 (2026-06-15) to v0.32.0 (2026-10-18): 9 minor releases in 125 days, about one every 14 days. SGLang went from v0.5.13 (2026-06-13) to v0.5.22 (2026-10-19): 9 releases in 128 days, also about every 14 days. TensorRT-LLM published 20 pre-releases after v1.2.1 and no new stable tag in the 191 days to 2026-10-28. The fast cadence means each upgrade is small, but a team pinning a version for six months will skip around 13 releases and face a large jump when it moves. Breaking changes do land: SGLang 0.5.20 stopped publishing CUDA 12 artifacts (see the [SGLang 0.5.20 upgrade guide](#)), and vLLM 0.29 changed runner and API behaviour (see [vLLM 0.29 upgrade checks](#)).

Build step. The old argument against TensorRT-LLM was the per-GPU, per-config engine build. On current main that step is gone. On stable v1.2.1, check which backend your deployment actually uses, because pipelines written for `trtllm-build` will not carry over to the RCs.

API surface. vLLM documents an "OpenAI-compatible API server, plus Anthropic Messages API and gRPC support" ([vLLM docs](#)). SGLang documents OpenAI-compatible, Anthropic-compatible and Ollama-compatible APIs ([SGLang documentation index](#)). `trtllm-serve` "starts an OpenAI compatible server" with `/v1/models`, `/v1/completions` and `/v1/chat/completions`, and the same page mentions a Responses API ([trtllm-serve](#)). "Compatible" never means identical. Sampling parameters, tool-call parsing, logprobs and stateful Responses features differ by engine and version.

Observability. vLLM exports Prometheus metrics on `/metrics`, including `vllm:time_to_first_token_seconds` and `vllm:kv_cache_usage_perc` ([vLLM metrics](#)). SGLang does the same behind `--enable-metrics` and adds request dump and replay ([SGLang observability](#)). The `trtllm-serve /metrics` endpoint "provides runtime-iteration statistics such as GPU memory use and inflight-batching details" ([trtllm-serve](#)). Confirm that your dashboards can read its output format before go-live.

Rollback. Pin image digests, model revisions and engine config; keep the previous image runnable.

Licensing and support

- **vLLM:** Apache-2.0 ([GitHub](#)).

- **SGLang**: Apache-2.0 ([GitHub](#)).
- **TensorRT-LLM**: "licensed under the Apache 2.0 license", with third-party portions under their own notices in the same file. GitHub's license detector reports NOASSERTION because of those additions ([LICENSE](#)). Have counsel read the file, not the badge.

None of the three repositories is a support contract. Commercial support comes through products built on the engines, such as Red Hat AI Inference for vLLM ([release notes](#)) or NVIDIA's NIM and AI Enterprise stack. Those products ship a specific, older engine version: Red Hat AI Inference 3.5.0 (GA 2026-08-27) packages vLLM v0.24.0 (tagged 2026-06-29), while upstream is at v0.32.0. GA releases get seven months of full support ([life cycle](#)). You trade feature lag for a support SLA.

A benchmark method you can reproduce

Do not copy anyone else's numbers into a selection decision, including the papers quoted above. Run the following on your own hardware:

1. **Fix everything except the engine.** Same model revision, weight precision, KV-cache dtype, GPU count, parallelism layout, driver and CUDA. If an engine cannot load your production checkpoint, that is the result.
2. **Use your traffic shape.** Take input/output lengths and arrival patterns from production logs. For agentic workloads, replay real multi-turn sessions so prefix reuse is exercised.
3. **Measure the right metrics.** Record TTFT, ITL, end-to-end latency and **goodput**, meaning requests per second that meet your TTFT and ITL SLOs. Don't optimize raw tokens per second. Definitions are in [TTFT](#), [ITL](#) and [goodput](#).
4. **Separate warm and cold.** Report results with an empty prefix cache and with a warmed one. Record the cache hit rate from the engine's metrics.
5. **Sweep load to the knee.** Increase arrival rate until goodput falls. Compare engines at the highest load at which each still meets the SLO, not at a single concurrency point.
6. **Record versions and flags.** Keep the full launch command, image digest and the date. A result without these cannot be reproduced after the next release.
7. **Check output equivalence.** Run a small evaluation set and structured-output validation on each engine; throughput from a kernel path that changes outputs is not a win.

Decision matrix

Workload or constraint	Lean toward	Why (documented basis)	Watch out for
Mixed or non-NVIDIA hardware (AMD, TPU, Ascend, Intel)	vLLM or SGLang	Both document multi-vendor support; TensorRT-LLM is NVIDIA-only	Plugin builds lag upstream; qualify per platform
NVIDIA-only fleet, Blackwell, FP4 a priority	Benchmark TensorRT-LLM alongside the others	Documented NVFP4/MXFP4 and NVFP4 KV cache on sm100/103; ModelOpt checkpoint path	Latest stable is v1.2.1 (2026-04-20); current architecture is RC-only

Workload or constraint	Lean toward	Why (documented basis)	Watch out for
Prefix-heavy, multi-turn, agentic	SGLang first, vLLM second	Radix cache with selectable eviction incl. t1ru, session-aware cache, HiCache	Router affinity decides real hit rate
Strict JSON/grammar output	Any; decide on API ergonomics	All three use xgrammar and/or llguidance	Grammar compile latency; per-request constraint limits
Day-one support for new open models	vLLM or SGLang	Transformers fallback in both	Fallback path may lack optimized kernels
Prefill/decode disaggregation at scale	Decide with the orchestrator	vLLM experimental, TensorRT-LLM beta, SGLang unlabeled	KV transport (NIXL/Mooncake) and fabric qualification
Small team, wants a support contract	vLLM via a supported distribution, or NIM	Vendor-packaged images and SLAs	Packaged versions trail upstream by many releases
Broadest quantization and checkpoint formats	vLLM	Widest documented list incl. GGUF, compressed-tensors, TorchAO	Format listed does not mean kernel exists on your GPU

FAQ

Can I switch engines later? Usually, if clients only use OpenAI-compatible endpoints and weights are in a format both engines load. The expensive parts are engine-specific: quantized checkpoint formats, disaggregation topology and KV connectors, tool-call and reasoning parsers, and dashboards built on engine metric names. A thin gateway in front makes a switch a routing change.

Are they really OpenAI-compatible? Each documents an OpenAI-compatible server. Compatibility is per endpoint and per parameter, and it changes between releases. Write contract tests for the exact request shapes your applications send, including streaming, tool calls, `response_format` and `logprobs`, and run them on every upgrade.

Which is fastest on H100 or B200? No version-independent answer exists. Run the method above with your model and traffic. Treat the paper figures (2-4× in 2023 for vLLM, up to 6.4× in 2023/2024 for SGLang) as evidence that each design worked against its baselines at the time, not as a current ranking.

If you want an independent qualification run on your own hardware, GPU Smith does this work: <https://gpusmith.com/contact>.

Sources

- [vLLM documentation](#) — vLLM project, latest, accessed 2026-10-28
- [vLLM installation](#) — vLLM project, latest, accessed 2026-10-28
- [vLLM structured outputs](#) — vLLM project, latest, accessed 2026-10-28
- [vLLM disaggregated prefilling \(experimental\)](#) — vLLM project, latest, accessed 2026-10-28
- [vLLM automatic prefix caching design](#) — vLLM project, latest, accessed 2026-10-28
- [vLLM production metrics](#) — vLLM project, latest, accessed 2026-10-28

- [vLLM supported models](#) — vLLM project, latest, accessed 2026-10-28
- [vLLM releases](#) — GitHub, accessed 2026-10-28
- [vLLM v0.32.0](#) — GitHub, 2026-10-18, accessed 2026-10-28
- [vLLM repository](#) — GitHub, accessed 2026-10-28
- [SGLang documentation](#) — SGLang project, accessed 2026-10-28
- [SGLang hardware platforms](#) — SGLang project, accessed 2026-10-28
- [SGLang CPU servers](#) — SGLang project, accessed 2026-10-28
- [SGLang TPU](#) — SGLang project, accessed 2026-10-28
- [SGLang quantization](#) — SGLang project, accessed 2026-10-28
- [SGLang structured outputs](#) — SGLang project, accessed 2026-10-28
- [SGLang PD disaggregation](#) — SGLang project, accessed 2026-10-28
- [SGLang radix cache eviction policies](#) — SGLang project, accessed 2026-10-28
- [SGLang observability](#) — SGLang project, accessed 2026-10-28
- [SGLang Transformers fallback](#) — SGLang project, accessed 2026-10-28
- [SGLang documentation index \(llms.txt\)](#) — SGLang project, accessed 2026-10-28
- [SGLang releases](#) — GitHub, accessed 2026-10-28
- [SGLang v0.5.22](#) — GitHub, 2026-10-19, accessed 2026-10-28
- [SGLang repository](#) — GitHub, accessed 2026-10-28
- [TensorRT-LLM overview](#) — NVIDIA, last updated 2026-10-26, accessed 2026-10-28
- [TensorRT-LLM supported hardware](#) — NVIDIA, last updated 2026-10-26, accessed 2026-10-28
- [TensorRT-LLM supported models](#) — NVIDIA, last updated 2026-10-26, accessed 2026-10-28
- [TensorRT-LLM KV cache system](#) — NVIDIA, accessed 2026-10-28
- [TensorRT-LLM quantization](#) — NVIDIA, accessed 2026-10-28
- [TensorRT-LLM disaggregated serving](#) — NVIDIA, last updated 2026-10-26, accessed 2026-10-28
- [TensorRT-LLM guided decoding](#) — NVIDIA, last updated 2026-10-26, accessed 2026-10-28
- [TensorRT-LLM: TensorRT backend removal](#) — NVIDIA, last updated 2026-10-26, accessed 2026-10-28
- [trtllm-serve](#) — NVIDIA, accessed 2026-10-28
- [TensorRT-LLM v1.0.0 release notes](#) — GitHub, 2025-09-24, accessed 2026-10-28
- [TensorRT-LLM v1.2.1](#) — GitHub, 2026-04-20, accessed 2026-10-28
- [TensorRT-LLM v1.3.0rc20 release notes](#) — GitHub, 2026-06-30, accessed 2026-10-28
- [TensorRT-LLM v1.4.0rc2](#) — GitHub, 2026-10-28, accessed 2026-10-28
- [TensorRT-LLM LICENSE](#) — NVIDIA, accessed 2026-10-28
- [Efficient Memory Management for Large Language Model Serving with PagedAttention](#) — Kwon et al., arXiv / SOSP 2023, submitted 2023-09-12, accessed 2026-10-28
- [SGLang: Efficient Execution of Structured Language Model Programs](#) — Zheng et al., arXiv, v1 2023-12-12, v2 2024-06-06, accessed 2026-10-28
- [Red Hat AI Inference 3.5.0 release notes](#) — Red Hat, GA 2026-08-27, accessed 2026-10-28
- [Red Hat AI Inference life cycle](#) — Red Hat, accessed 2026-10-28
- [NVIDIA NIM for LLMs 2.0.0 support matrix](#) — NVIDIA, last updated 2026-03-12, accessed 2026-10-28

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://docs.redhat.com/en/documentation/red_hat_ai_inference/3.5/html/release_notes/rhaii-350-ga-release-notes_release-notes
3. <https://access.redhat.com/support/policy/updates/rhaii>
4. <https://docs.nvidia.com/nim/large-language-models/2.0.0/reference/support-matrix.html>
5. <https://github.com/vllm-project/vllm/releases/tag/v0.32.0>
6. <https://github.com/vllm-project/vllm/releases>
7. <https://github.com/sgl-project/sglang/releases/tag/v0.5.22>
8. <https://github.com/sgl-project/sglang/releases>
9. <https://github.com/NVIDIA/TensorRT-LLM/releases/tag/v1.2.1>
10. <https://github.com/NVIDIA/TensorRT-LLM/releases/tag/v1.4.0rc2>
11. https://docs.vllm.ai/en/latest/features/structured_outputs.html
12. <https://arxiv.org/abs/2309.06180>
13. <https://docs.vllm.ai/en/latest>
14. https://docs.vllm.ai/en/latest/design/prefix_caching.html
15. <https://arxiv.org/abs/2312.07104>
16. <https://docs.sglang.io>
17. https://docs.sglang.io/docs/advanced_features/radix_eviction_policy
18. <https://github.com/NVIDIA/TensorRT-LLM/releases/tag/v1.0.0>
19. <https://github.com/NVIDIA/TensorRT-LLM/releases/tag/v1.3.0rc20>
20. <https://nvidia.github.io/TensorRT-LLM/legacy/tensorrt-backend-removal.html>
21. https://docs.vllm.ai/en/latest/getting_started/installation/index.html
22. <https://docs.sglang.io/docs/hardware-platforms/overview>
23. https://docs.sglang.io/docs/hardware-platforms/cpu_server
24. <https://docs.sglang.io/docs/hardware-platforms/tpu>
25. <https://nvidia.github.io/TensorRT-LLM/supported-hardware.html>
26. https://docs.vllm.ai/en/latest/models/supported_models.html
27. https://docs.sglang.io/docs/supported-models/transformers_fallback
28. <https://nvidia.github.io/TensorRT-LLM/models/supported-models.html>
29. https://docs.sglang.io/docs/advanced_features/quantization
30. <https://nvidia.github.io/TensorRT-LLM/features/quantization.html>
31. <https://docs.sglang.io/lms.txt>
32. https://docs.vllm.ai/en/latest/features/disagg_prefill.html
33. https://docs.sglang.io/docs/advanced_features/pd_disaggregation

- 34. <https://nvidia.github.io/TensorRT-LLM/features/disagg-serving.html>
- 35. https://docs.sglang.io/docs/advanced_features/structured_outputs
- 36. <https://nvidia.github.io/TensorRT-LLM/features/guided-decoding.html>
- 37. <https://nvidia.github.io/TensorRT-LLM/features/kvcache.html>
- 38. <https://docs.vllm.ai/en/latest/usage/metrics.html>
- 39. https://docs.sglang.io/docs/advanced_features/observability
- 40. <https://nvidia.github.io/TensorRT-LLM/commands/trtllm-serve/trtllm-serve.html>
- 41. <https://github.com/vllm-project/vllm>
- 42. <https://github.com/sgl-project/sglang>
- 43. <https://raw.githubusercontent.com/NVIDIA/TensorRT-LLM/main/LICENSE>

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.