



Prefix Cache Reuse Audit: Tokens, Eviction, Tenant Limits

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 · 2026-10-31 · prefix cache reuse audit · vllm prefix caching · llm prefix caching · kv cache prefix reuse · prefix cache hit rate · prefix cache eviction · cache_salt · multi-tenant cache isolation · token identity · inference troubleshooting

Original article: <https://gpusmith.com/articles/en/prefix-cache-reuse-audit-eviction-tenant-boundaries>



In this article

1. Executive Summary
 2. Introduction and Background
 3. Key Changes in the Reuse Contract
 4. Request-Trace Audit: Capture and Miss Taxonomy
 5. Controlled Paired Tests and Counter Reconciliation
 6. Implementation Considerations and Process Changes
 7. Data Analysis and Evidence
 8. Implications and Future Directions
 9. Frequently Asked Questions (FAQs)
 10. Conclusion
-

Executive Summary

A prefix cache reuse audit answers one operational question: when a prompt that looks repeated does not reuse earlier work, is the cause the tokens, the tenant policy, the engine configuration, or retained-cache pressure? This report proposes a single-replica, request-trace method for answering it before any decision to buy GPUs or adopt an external cache. As of October 31, 2026, the vLLM documentation defines block identity as a hash of the block's tokens plus the tokens before it, with extra identity such as LoRA (low-rank adapter) IDs, multimodal hashes and cache salts. Only full blocks are cached, and reuse reduces prefill, not decode.

The central finding is structural: a miss is a statement about the first token position where two requests diverge, not about how similar the prompts look. A timestamp in the first 16 tokens can zero out reuse for a 1,200-token prompt even when roughly 988 of 1,200 tokens are identical, and the audit's worksheet makes that visible with an eligibility ceiling of $\text{floor}(\text{shared prefix tokens} / \text{block tokens}) \times \text{block tokens}$, further capped because the last prompt token is always recomputed ([raw.githubusercontent.com](https://raw.githubusercontent.com/olliey/llm-d/main/content/trace/audit/worksheet.md)). The report defines eight miss classes, each with a falsifying paired test, and a named "unknown" class so attribution is never forced.

Counters need careful handling. vLLM exports prefix cache queries and hits in tokens, not requests, external hits are counted separately, and counters reset to zero when the instance restarts (prometheus.io). A token hit ratio is therefore the delta of hits over the delta of queried tokens in one stable window, and it is undefined when the denominator is zero. Averaging per-replica ratios is wrong: a hypothetical 60% replica on 1,000 tokens and a 20% replica on 9,000 tokens yield 24%, not 40%.

Tenant isolation is a constraint, not a defect. The cache salt is injected into the first block's hash so only same-salt requests share blocks ([raw.githubusercontent.com](https://raw.githubusercontent.com/olliey/llm-d/main/content/trace/audit/worksheet.md)), and academic work describes timing channels when caches are shared across users (arxiv.org). Per-user isolation prevents cross-tenant leakage but eliminates shared reuse (arxiv.org), so the audit never tries to raise reuse across approved trust boundaries.

Existing guides already explain prompt layout (handbook.modular.com) and distributed routing (llm-d.ai); recent research proposes trace-based working-set estimation (arxiv.org). No benchmark was executed for this report, so it publishes a worksheet and illustrative arithmetic, never a claimed speedup. A linear saved-prefill

fraction is not a throughput or GPU-count forecast. The decision order is: preserve required prompt semantics, stabilize representation where valid, correct configuration, change retention policy, and only then hand evidence to routing and capacity references.

Introduction and Background

Operators of private inference services often notice that prefix cache hit rate is lower than the prompt design suggests. A system prompt is shared, a document set is shared, a conversation grows turn by turn, and yet the engine reports a modest fraction of reused tokens. The tempting responses are to add GPUs, to bolt on an external cache tier, or to disable tenant separation. Each of those is expensive or risky before the cause is known.

This report is written for enterprise private-inference operators, application platform engineers and performance owners. It treats the question as an audit: capture what the engine actually saw, classify each miss against a closed taxonomy, and run controlled paired tests that can falsify each class. It deliberately starts with one pinned serving replica, so that multi-replica routing is a confounder to isolate rather than the subject. The author, GPU Smith, is an independent private AI infrastructure engineering consultancy acting here as an adjacent advisor, not a provider of inference engines, caches or cloud capacity.

Useful guidance already exists, and this report does not claim otherwise. The `llm-d` project describes distributed cache locality and scheduling, noting that each vLLM pod manages its own cache in isolation. The Modular handbook explains exact-prefix mechanics and prompt organization, including that even a single character difference breaks the cache. A Ceph engineering report documents a reproducible cold-start test method for external caching (www.ceph.com). A September 2026 preprint, KVSET, defines a KV (key-value) cache working set as the minimum capacity needed to reach a target hit rate (arxiv.org). The distinction here is narrower: a single-replica trace that separates changed token identity, deliberate policy separation, block eligibility, cold state and eviction, and that ends in a decision about where to act.

Two scope notes apply. First, vLLM documentation moves quickly; the pages read for this report are the "latest" documentation as of October 31, 2026, and PyPI listed vllm 0.32.0 with an October 18, 2026 upload date (pypi.org). Operators should pin the exact version, model, attention layout and tokenizer revision they run, and re-read the matching documentation before copying any command. Second, the mechanics below are drawn from documentation and source comments, not from a benchmark run by GPU Smith. They motivate tests; they do not replace them.

Key Changes in the Reuse Contract

This section collects the mechanics that decide whether a repeated prompt can reuse earlier work. Each is stated as documented, with the audit implication following.

Block-chained token identity

vLLM hashes each KV block by the tokens in the block and the tokens in the prefix before it (docs.vllm.ai). The hash combines a parent hash, the block's exact tokens and extra values that make the block unique, such as LoRA IDs (docs.vllm.ai). Because each block depends on every earlier block, a single changed token early in

the prompt changes every later block's identity. For the first block, a seed stands in for the missing parent, and the source describes it as the hash seed for the first block of any prefix block sequence (raw.githubusercontent.com).

The audit implication is that identity is token identity, not text identity. Two prompts that render to the same words can still differ if the chat template, special tokens or tokenization boundary differ. The Hugging Face documentation notes that chat templates already include the necessary special tokens (huggingface.co), so a client that adds them again changes the token sequence. The Tokenizers documentation describes pre-tokenization as splitting input so that the model does not build tokens across multiple splits (huggingface.co), and the Transformers tokenizer summary notes that a pre-tokenizer splits text on whitespace or other rules (huggingface.co), and the SentencePiece README guarantees identical tokenization from the same model file in any environment (github.com), which is why the tokenizer revision belongs in the trace. A research paper names the prompt boundary problem, in which tokenizing a prompt and a completion separately differs from tokenizing the combined text (arxiv.org).

Hash algorithm choice is a configuration fact worth recording. On recent vLLM releases the default hashing algorithm is sha256 (confirm it on your pinned version), and the engine arguments page explains that it is the default because it is the most secure choice to avoid hash collisions (docs.vllm.ai). For a single-replica audit this mostly matters for restarts and upgrades: record the algorithm and version, because comparing fingerprints across versions or algorithms is not meaningful without them.

Full-block eligibility and the last-token rule

Only full blocks are cached (docs.vllm.ai). The source documents a second limit: when all tokens would hit the cache, the engine must still recompute the last token to obtain logits (raw.githubusercontent.com), and because allocation is block aligned this can trigger recomputation of an entire block (raw.githubusercontent.com). So even an identical request sent twice cannot report a 100% token hit.

The default block size is 16 tokens according to the PagedAttention paper (arxiv.org), but current engine documentation describes the setting only as a block size in tokens, so record the effective value from your startup log rather than assuming it.

Hybrid-model group constraints

The conventional formula assumes one full-attention cache. Hybrid models differ. In vLLM's hybrid KV cache manager, the cache hit prefix of each group is checked and the intersection is returned (docs.vllm.ai). For Mamba-style state, the documentation says state is stored only on the Mamba block grid (docs.vllm.ai), so a hit can resume only at a block boundary of that grid. For these models, the eligibility ceiling in this report is only a starting hypothesis and must be replaced by the documented group constraints for the pinned version.

Tenant separation with cache_salt

vLLM supports isolating prefix cache reuse through an optional per-request salt. The value is injected into the hash of the first block, which means only requests with the same salt can reuse cached blocks (docs.vllm.ai). The documentation frames the purpose as preventing an adversary from inferring cached content by observing

latency differences, and the server documentation describes the request field as a guard against guessing prompts in multi-user environments. The source comment adds that the salt remains the mechanism for intentional cache isolation ([raw.githubusercontent.com](https://raw.githubusercontent.com/vllm-project/vllm/main/docs/requirements.md)).

Adapters carry identity too. The source includes the adapter path in the hash so that re-pointing a LoRA name at a different adapter changes block identity ([raw.githubusercontent.com](https://raw.githubusercontent.com/vllm-project/vllm/main/docs/requirements.md)). In an audit, the tenant and adapter label of every request is therefore a column in the worksheet, represented by an approved opaque label and never by the salt value itself.

Counters, per-request evidence and resets

The production metrics define prefix cache queries in terms of queried tokens ([docs.vllm.ai](https://docs.vllm.ai/en/latest/metrics.html)) and hits in terms of cached tokens ([docs.vllm.ai](https://docs.vllm.ai/en/latest/metrics.html)). External hits from a KV connector are counted in separate metrics ([docs.vllm.ai](https://docs.vllm.ai/en/latest/metrics.html)), and a by-source counter splits hit tokens by the tier that supplied them ([raw.githubusercontent.com](https://raw.githubusercontent.com/vllm-project/vllm/main/docs/requirements.md)). The metrics design page explains that the periodic log line uses a window fixed to the 1,000 most recent queries ([docs.vllm.ai](https://docs.vllm.ai/en/latest/metrics.html)), so it is not interchangeable with a counter delta over your own window.

For per-request evidence, the server can include prompt token details in the usage object when started with the corresponding flag, which defaults to off ([docs.vllm.ai](https://docs.vllm.ai/en/latest/metrics.html)). The statistics module separates tokens prefilled locally as actual compute work ([raw.githubusercontent.com](https://raw.githubusercontent.com/vllm-project/vllm/main/docs/requirements.md)) from cache-hit tokens, and a histogram reports new KV tokens computed during prefill. These are the fields that let a trace row say how many tokens were actually processed.

Three bookkeeping rules affect reconciliation. Requests that skip the lookup, for example when prompt logprobs are required, do not use the cache ([raw.githubusercontent.com](https://raw.githubusercontent.com/vllm-project/vllm/main/docs/requirements.md)), and the V1 guide states that such requests ignore the prefix cache and recompute the prefill of the full prompt. Stats are recorded at admission, so unscheduled lookups are not counted. Previously preempted requests are tracked in separate fields ([raw.githubusercontent.com](https://raw.githubusercontent.com/vllm-project/vllm/main/docs/requirements.md)), and per-request prompt statistics are tracked on the first scheduled prefill rather than on post-preemption repeats ([raw.githubusercontent.com](https://raw.githubusercontent.com/vllm-project/vllm/main/docs/requirements.md)). Whether your pinned version exports recomputation after preemption in the main counters should be verified, not assumed.

Request-Trace Audit: Capture and Miss Taxonomy

What to capture

The audit begins by writing down the qualified workload: one model, one tokenizer and chat template revision, one replica, one tenant policy, one cache mode, and the user latency and quality requirements the change must respect. Then capture, per request, the input the engine actually saw. Do not retain private prompt bodies in any shared or public artifact; keep fingerprints and counts.

- **Rendered message bytes:** the exact string after the chat template, hashed with a keyed or salted digest of your own, not stored in clear text.
- **Token IDs:** the sequence after tokenization, or at minimum fingerprints at fixed checkpoints (for example the first 16, 64, 256 and 1,024 tokens).

- **Template inputs:** system and tool templates, tool definition ordering, document ordering, and any timestamp, request ID or user field placed in the prompt.
- **Revisions:** model, tokenizer, chat template and adapter revisions, plus the vLLM version and hash algorithm.
- **Policy label:** an approved opaque tenant or trust-group label standing in for the salt value.
- **Timing and placement:** arrival time, serving instance, queue delay and TTFT (time to first token).
- **Engine evidence:** cached tokens from the usage object where enabled, and metrics deltas around the request.

A caution on the cached-token field: it is a hit signal. Research on shared caches shows that response timing depends on what has been cached, so cached prompts are processed faster than non-cached prompts. Keep per-request cached-token counts in internal telemetry. Whether to expose them to tenants is a policy decision for the operator; this report's inference is that mutually distrustful tenants should not receive them.

The miss taxonomy

Table 1 below lists the miss classes. Each pairs an observable signal with a falsifying test and a next action. Classes are mutually exclusive per request at the first divergence, and "U" is mandatory when evidence is missing.

Class	Meaning	Evidence in the trace	Falsifying paired test	Next action
M0 Cold or first use	No earlier request had this prefix since the last restart or reset	First occurrence of the prefix fingerprint after restart; zero reuse	Send the same request twice; second request should reuse	Exclude from steady-state hit rate; report separately
M1 Changed initial tokens	Tokens diverge early (timestamp, request ID, user field, template change)	Longest common initial token count is below the first block boundary	Remove or move the variable field; reuse should rise	Stabilize representation if semantics allow
M2 Shared suffix, no shared prefix	Identical content appears, but after a different start (reordered documents, tool lists)	High overall similarity but low initial-token match	Restore the original order; reuse should return to the ceiling	Fix ordering or serialization
M3 Ineligible partial block	Shared tokens end inside a block, or the last token is recomputed	Observed reuse equals the eligibility ceiling, below the shared count	Vary shared length by one block; reuse should step by one block	Accept; document the ceiling
M4 Deliberate policy split	Different salt, adapter or multimodal content	Different policy label or adapter revision, same tokens	Same label twice should reuse; different labels must not	Preserve; never "fix"
M5 Absent or evicted	State existed but was displaced by pressure	Reuse seen at short gaps but not long gaps; pressure signals	Repeat with cache pressure raised and lowered	Change retention or capacity policy
M6 Other instance	Request landed on a replica without the state	Different instance ID than the original	Pin to one replica and repeat	Hand to the routing reference

Class	Meaning	Evidence in the trace	Falsifying paired test	Next action
M7 Unsupported or skipped	Lookup skipped or model path not cache-eligible	Prompt logprobs requested, caching disabled, unsupported layout	Remove the feature and repeat	Correct configuration or accept
U Unknown	Evidence incomplete	Missing token IDs, label or counters	None until captured	Capture and re-run

The table is deliberately conservative. M4 is not a miss to repair, because separation by salt is the documented purpose of that field. M5 is distinguished from M0 by timing and pressure, since vLLM evicts the least recently used free block (docs.vllm.ai) and a request's last block is evicted first because it hashes more tokens. M6 is included only so that it can be excluded on a pinned replica. M2 is common in retrieval workloads, where the llm-d post notes that the exact documents and their order often change between queries (llm-d.ai).

The token-identity worksheet

Table 2 shows the worksheet columns with one illustrative row. The row is a hypothetical example, not a measurement.

Request ID	Prefix fingerprints (16 / 64 / 1,024)	Longest common initial tokens vs reference	Eligible boundary	Tenant / adapter label	Instance	Cache state	Observed reuse (tokens)	Misclass	Next test
R-0007 (Hypothetical Example)	A1 / B4 / C9 vs reference A1 / B4 / D2	1,000	992	tenant-a / adapter v3	replica-1	warm, short gap	992	M3	Shift shared length by 16 and confirm a one-block step

The eligibility ceiling for conventional fixed-block, full-attention caching can be written as $\text{floor}(\min(\text{shared prefix tokens}, \text{prompt tokens} - 1) / \text{block tokens}) \times \text{block tokens}$. This is a derived planning formula based on the documented full-block rule and last-token recompute rule, not a published vLLM expression, and it applies to conventional fixed-block full-attention caching only. If the request or cache contract caps reuse lower, use the lower cap. Hybrid models need their documented group constraints instead.

Interpreting the row: 1,000 shared initial tokens with a 16-token block gives $\text{floor}(1000 / 16) \times 16 = 992$ eligible tokens, and an observed reuse of 992 is explained, so the class is M3, not a defect. A gap between ceiling and observation is what demands further classification.

Controlled Paired Tests and Counter Reconciliation

Paired tests with a named baseline

Each experiment changes one factor and restores a named baseline before the next. The Ceph report offers a pattern worth copying: it restarted the engine and cleared stored blocks so that all KV caches were cold at the beginning of each test (www.ceph.com). For a single replica, the equivalent is a restart or a documented cache reset. vLLM exposes a reset endpoint in development mode, listed as able to disrupt service, and the documentation warns that the development endpoints should not be used in production (docs.vllm.ai). A reset removes all cached hashes so that no new blocks will hit (raw.githubusercontent.com), but it fails if blocks are still in use (raw.githubusercontent.com). Use these on a staging replica, drain the queue first, and check the returned status.

The vLLM benchmarking documentation adds a warning that applies to audits too: repeating a run against the same server can reuse prompts left in the prefix cache and inflate results. That is why baseline restoration is a named step, not an afterthought.

Table 3 lists the paired tests. All use synthetic, approved prompts.

Test	Change	Expectation if the hypothesis holds	Confounder to control
T1 Same request twice	None	Second request reaches the ceiling, below 100% because of the last-token rule	Cold state; send once first, and wait for the first response to start before sending the second (hosted APIs document that an entry only becomes available after the first response starts, (platform.claude.com))
T2 Static prefix, changing suffix	Suffix only	Reuse equals the largest full-block multiple of the static prefix	Block size; read it from the log
T3 Timestamp near the start	Insert a changing field in the first 16 tokens	Reuse falls to the first block boundary or zero	Template or tokenizer drift
T4 Reordered documents	Swap two documents	Reuse stops at the first differing document	Document lengths
T5 Different trust labels	Change salt label only	No reuse across labels; reuse within a label	Gateway dropping the salt field
T6 Varying inter-request gaps	Gap of seconds, minutes, longer	Hits persist until pressure displaces blocks, not for a fixed time	Background traffic
T7 Cache pressure	Raise concurrent long requests	Reuse falls as free blocks are consumed	Preemption counters
T8 Engine restart	Restart, repeat T1	First request is cold; counters reset	Prometheus reset handling

Table 3 contains two points that often surprise teams. First, T6 does not measure a time-to-live. Hosted APIs document lifetimes, for example a default 5-minute cache lifetime that is refreshed on use (platform.claude.com) with a 1-hour option (platform.claude.com), or, under OpenAI's model-dependent retention options, typically around 5 to 10 minutes of inactivity for the `in_memory` option, with longer retention options available (developers.openai.com), but in vLLM retention follows free-queue order and memory pressure. The vLLM design page describes eviction from the head of the free queue, so a long gap on an idle server may still hit, and a short gap on a busy one may not. Second, T5 depends on the ingress. The salt is a request field, so a gateway that drops or rewrites unknown fields can silently place every tenant in the default group; the test must run through the real ingress path, and a pass means labels A and B did not share while two requests with label A did.

Hosted-provider documentation echoes the exact-prefix rule and is useful context for application teams. One provider states that cache reuse requires the entire rendered prefix to match (platform.openai.com), another that changes at each level invalidate that level and all subsequent levels (platform.claude.com), and a third advises putting large, common content at the beginning of the prompt (ai.google.dev). Those are each vendor's own documented behavior for its own service, and they are consistent with the engine-level mechanics above without being evidence about any self-hosted deployment.

Reconciling counters with per-request evidence

Reconciliation compares two ledgers: the sum of per-request cached tokens and queried tokens in the trace, and the delta of exported counters over the same window. The rules are:

- **Use tokens, not requests.** Prefix queries and hits are token counters, so a request hit rate is a different quantity.
- **Use deltas within a stable process.** The token hit ratio is delta hits divided by delta queried tokens over a window with no restart; it is undefined when the delta of queried tokens is zero.
- **Never average ratios.** Weight by queried tokens; an unweighted mean of replica ratios is wrong.
- **Mark reset boundaries.** Counters reset to zero when the instance restarts (docs.vllm.ai); Prometheus counters can only increase or reset to zero on restart (prometheus.io), and rate functions adjust for such resets (prometheus.io). Take the rate before aggregating so resets are detectable (prometheus.io), and remember that range functions extrapolate to the ends of the window (prometheus.io), so exact reconciliation should use two direct scrapes.
- **Separate local from external.** External hits are separate metrics, and the by-source counter shows which tier supplied hit tokens.
- **Do not count retries as unique demand.** Deduplicate by an idempotency key in the trace; a retry of an identical request inflates queries and hits together.
- **Distinguish the log line from counters.** The periodic log hit rate uses a fixed recent-query window and the in-process window is cleared on reset, so use counter deltas, not the log line, for the audit.

The exported metric names may carry a counter suffix on the scrape endpoint; confirm names with a direct scrape on the pinned version before writing queries. A query of the form `increase(hits)[window]` divided by `increase(queries)[window]`, guarded by a condition that the queries delta is greater than zero, implements the

rule. Where the guard fails, record "undefined", not zero; in PromQL, series with no matching entry in the opposite vector are not part of a division result (prometheus.io), so an empty window shows up as a missing sample, not a zero.

Implementation Considerations and Process Changes

Decision order

Once classes are attributed, the action follows the largest attributed share of queried tokens. The order matters because each earlier step is cheaper and less risky than the next.

- **Preserve required semantics first.** If a changing field is functionally required at the start (for example a policy banner that must precede other text), do not move it; record the cost.
- **Stabilize representation where valid (M1, M2).** Move timestamps, request IDs and user-specific content after the shared material, which is the layout advice in the Modular handbook (handbook.modular.com) and in hosted-provider guides that say to put stable instructions and shared reference material first (platform.openai.com). Normalize serialization: unstable key ordering and structure causes misses for logically identical content (handbook.modular.com), and a hit also depends on serializing previous messages identically (handbook.modular.com). Fix document ordering where ranking is not semantically required, and keep tool definitions in a stable order, since hosted-provider guides list tool names, descriptions, schemas and ordering among the changes that alter the rendered prefix (platform.openai.com).
- **Correct configuration (M7).** Confirm caching is enabled, that requests are not asking for features that skip the lookup, and that the hash algorithm and block size are what you recorded. The Modular handbook notes that disabling prefix caching is usually not a performance optimization (handbook.modular.com), so a disabled flag in an old launch script is a plausible finding; other engines expose equivalent switches, for example SGLang's option to disable RadixAttention prefix caching (docs.sglang.ai).
- **Change retention policy (M5).** Only after M1 to M4 are excluded. Levers include free KV memory, concurrency limits and request mix. The vLLM optimization guide says preempted requests are recomputed (docs.vllm.ai), that recompute is the V1 default preemption mode, and that preemption and recomputation can adversely affect end-to-end latency. A staging test can force pressure by raising concurrent long requests (T7).
- **Pass evidence onward (M6).** If the trace shows requests landing on other instances, hand the table to the routing reference. Do not treat routing as the audit's product-selection subject.

Eviction behavior differs by engine, so record it. SGLang's RadixAttention paper describes an LRU policy that evicts the least recently used leaves (arxiv.org), marks a node evictable only when its reference counter is zero, and lets cached tokens and running requests share one memory pool. SGLang's server arguments expose the radix tree eviction policy as a setting (docs.sglang.ai).

Tenant boundaries stay intact

The audit never recommends removing salts to raise hit rate. Hosted services draw similar boundaries; one provider documents that caches are not shared across organizations (platform.openai.com). Research on timing in shared caches distinguishes sharing scopes: per-user caching has no potential privacy leakage from prompt caching (arxiv.org), while global sharing carries the highest risk (arxiv.org), and that paper concludes only per-user caching should be allowed where leakage matters (arxiv.org). A later paper describes an adaptive black-box adversary who can create arbitrary accounts (arxiv.org), and notes that per-user isolation prevents cross-tenant leakage but eliminates shared reuse (arxiv.org). Another work chose prefill time as the attack vector because the attacker can stop after the first token (arxiv.org), and lists distinct cache namespaces as a mitigation.

The practical rules for the audit are: run probes with the same labels production uses; never run hit-versus-miss timing probes across tenants on a production replica; and treat M4 rows as correct behavior. The detection method in the research uses a one-sided statistical test, because hits are expected to be faster (arxiv.org), with randomized sample order; an operator checking that salts truly separate groups in staging can borrow that design, using synthetic prompts. Because miss timing varies from request to request, a single slow or fast request proves nothing. The threat model assumes an ordinary user with no direct control over the inference server (www.ndss-symposium.org).

Validate before promoting

A change is promoted only after a before and after record is complete. Publish a blank worksheet in the runbook and fill, for each candidate change, the following columns:

- **Actual processed input:** tokens computed in prefill, from the computed-tokens histogram (raw.githubusercontent.com).
- **Cache-reused tokens:** local and external, kept separate.
- **Queue delay:** the waiting-phase histogram.
- **TTFT and inter-token latency distributions:** percentiles, not means.
- **Successful output:** completion rate and goodput, meaning requests meeting the latency objective.
- **Semantic quality:** an agreed evaluation set, because moving static text can change model behavior.
- **Tenant boundary check:** T5 results through the real ingress.
- **Rollback condition:** the previous template and configuration, and the trigger for returning to them.

Semantic quality deserves emphasis. Reordering a prompt to create a longer static prefix changes what the model reads first. A reuse gain that lowers answer quality is not a gain, and the audit treats it as a failed candidate.

Data Analysis and Evidence

This section separates three kinds of numbers: published figures from other parties with their stated setups, documented defaults, and the hypothetical arithmetic used to teach the worksheet. None of them is a GPU Smith measurement.

Published figures and their limits

The SGLang paper reports cache hit rates ranging from 50% to 99% across its benchmark suite ([arxiv.org](#)), and in a month of production use with one worker per model it observed a 52.4% hit rate for one multimodal model and 74.1% for another ([arxiv.org](#)). It attributes hits to common system messages, frequently reused example images and multi-turn histories. These figures show that hit rate is a property of workload shape, which is why this report avoids a target number. The llm-d post reports a benchmark on a cluster with 16 H100 GPUs in which the precise-scheduling configuration delivered a P90 TTFT of 0.542 seconds against more than 90 seconds for cache-blind schedulers (([llm-d.ai](#))). That is a project-reported multi-replica routing result; it addresses the confounder this audit isolates, not the single-replica attribution question.

The research literature adds one caution on sizing. KVSET estimates working sets using the Mattson stack algorithm to estimate hit rates across capacities ([arxiv.org](#)), and reports validation on production LLM traces ([arxiv.org](#)). It is a submitted preprint of September 23, 2026; this report cites it as research context, not as a tested tool, and not as an algorithm invented here.

On external caches, the LMCache documentation describes turning KV cache from temporary state into reusable state ([docs.lmcache.ai](#)), and its calculator estimates how many tokens fit in available GPU memory ([docs.lmcache.ai](#)). The Ceph report notes that bytes per token is model dependent ([www.ceph.com](#)) and states that KV caching can reduce TTFT more than tensor parallelism in its cold-start tests on its own hardware ([www.ceph.com](#)), a claim about an external tier, not about the single-replica attribution problem. SGLang's hierarchical cache builds on its radix tree ([docs.sglang.ai](#)), and LMCache documents non-prefix KV reuse ([docs.lmcache.ai](#)); both belong to tiering and capacity discussions. Those are capacity and tier questions, handled by the capacity and tiering references, and are out of scope here.

Illustrative arithmetic (Hypothetical Example)

Assume a 16-token block (verify on your system), a request of 1,200 tokens made of a 1,000-token static prefix and a 200-token user suffix. All numbers are illustrative.

- **Same request twice:** maximum hit length is capped at 1,199 tokens, so the ceiling is $\text{floor}(1199 / 16) \times 16 = 1,184$ tokens, or 98.7% of 1,200.
- **Static prefix, changing suffix:** shared tokens are 1,000, so the ceiling is $\text{floor}(1000 / 16) \times 16 = 992$ tokens (62 blocks), or 82.7%.
- **Timestamp at token 12:** the longest common initial run is 12 tokens, so $\text{floor}(12 / 16) \times 16 = 0$. Roughly 988 identical tokens after the timestamp earn nothing, because every later block's hash chains from the changed block. This is M1.
- **Timestamp moved to token 1,150:** $\text{floor}(1150 / 16) \times 16 = 1,136$ tokens (71 blocks), or 94.7%.
- **Two 400-token documents swapped after a 200-token system prompt:** the shared initial run is 200 tokens, so $\text{floor}(200 / 16) \times 16 = 192$ tokens, or 16.0%. This is M2.

Window arithmetic follows the same discipline. Suppose a counter scrape at the start of a window shows 10,000 queried tokens and 4,000 hit tokens, and a later scrape in the same process shows 22,000 and 9,400. The deltas are 12,000 queried and 5,400 hit tokens, so the token hit ratio is $5,400 / 12,000 = 45.0\%$. Now

suppose two replicas report ratios of 60% on 1,000 queried tokens and 20% on 9,000. The unweighted mean is 40%, but the weighted ratio is $(600 + 1,800) / 10,000 = 24\%$. If the queried delta had been zero, the ratio would be undefined, not zero.

The saved-prefill fraction deserves a final warning. If 992 of 1,200 prompt tokens are reused, the linear saved-prefill fraction is 82.7%, but reuse reduces prefill work only (docs.vllm.ai) and does not reduce the time of generating new tokens (docs.vllm.ai). A request dominated by a long output sees little benefit. The fraction is therefore not a throughput or GPU-count forecast, and queueing, batching and decode length all intervene.

Implications and Future Directions

For operators, the immediate implication is procedural. A hit-rate dashboard without a miss taxonomy invites the wrong spending decision, and a taxonomy without paired tests invites rationalization. The worksheet turns "hit rate is low" into a ranked list of token-level causes, each with a falsifier. A team that finds most queried tokens in M1 and M2 has a serialization project, not a hardware project. A team that finds M4 dominant has a correctly isolated multi-tenant service and should look at per-tenant prefix design instead. A team that finds M5 dominant has evidence for a retention discussion, which may eventually lead to capacity or tiering decisions, supported by working-set analysis of the kind KVSET proposes.

Several areas remain open and are flagged as unverified here. Hybrid and state-space models have documented group constraints that change the eligibility ceiling. Feature combinations can also change what a shared text prefix means for cached state. Counter semantics around preemption and reset should be re-verified at each upgrade, since the sources show the statistics code tracks preempted requests separately (raw.githubusercontent.com). The effective model and attention cache path must be confirmed for the pinned release, because documentation pages for one version can disagree with each other on model support.

For security reviewers, the audit supplies evidence about whether policy separation behaves as documented without weakening it. Any proposal to share prefix state across trust boundaries remains a separate risk decision, informed by the timing research summarized above.

Frequently Asked Questions (FAQs)

What is prefix caching in LLM inference, and what does it save?

Prefix caching stores KV blocks from processed requests and reuses them when a new request begins with the same tokens. The vLLM design documentation describes it as avoiding redundant prompt computations. It reduces prefill time, not decode time.

How does KV cache prefix reuse decide a match?

A block matches only if its tokens and every earlier block's tokens match, plus matching extra identity such as adapter and salt. Partial trailing blocks are not cached.

How is a prefix cache hit rate audited?

Capture token IDs and fingerprints per request, compute the longest common initial token count against a reference, derive the eligibility ceiling, compare with observed reuse, and assign a miss class from Table 1. Reconcile with counter deltas over a stable window.

What is the token identity cache key in vLLM?

Conceptually it is the chained hash of parent hash, block tokens and extra hashes. The algorithm defaults to sha256 on recent releases; confirm it on your pinned version.

Which eviction policy applies?

For vLLM's block pool, the least recently used free block is evicted first. SGLang describes LRU eviction of radix-tree leaves with a configurable policy setting.

Does multi-tenant prefix cache isolation cost reuse?

Yes, by design. Same-salt requests share and others do not. Research states that per-user isolation eliminates shared reuse across users, as described above.

Can timing reveal another tenant's prompt?

Research shows that when a cache is shared across users, response time depends on what has been cached, which is why salts are documented as a timing mitigation. Shared-cache designs across trust boundaries need a security decision, not a hit-rate target.

Why would a long idle gap still hit, or a short gap miss?

Retention in vLLM follows free-queue order and memory pressure, not a fixed lifetime as in hosted APIs (platform.claude.com).

Conclusion

Low prefix reuse is a diagnosis problem before it is a capacity problem. The documented mechanics show that identity is token identity chained across blocks, eligibility is block-aligned, tenant labels deliberately split the cache, and counters are token-weighted with defined reset behavior. From those, this report builds a worksheet with eight miss classes plus an explicit unknown, controlled paired tests that can falsify each, and a reconciliation procedure that avoids averaged ratios and zero denominators.

The decision order is to preserve required semantics, stabilize representation where valid, correct configuration, change retention policy, and only then pass evidence to routing and capacity references. Tenant boundaries stay intact throughout: rows classified as deliberate policy separation are treated as correct behavior, and no step in the method asks for reuse across a trust boundary. Teams that act on the table in order will usually find that the cheapest causes, unstable serialization and early variable fields, are exhausted before a retention or capacity discussion is warranted. No benchmark was executed for this report; the arithmetic is illustrative and the sources are documentation, source comments and published research, each dated or versioned where the source permits. Operators should pin their versions, record the block size, hash

algorithm and tokenizer revision, re-read the matching documentation, run the paired tests on a staging replica with approved synthetic prompts, and promote changes only after latency, quality, tenant-boundary and rollback checks are recorded.

Source links

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

1. https://raw.githubusercontent.com/vllm-project/vllm/main/vllm/v1/core/kv_cache_manager.py
2. https://prometheus.io/docs/concepts/metric_types
3. https://raw.githubusercontent.com/vllm-project/vllm/main/vllm/v1/core/kv_cache_utils.py
4. <https://arxiv.org/html/2502.07776v2>
5. <https://arxiv.org/html/2508.08438v2>
6. <https://handbook.modular.com/inference-optimization/prefix-caching>
7. <https://llm-d.ai/blog/kvcache-wins-you-can-see>
8. <https://arxiv.org/abs/2609.27746>
9. <https://www.ceph.com/en/news/blog/2025/vllm-kv-caching>
10. <https://pypi.org/project/vllm>
11. https://docs.vllm.ai/en/latest/design/prefix_caching
12. https://huggingface.co/docs/transformers/chat_templating
13. <https://huggingface.co/docs/tokenizers/components>
14. https://huggingface.co/docs/transformers/tokenizer_summary
15. <https://github.com/google/sentencepiece>
16. <https://arxiv.org/abs/2506.14123>
17. https://docs.vllm.ai/en/latest/configuration/engine_args
18. <https://arxiv.org/pdf/2309.06180>
19. https://docs.vllm.ai/en/latest/design/hybrid_kv_cache_manager
20. https://docs.vllm.ai/en/latest/features/automatic_prefix_caching
21. <https://docs.vllm.ai/en/latest/usage/metrics>
22. <https://raw.githubusercontent.com/vllm-project/vllm/main/vllm/v1/metrics/loggers.py>
23. <https://docs.vllm.ai/en/latest/design/metrics>
24. <https://docs.vllm.ai/en/latest/cli/serve>
25. <https://raw.githubusercontent.com/vllm-project/vllm/main/vllm/v1/metrics/stats.py>
26. <https://raw.githubusercontent.com/vllm-project/vllm/main/vllm/v1/core/sched/scheduler.py>
27. https://docs.vllm.ai/en/latest/serving/online_serving
28. https://raw.githubusercontent.com/vllm-project/vllm/main/vllm/v1/core/block_pool.py
29. <https://platform.claude.com/docs/en/build-with-claude/prompt-caching>
30. <https://developers.openai.com/api/docs/guides/prompt-caching>

31. <https://platform.openai.com/docs/guides/prompt-caching>
32. <https://ai.google.dev/gemini-api/docs/caching>
33. <https://prometheus.io/docs/prometheus/latest/querying/functions>
34. <https://prometheus.io/docs/prometheus/latest/querying/operators>
35. https://docs.sglang.ai/advanced_features/server_arguments.html
36. <https://docs.vllm.ai/en/latest/configuration/optimization>
37. <https://arxiv.org/pdf/2312.07104>
38. <https://arxiv.org/html/2411.18191v2>
39. <https://www.ndss-symposium.org/wp-content/uploads/2025-1772-paper.pdf>
40. <https://docs.lmcache.ai>
41. https://docs.lmcache.ai/getting_started/kv_cache_calculator.html
42. https://docs.sglang.ai/advanced_features/hicache_design.html

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.