GPU SMITH
COMPUTE ENGINEERING

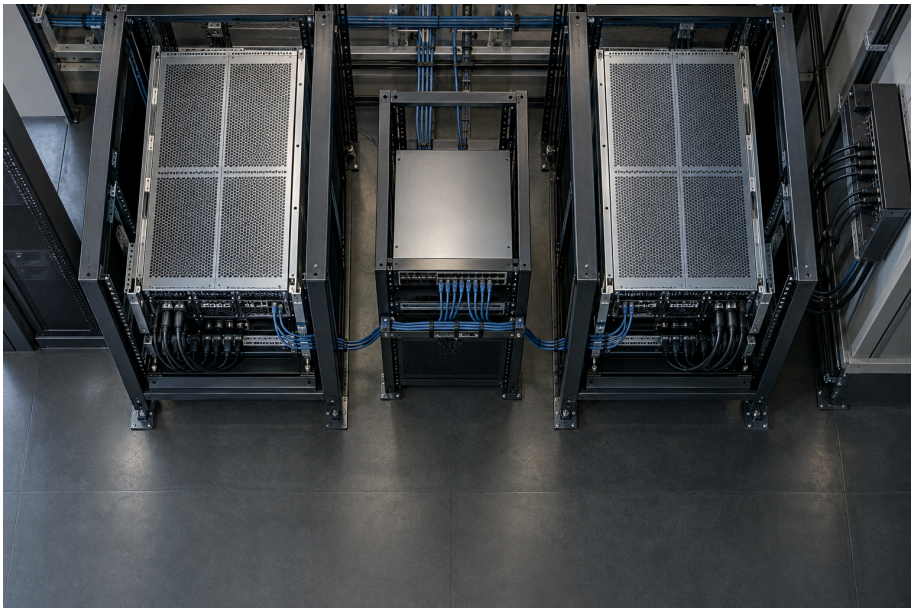
GPU SMITH / RESEARCH & PRACTICAL GUIDANCE

vLLM 0.29 Upgrade Guide: Runner V2 and API Checks

> 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 · 2026-09-19 · vllm 0.29 upgrade guide · vllm migration · model runner v2 · llm inference · api security · gpu infrastructure · performance testing · canary rollout · private ai

Original article: https://gpusmith.com/articles/en/vllm-029-upgrade-runner-api-checks



In this article

1. Executive Summary
2. Introduction and Background
3. Key Changes in vLLM 0.29.0
4. Implementation Considerations and Process Changes
5. Artifact, Capacity, and Rollout Strategy
6. Endpoint Exposure and Security Acceptance
7. Data Analysis and Evidence
8. Implications and Future Directions
9. Frequently Asked Questions (FAQs)
10. Conclusion

Executive Summary

vLLM 0.29.0 is not a routine package refresh. Released on **September 9, 2026**, at commit **98dff2a**, it makes **Model Runner V2 (MRV2)** the default for all models (github.com) (github.com) (github.com). The correct promotion question is therefore not simply whether 0.29 starts. It is whether each production tuple, meaning model, hardware backend, quantization, attention path, parallelism, speculative mode, Low-Rank Adaptation (LoRA), multimodal behavior, and Key-Value (KV) transfer, runs through the expected runner and still meets an operator-defined contract.

The release artifact choice must also be explicit. It publishes separate CUDA, ROCm, XPU, and CPU distributions, and the fetched CUDA 12.9 image metadata binds its tag to the release commit (hub.docker.com). Pin the resolved image digest or wheel hash, record the host driver and runtime, and preserve the complete dependency lock. Docker documents digest pulls as the way to select an exact image version, while pip hash-checking requires a hash for every requirement (docs.docker.com) (pip.pypa.io).

Acceptance should combine **correctness, capacity, performance, and exposure**. Replay the application's real request distribution, compare request schemas and application-visible outputs, then measure time to first token (TTFT), inter-token latency (ITL), end-to-end latency, throughput, error rate, GPU memory high-water mark, and maximum safe concurrency. Aggregate latency with distributions that remain comparable across replicas, a use case Prometheus documents for histograms (prometheus.io). Do not promote based on pull-request microbenchmarks or exact token identity alone.

Finally, **`--api-key` is not a perimeter**. The versioned security guide documents routes that remain credential-free even when the key is configured (docs.vllm.ai). Production acceptance must enumerate the live route table, test authentication route by route, block direct access, and fail the canary if any unauthorized route is reachable. This aligns with NIST's position that network location alone grants no implicit trust (csrc.nist.gov).

Introduction and Background

This report is an operational **vLLM 0.29 migration guide** for private-LLM platform engineers, site reliability engineers (SREs), security engineers, and infrastructure owners. It supplies a **vLLM Model Runner V2 validation** plan, explains the **vLLM 0.29 Model Runner V2** default, identifies **vLLM 0.29 API changes**, defines **vLLM API exposure checks**, and turns **vLLM upgrade compatibility testing** into measurable acceptance evidence. Those changes affect correctness, memory use, latency, capacity, and reachable attack surface, so a package-level smoke test is insufficient.

MRV2 is designed around persistent batch state, asynchronous scheduling, GPU-visible metadata, and explicit CUDA graph management. These are meaningful architectural changes, but they are not a promise of a universal speedup. The design page cautions that MRV2 is not yet feature-complete or rigorously tested (docs.vllm.ai).

The practical unit of change control is the **complete serving tuple**, not the vLLM version alone. Kernel choice, attention backend, model revision, tokenizer or processor, quantization, graph capture, parallelism, speculative decoding, and request mix can all change what executes or what users observe. PyTorch also warns that deterministic algorithms alone do not guarantee full application reproducibility (docs.pytorch.org). The upgrade plan therefore treats deterministic seeds as a diagnostic aid, not proof that two builds are semantically identical.

GPU Smith is an **adjacent independent engineering advisor**, not a vLLM vendor. Its published method says that vLLM and Triton are configured to measured requirements and that integration is accepted against written criteria (gpusmith.com) (gpusmith.com). That framing fits this decision: promote only when a recorded configuration passes a workload-specific acceptance contract.

Key Changes in vLLM 0.29.0

Release identity and artifact families

The first control is to identify exactly what is under test. The release publishes multiple wheel and container families rather than one interchangeable binary (github.com):

- **CUDA 13.0:** default PyPI install and `vllm/vllm-openai:v0.29.0` container.
- **CUDA 12.9:** `vllm/vllm-openai:v0.29.0-cu129` container and separate release assets.
- **ROCm:** versioned ROCm 7.2.3 wheel index and `vllm/vllm-openai-rocm:v0.29.0` container.
- **XPU:** a release-specific XPU wheel and container family.
- **CPU:** a release-specific CPU container and prebuilt wheel assets.

These labels select a family, not the operator's immutable artifact. Record the registry digest, architecture-specific manifest digest, and wheel SHA256 actually pulled. OCI guidance says retrieved content should be checked against its descriptor digest when consumed from an untrusted source (github.com). Provenance can supplement that record by describing where, when, and how an artifact was produced (slsa.dev).

Model Runner V2 becomes the default

The central behavior change is simple to state and easy to misapply: **MRV2 is the default for all models**, but it is not necessarily the effective runner for every workload. The release identifies sequence parallelism, dual-batch overlap, elastic expert parallelism, custom logits processors, and certain speculative-decoding methods among temporary gaps (github.com). It states that configured unsupported features still fall back to MRV1 (github.com). The pinned configuration also falls back when Triton is unavailable (github.com).

MRV2 can change resource behavior even when the API surface appears unchanged:

- **Persistent state:** batch state remains resident rather than being reconstructed as one monolithic per-step input.
- **Asynchronous preparation:** CPU scheduling and input work can overlap current GPU execution.
- **GPU metadata access:** include host-resident metadata traffic in profiler and latency comparisons.
- **Sampling memory:** test logprob-heavy requests as their own memory cohort.
- **Graph control:** full CUDA graphs are captured and launched through standard PyTorch APIs (docs.vllm.ai).

These mechanisms explain what to measure. They do not justify transplanting a maintainer microbenchmark into a production capacity forecast.

API behavior that needs explicit compatibility checks

The main compatibility risk is rarely a removed URL. It is a small semantic change in tokenization, chat formatting, structured output, tool-call parsing, stop handling, multimodal representation, or errors.

- **Chat templates:** compare the configured template and rendered prompt; Hugging Face states that a template should match the format used during model training (huggingface.co).
- **Multimodal templates:** the template can live on the processor rather than the tokenizer, so both revisions belong in the bill of materials (huggingface.co).
- **Structured output:** the `json` constraint targets a JSON Schema, so validate the complete schema contract (docs.vllm.ai).
- **Schema assertions:** object properties are optional unless listed as required, so a weak test schema can pass incomplete output (json-schema.org).
- **Logprobs and errors:** compare sampled-token probability, alternatives, null fields, and status codes; the compatible contract always returns the sampled token's log probability when requested (developers.openai.com).
- **Stops and seeds:** the compatible completion contract excludes the matched stop sequence from returned text, while seeded sampling remains best-effort rather than guaranteed deterministic behavior (developers.openai.com) (developers.openai.com).

Implementation Considerations and Process Changes

Freeze the production tuple before testing

An upgrade test is interpretable only when the old and new tuples are reproducible. *Table 1* is the minimum immutable bill-of-materials (BOM) diff. Populate observed values, not expected defaults.

BOM field	Old production evidence	v0.29 candidate evidence	Acceptance check
Container or wheel	Registry digest, manifest digest, or wheel SHA256	Resolved digest or hash, never only a tag	Pull by digest; verify content; retain prior artifact
Core software	vLLM commit, Python, PyTorch, Triton	v0.29.0 commit plus installed package inventory	Diff every direct and transitive dependency
GPU stack	GPU SKU, firmware, driver, CUDA or ROCm	Same fields captured from the candidate host	Match the selected artifact's compatibility matrix
Model assets	Model, tokenizer, processor, code revisions	Immutable repository commits and local hashes	Prove that only approved revisions changed
Engine path	Effective runner, attention backend, graph mode	Startup evidence and effective configuration	No assumed defaults or silent fallback
Features	Quantization, parallelism, speculation, LoRA, multimodal, KV transfer	Exact flags and environment variables	Map each workload to a qualified feature path
Network surface	Listener, proxy policy, live routes, plugins, tool servers	Route export and reachability test	Default deny outside the approved allowlist

The table turns “same model” into a falsifiable statement. Exact dependency pinning matters because a vLLM artifact is only one layer of the serving environment. Pip's secure-install guidance requires exact version, URL, or path pins in hash-checking mode (pip.pypa.io).

Determine the runner that actually executes

Create one qualification record per workload, not one per cluster. Each record should include:

- **Identity:** model and tokenizer or processor commit, served name, trust-remote-code state.
- **Backend:** CUDA, ROCm, XPU, or CPU artifact and resolved digest.
- **Execution:** effective MRV1 or MRV2 runner, attention backend, eager or graph mode.
- **Precision:** weight quantization, activation and KV-cache data types.
- **Parallelism:** tensor, pipeline, data, expert, and sequence settings.
- **Advanced paths:** speculative method and draft model, LoRA adapter, multimodal processor, KV connector.
- **Evidence:** startup log, effective configuration dump, route inventory, test case IDs, and results.

Start the candidate with production-equivalent flags, then capture the effective configuration after feature validation. Assert the runner in logs or instrumentation, exercise each optional feature, and deliberately test unsupported combinations in staging. A startup success with an MRV1 fallback is not an MRV2 qualification. Conversely, forcing MRV2 does not prove that every feature is implemented correctly.

Build a correctness regression suite

Correctness acceptance should compare application-visible invariants before considering performance:

- **Request validation:** valid and invalid schemas, missing fields, bounds, media types, and status codes.
- **Tokenization:** rendered prompts, control tokens, token IDs, truncation, and special-token handling.
- **Generation:** stop strings, stop token IDs, finish reasons, streaming chunks, usage accounting.
- **Probabilities:** sampled token, returned logprob, top alternatives, and absent or null fields.
- **Structured output:** schema validation with required fields, enums, arrays, nested objects, and refusals.
- **Tool calls:** names, arguments, IDs, parallel calls, parser failures, and streamed deltas.
- **Multimodal:** image, audio, or video placement, processor revision, modality limits, and malformed inputs.
- **LoRA:** static adapter selection, adapter identity, concurrency, and base-model fallback.
- **Failure behavior:** overload, cancellation, timeout, invalid adapter, oversized request, and dependency failure.

Include static LoRA selection and multimodal content placement in the exact request fixtures. Multimodal template state can live on the processor rather than the tokenizer, which makes the processor revision part of reproducibility (huggingface.co). These are examples of semantic contracts that token-only comparisons can miss.

Classify differences before failing them. A schema violation, changed stop behavior, wrong tool call, or new authorization result is application-visible. Small floating-point or probability differences may be acceptable if outputs remain within a predeclared semantic and statistical envelope. Record the seed, but do not make bitwise identity the only criterion.

Artifact, Capacity, and Rollout Strategy

Select and pin the correct artifact

Match artifact family to the host rather than trying the default wheel everywhere. NVIDIA states that the installed driver must meet or exceed the CUDA Toolkit minimum, and documents **driver 580 or later** for CUDA 13.x minor-version compatibility (docs.nvidia.com) (docs.nvidia.com). Its table lists CUDA 12.9 GA minima of **575.51.03 on Linux x86_64** and **576.02 on Windows x86_64** (docs.nvidia.com). AMD likewise directs operators to a release-specific matrix covering ROCm, operating systems, and accelerators (rocm.docs.amd.com).

For each candidate host:

- **Resolve:** convert the tag to registry index and platform-manifest digests.
- **Inspect:** capture OS, architecture, CUDA or ROCm libraries, Python, PyTorch, Triton, FlashInfer, and vLLM commit.
- **Verify:** compare driver, firmware, runtime, and GPU support against the relevant matrix.
- **Lock:** hash wheels and every Python dependency; retain the lockfile with the test result.
- **Archive:** cache the previous image, configuration, model manifest, proxy policy, and dashboards.
- **Label:** attach the candidate digest and configuration hash to metrics, logs, and traces.

Docker's inspection tooling exposes image and manifest-list digests for this record (docs.docker.com). Do not infer a runtime from a floating tag or generic installation page.

Qualify capacity with the production distribution

Performance testing should preserve fixed model revisions, hardware, precision, input and output length distributions, concurrency or request-rate shape, and feature mix. Warm and cold behaviors are separate tests because image pull, model load, kernel compilation, graph capture, and cache population affect readiness.

Measure at least:

- **Startup:** image pull, process start, model load, graph capture, warmup, and healthy-ready time.
- **Latency:** TTFT, ITL, time per output token, and end-to-end p50, p95, and p99.
- **Capacity:** completed tokens per second, requests per second, maximum admitted concurrency, and queue time.
- **Reliability:** HTTP and engine error rate, cancellation, timeout, crash, GPU reset, and out-of-memory count.
- **Memory:** model footprint, free memory before load, KV-cache allocation, KV utilization, and high-water mark.
- **Quality:** structured-output pass rate, tool-call pass rate, stop correctness, and task-specific scoring.

Prometheus histograms permit latency percentiles to be aggregated across replicas with `histogram_quantile()` (prometheus.io). Check the candidate's live metrics before changing dashboards. NVIDIA similarly recommends using the exporter's `/metrics` endpoint to see effective runtime output (docs.nvidia.com).

Canary and rollback

Run the old and candidate artifacts on identical qualified hardware. Replay a permitted trace or shadow sanitized traffic, then route a small live share only after correctness and exposure gates pass. Kubernetes defines a canary as a new version receiving a small percentage of production traffic alongside the stable

version (kubernetes.io). Its Gateway API example shows an explicit **90/10** stable-to-canary split (kubernetes.io). That is an example, not a universal starting percentage.

Define abort thresholds before traffic starts. Include correctness mismatch, unauthorized route reachability, crash or reset, out-of-memory event, tail-latency breach, error-rate breach, loss of rollback capacity, and rollback-time breach. On abort, remove candidate traffic, restore stable capacity, and preserve evidence. Kubernetes notes that scaling a canary to zero retains its configuration for inspection (kubernetes.io).

Endpoint Exposure and Security Acceptance

Treat the live route table as the source of truth

The acceptance test must probe the deployed middleware with and without credentials.

Table 2 turns the documented exposure classes into a proxy policy. Exact enabled routes depend on flags, task, and plugins, so export `app.routes` or an equivalent live inventory.

Path or class	Documented function and built-in auth status	Required audience	Proxy and control action
`/v1/*`, `/v2/*`, selected `/inference/*`	Intended authenticated API families; validate live route access	Approved clients	Allow exact required methods; enforce proxy identity, quota, size, and timeout
`/invocations`, `/generative_scoring`, `/pooling`, `/classify`, `/score`, `/rerank`	Credential-free inference variants are listed	Usually none externally	Block by default; expose only through a separately authenticated use case
`/pause`, `/resume`, `/abort_requests`, scaling and weight routes	Operational control without built-in key enforcement	Trusted operators only	Put on a separate management plane; deny from client networks
`/tokenize`, `/detokenize`, `/health`, `/ping`, `/version`, `/load`	Utility and health routes without built-in key enforcement	Load balancer or monitoring only	Allow exact source identities and methods; minimize response exposure
`/tokenizer_info`	Optional route that can reveal templates and configuration	Administrators only	Keep disabled or restrict to the management plane
Development and cache-control routes	Present only with `VLLM_SERVER_DEV_MODE=1`	No production audience	Assert the variable is absent; block paths independently
`/start_profile`, `/stop_profile`	Present when profiler configuration enables them	Local diagnostic workflow only	Disable in production; never publish through the client proxy
Dynamic LoRA routes	Model-changing operations under `/v1` when runtime updates are enabled	Trusted administrators only	Separate authorization and audit trail from ordinary inference
Plugin-defined routes	Outside protected prefixes unless the plugin authenticates them	Explicitly approved consumers	Allowlist plugin names and paths; re-enumerate routes after startup

Path or class	Documented function and built-in auth status	Required audience	Proxy and control action
Tool-server calls	External capability, disabled by default	Explicitly approved workflows	Keep off unless required; isolate credentials, egress, and tool permissions

The important distinction is **reachable** versus **enabled**. The security guide says `/tokenizer_info` can expose chat templates and tokenizer configuration (docs.vllm.ai). The proxy should still deny optional development and profiler paths even when startup policy says they are disabled. OWASP independently recommends keeping management endpoints off the public Internet (cheatsheetseries.owasp.org).

Enforce a narrow production boundary

The acceptance sequence should be mechanical:

- **Enumerate:** capture every live method and path after all plugins and routers load.
- **Classify:** assign end-user, service, operator, monitoring, or no audience.
- **Deny:** block direct network access to vLLM and default-deny every unapproved path at the proxy.
- **Authenticate:** enforce workload or user identity at the proxy, including on nominally protected vLLM routes.
- **Authorize:** separate inference, monitoring, and model-changing administration roles.
- **Constrain:** limit body size, tokens, output count, concurrency, request rate, and time.
- **Observe:** log principal, route, model, status, latency, admitted resource envelope, and policy decision.
- **Test:** send authenticated and unauthenticated requests to every route from allowed and disallowed networks.

OWASP recommends maximum sizes for incoming parameters and payloads, plus per-client interaction limits (api-security.owasp.org) (api-security.owasp.org). It also advises keeping management endpoints off the public Internet and recording audit events before and after security-relevant actions (cheatsheetseries.owasp.org) (cheatsheetseries.owasp.org). NIST's zero-trust guidance reinforces that network location or ownership alone does not create implicit trust (csrc.nist.gov).

Resource controls need workload-specific values. The vLLM guide documents `VLLM_MAX_N_SEQUENCES` with a default of **16,384**, but that ceiling is not a safe production quota for every model or GPU (docs.vllm.ai). Set smaller request and aggregate limits from measured memory and latency envelopes. NGINX can restrict access by client address, but address rules should complement identity-aware authorization, not replace it (nginx.org).

Data Analysis and Evidence

The quantitative core of an upgrade is a paired experiment, not a borrowed benchmark. Old and new candidates must receive the same trace distribution on the same qualified hardware, with the same model, tokenizer, precision, prompt-length distribution, output-length distribution, request rate, concurrency policy, and feature mix. Preserve raw request IDs and metric exports so that aggregate changes can be traced to individual failures.

Table 3 is a results and abort matrix. Operators must fill the threshold column from their service-level objectives and risk policy. This report intentionally does not invent universal limits.

Measure	Required evidence	Comparison	Operator-supplied acceptance or abort rule
Correctness	Schema, finish reason, stop behavior, tool-call and multimodal case results	Old versus new pass rate and mismatch classes	Zero critical contract regressions; explicit tolerance for numerical variation
Request distribution	Input and output token histograms, feature flags, arrival process	Demonstrate paired workload equivalence	Reject run if distributions differ beyond the experiment plan
Latency	TTFT, ITL, end-to-end p50, p95, p99	Candidate delta by request cohort	Abort on the service's declared tail-latency breach
Throughput	Requests and generated tokens per second at fixed load	Saturation point and stable operating point	Candidate must sustain declared demand with reserve
Reliability	HTTP errors, engine errors, timeouts, cancellations, crashes and resets	Rate and failure class	Abort on declared error-rate or any critical failure trigger
Memory and KV cache	HBM before load, post-load, peak, KV allocation and utilization	Headroom at each concurrency point	Maintain declared reserve; abort on out-of-memory
Startup	Pull, load, compile, graph capture, warmup and readiness durations	Cold and cached paths	Rollback objective must include measured warmup time
Exposure	Live routes and authorization outcomes by identity and network	Expected versus observed matrix	Abort on any unauthorized successful response
Rollback	Drain, scale, start, warm, health check and traffic restore times	Measured recovery against objective	Abort promotion if prior capacity cannot be restored in time

The v0.29.0 benchmark can cap actual concurrency separately from arrival rate (docs.vllm.ai). Use that separation to test both queueing and execution saturation. For KV capacity, measure free KV bytes after model load and graph capture, then estimate the per-request envelope from the production token distribution:

```
admitted concurrency <= floor((measured free KV bytes - reserve bytes) / measured p95 KV bytes per request)
```

This is a planning bound, not a theoretical constant. Validate it by load testing because prefix caching, multimodal state, speculative decoding, parallelism, and fragmentation can change realized memory. vLLM permits an explicit KV-cache byte allocation for finer control and reports group-aware cache capacity per data-parallel engine (docs.vllm.ai) (docs.vllm.ai).

Version labels are useful only when bounded. Prometheus notes that every unique label-set combination creates a new time series, so use controlled values such as release, artifact digest prefix, runner, and configuration ID rather than request-level identifiers (prometheus.io). The evidence pack should contain the BOM diff, qualification matrix, raw benchmark output, route audit, proxy policy, dashboards, abort decisions, rollback timing, and named approvers.

Implications and Future Directions

MRV2's default status changes the baseline for future upgrades. MRV1-specific optimization is a shrinking investment, but the immediate response should not be to force every workload onto MRV2. Instead, classify remaining V1 workloads, document why each falls back, and retest them against subsequent patch releases. This makes retirement work visible without turning an unsupported combination into an availability risk.

Three practices follow from the evidence:

- **Make runner identity observable.** Emit release, artifact digest, runner, backend, model revision, and configuration hash as bounded deployment metadata.
- **Promote feature paths, not images.** A digest is eligible only for the exact model and option combinations that passed.
- **Treat routes as deployable state.** Re-audit route inventory whenever vLLM, flags, plugins, tool servers, or proxy configuration changes.
- **Retain evidence by patch.** Each 0.29.x candidate should have a new BOM diff, targeted regression run, exposure audit, and rollback rehearsal.
- **Separate compatibility from optimization.** First prove application contracts, then tune graph capture, KV allocation, batching, and concurrency.
- **Recheck metrics before dashboard migration.** Metric names and labels are code-version behavior, not timeless API contracts.

Optional tool-server and plugin surfaces should be part of configuration review and route testing, not assumed absent forever. Apply the same bounded-input and per-client request controls to these paths that OWASP recommends for API resource consumption (api-security.owasp.org).

The durable outcome is a repeatable promotion system. It can qualify later vLLM patches, new GPUs, new model revisions, and new attention or speculative paths using the same evidence structure. That is more valuable than a one-time claim that 0.29 is faster or safer.

Frequently Asked Questions (FAQs)

Are there vLLM 0.29 breaking changes?

It is an execution-path change with possible application-visible effects, even where HTTP schemas remain compatible. MRV2 becomes the default, unsupported combinations can fall back to MRV1, and tokenizer, structured-output, tool-call, memory, and graph behavior require regression tests. Treat it as a controlled production change rather than assuming semantic compatibility from the version number.

How to validate a vLLM upgrade

Record the complete workload tuple, capture the effective runner after startup validation, and exercise every enabled feature. Compare old and new correctness first, then capacity and latency under an identical trace distribution. The test result must say which runner actually executed, not merely which default the release announced.

Does `--api-key` secure every vLLM endpoint?

No. Versioned documentation lists unauthenticated inference variants, operational controls, utilities, and optional development surfaces. Put vLLM behind a reverse proxy, deny direct access, allowlist exact routes and methods, apply separate authorization for administration, and test every live route without credentials. OWASP's guidance to keep management endpoints off the Internet supports that design (cheatsheetseries.owasp.org).

Which vLLM 0.29 artifact should be installed?

Choose the release family that matches the qualified GPU backend and host driver, then pin the resolved digest or wheel hash. The default PyPI wheel should not be assumed suitable for every CUDA, ROCm, or XPU host. Preserve the installed package inventory and runtime inspection with the test evidence.

What should trigger rollback?

Operator-defined triggers should include critical output-contract mismatch, unauthorized route reachability, crash or GPU reset, out-of-memory, tail-latency or error-rate breach, and inability to restore the prior qualified capacity within the recovery objective. Kubernetes supports rollback to a retained specific revision, but that mechanism still needs a timed rehearsal (kubernetes.io).

Conclusion

vLLM 0.29.0 should be promoted by evidence, not by version label. Its September 2026 release moves all models to an MRV2 default while preserving conditional MRV1 paths, and it publishes distinct CUDA, ROCm, XPU, and CPU artifacts. That combination requires an immutable BOM and a feature-path qualification matrix for every production workload.

The acceptance order matters. First, prove request validation, tokenization, chat templates, stop conditions, logprobs, structured output, tool calls, multimodal processing, LoRA, and failure behavior. Second, measure startup, KV capacity, concurrency, latency distributions, throughput, error rates, and GPU memory using the

real workload shape. Third, enumerate the deployed routes and prove that the reverse proxy denies every path, method, identity, and network combination outside the approved policy.

Promotion should proceed through a version-labeled canary with predeclared abort thresholds and a timed rollback. The retained evidence pack should include artifact hashes, dependency locks, runtime inspection, effective runner, raw regression data, metrics mapping, route audit, proxy policy, and rollback timings. With those controls, 0.29 becomes a bounded engineering change. Without them, the new default runner and incomplete built-in authentication boundary remain assumptions hidden behind a successful health check.

Source links

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

1. https://github.com/vllm-project/vllm/releases/tag/v0.29.0
2. https://github.com/vllm-project/vllm/commit/98dff2a81d747d1dba01a47f939f48c3526d4206
3. https://hub.docker.com/layers/vllm/vllm-openai/v0.29.0-cu129/images/sha256-3e10e8189823e0f7ae4620c271bcdaaf64127ec7d0edc351591a508498b7684a
4. https://docs.docker.com/reference/cli/docker/image/pull/
5. https://pip.pypa.io/en/latest/topics/secure-installs/
6. https://prometheus.io/docs/practices/histograms/
7. https://docs.vllm.ai/en/v0.29.0/usage/security/
8. https://csrc.nist.gov/pubs/sp/800/207/final
9. https://docs.vllm.ai/en/latest/design/model_runner_v2/
10. https://docs.pytorch.org/docs/main/generated/torch.use_deterministic_algorithms.html
11. https://gpusmith.com/
12. https://github.com/opencontainers/image-spec/blob/main/descriptor.md?plain=1
13. https://slsa.dev/spec/v1.2/provenance
14. https://github.com/vllm-project/vllm/blob/v0.29.0/vllm/config/vllm.py
15. https://huggingface.co/docs/transformers/chat_templating_writing
16. https://docs.vllm.ai/en/v0.29.0/features/structured_outputs/
17. https://json-schema.org/understanding-json-schema/reference/object
18. https://developers.openai.com/api/reference/java/resources/completions/methods/create
19. https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/
20. https://rocm.docs.amd.com/en/docs-7.0.2/compatibility/compatibility-matrix.html
21. https://docs.docker.com/dhi/explore/security-concepts/digests/
22. https://docs.nvidia.com/datacenter/dcgm/latest/reference/dcgm-exporter-metrics.html
23. https://kubernetes.io/docs/tutorials/stateless-application/canary-deployment/
24. https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html
25. https://api-security.owasp.org/editions/2023/en/0xa4-unrestricted-resource-consumption/
26. https://nginx.org/en/docs/http/ngx_http_access_module.html

27. https://docs.vllm.ai/en/v0.29.0/cli/bench/serve/
28. https://docs.vllm.ai/en/v0.29.0/api/vllm/config/cache/
29. https://prometheus.io/docs/practices/naming/
30. https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

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.