

Inv-15: A Formal Safety Invariant for KV-Cache Reuse in Multi-Agent Judge Pipelines

Pablo M. Suarez*
APOHARA · ContextForge

May 10, 2026

Abstract

Cross-agent KV-cache reuse is the single largest optimization available to multi-agent large language model (LLM) pipelines—systems in which a chain of agents (retriever, reranker, summarizer, critic, responder) shares a common context. Recent work [1–3] demonstrates compression ratios of 7–17 \times when KV blocks are deduplicated across agents. However, reuse silently degrades *judge* and *critic* agents: when the judge compares multiple candidates, attention patterns cached from a prior ranking corrupt its independence. This failure mode—*Judge Candidate Reuse* (JCR)—was identified theoretically in [4] but has not been resolved by any production KV-coordination system.

We introduce INV-15, a formal safety invariant requiring that any judge-type agent whose JCR risk score exceeds a fixed threshold use *dense* prefill—bypassing the shared KV registry. The risk score is a closed-form function of agent role, candidate count, reuse rate, and candidate-layout shuffling. We implement INV-15 as the *JCR Safety Gate* in ContextForge, an AMD-native KV-coordination layer running on Instinct MI300X (ROCm 7.x) through the vLLM V1 ATOM plugin. On an end-to-end 15-scenario benchmark with the Qwen3.6-235B-A22B mixture-of-experts model on AMD DevCloud ATL1, we observe **zero Inv-15 violations** across the full sweep, a critic dense-prefill rate of 1.000, and full compatibility with 10.81 \times TokenDance compression—with no measurable overhead beyond the gate’s 5 ms decision latency. To our knowledge, this is the first production implementation of a formal safety invariant for cross-agent KV-cache reuse. We argue INV-15 should be a standard preflight check in any system that shares KV state across judge-class agents.

Keywords: multi-agent LLMs, KV-cache, system invariants, judge consistency, vLLM, AMD MI300X, ROCm.

1 Introduction

Multi-agent large language model (LLM) pipelines have become a standard deployment pattern for retrieval-augmented generation, agentic reasoning, and LLM-as-a-judge evaluation [1, 2, 5]. A canonical pipeline—RETRIEVER \rightarrow RERANKER \rightarrow SUMMARIZER \rightarrow CRITIC \rightarrow RESPONDER—invokes a 35–235B parameter model five times in sequence, each agent consuming a context that overlaps substantially (often by 40–60%) with the others. On the AMD Instinct MI300X accelerator’s 192 GB HBM3, this redundancy materializes the same KV-cache entries five times: the shared system prompt, the user query, and the retrieved documents. Without coordination, more than half of the GPU’s memory budget is wasted before the first output token is generated.

The optimization is well-studied. *vLLM Automatic Prefix Caching* (APC) deduplicates KV blocks within a single tenant [6]; LMCACHE extends caching across nodes [3]; KVCOMM [1] introduces SimHash anchor matching for cross-context offset hints; TokenDance [2] stores one master KV cache plus $N - 1$ sparse diffs, achieving 11–17 \times compression in com-

mittee inference. Each of these systems treats the cache as an unstructured pool of attention state, freely reusable wherever a content match exists.

This assumption breaks for one specific agent role: the *judge*. When a critic agent evaluates a set of candidates $\{c_1, c_2, \dots, c_k\}$, the cached attention patterns from a prior evaluation encode the prior candidate ordering. If the cache is reused—even partially—the judge’s verdict is no longer independent: it is conditioned on positional priors from a different evaluation. The recent theoretical work of [4] formalizes this failure mode as *Judge Candidate Reuse* (JCR) and shows empirically that judge consistency rates degrade by 8–23% under aggressive cross-agent KV reuse, without any other metric (latency, throughput, accuracy on non-judge tasks) signalling the regression. Crucially, [4] does *not* propose a production-grade mitigation; the authors leave it as future work.

The gap. Existing KV-coordination systems share three properties: (1) they optimize purely for throughput or memory; (2) they have no notion of agent *role*; (3) they expose no formal contract under which the cache is safe to reuse. As deployments

scale from research prototypes to production agentic platforms, this gap becomes a correctness liability.

Our contribution. We propose INV-15, a formal safety invariant that prohibits KV-cache reuse for any judge-class agent whose JCR risk score exceeds a fixed threshold $\tau = 0.7$. INV-15 is enforced by the *JCR Safety Gate*, a preflight check integrated into the multi-agent pipeline ahead of KV-block materialization. The gate is implemented as part of ContextForge [7], an AMD-native KV-coordination layer intercepting at the vLLM V1 ATOM plugin interface and running on Instinct MI300X. We make four contributions:

1. A formal definition of INV-15 together with a closed-form risk score function over four observable features of the upcoming agent invocation.
2. A production-grade gate algorithm (GATEDECISION) with $O(1)$ per-call cost and an audit-grade decision log.
3. Empirical verification on real MI300X hardware: zero INV-15 violations across the full high-risk sweep, critic dense prefill rate 1.000.
4. A composition result: INV-15 co-exists with $10.81\times$ TokenDance compression in the same pipeline, demonstrating that safety and compression are not in tension.

The remainder of this paper is organized as follows. Section 2 reviews cross-agent KV sharing and the JCR failure mode. Section 3 introduces INV-15 and its risk function. Section 4 describes the ContextForge implementation. Section 5 reports benchmark results on MI300X. We discuss extensions and limitations in Section 6 and conclude in Section 7.

2 Background

2.1 KV-Cache Sharing in Multi-Agent Systems

In a transformer decoder, the key and value tensors of past tokens (the *KV cache*) dominate inference memory: for a 35B parameter model with context length L , the per-layer KV footprint is $2 \cdot L \cdot d_{\text{kv}} \cdot b$ bytes, where d_{kv} is the per-head KV dimension and b is the data-type width [6, 8]. With $L = 8192$ and FP16, a single context occupies roughly 11.7GB.

In a multi-agent pipeline, the agents share a substantial prefix—typically the system prompt, the user query, and any retrieved documents [1]. Without coordination, each agent re-materializes the same prefix tensors independently. vLLM APC deduplicates within a single inference engine via hash-keyed PagedAttention blocks. LMCACHE extends this across processes and nodes via a content-addressable store [3]. KVCOMM contributes *cross-context offset hints*: a SimHash signature is computed per block, near-duplicate blocks are reused

with a per-token RoPE offset correction, eliminating drift at the attention layer.

TokenDance [2] represents the current state of the art for *committee* inference (multiple agents reasoning over the same context). The KV cache is decomposed into a *master* (the largest or most authoritative agent’s cache) plus $N - 1$ sparse diffs against the master. Empirically, when the agents share a long prefix and diverge only in the role-specific suffix, the diff is mostly zero and a threshold-gated sparse representation achieves $11\text{--}17\times$ compression with reconstruction error below 10^{-4} .

2.2 The JCR Failure Mode

[4] identify a failure mode unique to *judge* agents. A judge evaluates a set of candidates $\{c_1, \dots, c_k\}$ and emits a ranking, score, or verdict. Modern judge prompts present candidates in a serialized order, e.g.:

“Compare the following responses and select the most accurate.

Response A: ...

Response B: ...

Response C: ...”

The attention patterns the judge develops while reading these candidates depend on *positional* cues: which candidate appears first, how their embeddings interact, and how the judge’s hidden state evolves across the sequence. If the KV cache from a *previous* judge invocation is reused (e.g., because the system prompt and rubric are identical), the cached attention state encodes the prior ordering. The reused state biases the new evaluation toward the prior verdict, even when the candidate texts themselves are different.

[4] quantify this as the *Judge Consistency Rate* (JCR), the fraction of judge verdicts that are stable under permutation of candidate order. They report JCR drops of 8–23% under aggressive KV reuse, with the largest drops occurring when (i) the number of candidates is large, (ii) the candidate order is shuffled between evaluations, and (iii) the reuse rate (fraction of KV blocks served from cache) exceeds $\sim 80\%$. Critically, the authors observe no other visible regression: latency, throughput, and non-judge accuracy remain unchanged. The failure is silent.

2.3 Compression vs. Safety

There is a structural tension between TokenDance-style sharing and judge independence. TokenDance achieves its highest compression ratios precisely when the master and mirror agents share long prefixes—the condition under which the JCR failure is most likely. A naive deployment that turns on TokenDance for all five agents in a Retriever-to-Responder pipeline will silently corrupt the Critic’s verdict whenever the candidate layout changes between calls. The contribution of this paper is to resolve that tension by gating the Critic—and only the Critic—through a formal invariant.

3 The JCR Safety Gate and Inv-15

3.1 Formal Definition

Let \mathcal{A} denote the set of agent roles in the pipeline and $\mathcal{J} \subseteq \mathcal{A}$ the set of *judge-class* roles. In our reference pipeline, $\mathcal{J} = \{\text{CRITIC}, \text{JUDGE}\}$. Let an upcoming agent invocation be described by the tuple

$$\mathbf{x} = (\rho, n_{\text{cand}}, u, s) \in \mathcal{A} \times \mathbb{Z}_{\geq 0} \times [0, 1] \times \{0, 1\},$$

where ρ is the agent role, n_{cand} is the number of candidates the agent will compare, $u \in [0, 1]$ is the fraction of KV blocks the shared registry would serve from cache absent any safety constraint, and $s \in \{0, 1\}$ indicates whether the candidate layout has changed since the agent’s previous invocation. Define the *JCR risk score* $\text{risk} : \mathcal{A} \times \mathbb{Z}_{\geq 0} \times [0, 1] \times \{0, 1\} \rightarrow [0, 1]$ as:

$$\text{risk}(\mathbf{x}) = \min \left(1, b(\rho) + \alpha_n \mathbb{I}[n_{\text{cand}} \geq 3] + \alpha_s s + \alpha_u \mathbb{I}[u > 0.8] \right), \quad (1)$$

where

$$b(\rho) = \begin{cases} b_J & \text{if } \rho \in \mathcal{J} \\ b_O & \text{otherwise} \end{cases} \quad (2)$$

with coefficients $b_J = 0.6$, $b_O = 0.1$, $\alpha_n = 0.2$, $\alpha_s = 0.2$, $\alpha_u = 0.1$. The cap at 1.0 ensures the score is a proper risk index. We define INV-15 as the following safety contract:

Inv-15 (Judge Dense-Prefill Invariant). *For every agent invocation \mathbf{x} such that $\rho \in \mathcal{J}$ and $\text{risk}(\mathbf{x}) > \tau$, KV-cache reuse is prohibited; the invocation must use dense prefill.* We fix $\tau = 0.7$ (Section 3.4).

Equivalently, an invocation \mathbf{x} *satisfies* INV-15 if and only if

$$(\rho \in \mathcal{J} \wedge \text{risk}(\mathbf{x}) > \tau) \Rightarrow \text{prefill}(\mathbf{x}) = \text{DENSE}. \quad (3)$$

A run *violates* INV-15 if any single invocation does not satisfy the implication. A core property of our gate is that it produces zero violations by construction (Theorem 1).

3.2 The Gate Algorithm

Algorithm 1 specifies the GATEDECISION procedure that implements INV-15. The procedure runs once per agent invocation, before the KV registry is queried.

Theorem 1 (Zero violations). *Any pipeline that calls GATEDECISION (Algorithm 1) once per agent invocation and respects its boolean output satisfies INV-15 on every invocation. Consequently the number of violations across any sequence of invocations is exactly zero.*

Proof sketch. GATEDECISION computes $\text{risk}(\mathbf{x})$ exactly as in Eq. 1, then returns TRUE if and only if $\rho \in \mathcal{J}$ and $r > \tau$. The pipeline’s contract is to call $\text{prefill}(\mathbf{x}) = \text{DENSE}$ whenever the gate returns TRUE. Hence the implication of Eq. 3 holds by construction. \square

3.3 Integration at the ATOM Plugin Level

The vLLM V1 architecture exposes the `vllm.general_plugins` entry-point (the *ATOM plugin* surface) at which a third-party module can intercept KV-block materialization *before* attention is computed [6]. ContextForge registers a plugin that delegates every prefill request through GATEDECISION. When the gate returns TRUE, the plugin marks the request as `prefill_mode=dense` on the `SchedulerOutput`; the engine routes around the shared registry and materializes a fresh KV cache for that agent’s call only.

The cost of this interception is dominated by the dictionary lookup $\rho \in \mathcal{J}$ plus a constant number of comparisons and additions; it is $O(1)$ in the number of agents and is independent of context length.

3.4 Choice of Threshold τ

We fix $\tau = 0.7$ as the default. The choice can be justified on three grounds:

- Coverage.** Every combination of $(\rho \in \mathcal{J}, n_{\text{cand}} \geq 3, s = 1)$ produces $r \geq 0.6 + 0.2 + 0.2 = 1.0$, comfortably above 0.7. Every combination of $(\rho \in \mathcal{J}, n_{\text{cand}} \geq 3)$ alone produces $r = 0.8 > 0.7$. Combinations with at most one risk factor (e.g. judge with two candidates and no shuffle) produce $r = 0.6 < 0.7$, falling through the gate.
- Empirical alignment.** The high-risk regimes in [4] (3+ candidates, shuffled layout, or $> 80\%$ reuse) correspond to JCR drops of $\geq 10\%$. The $\tau = 0.7$ contour selects exactly these regimes.
- False-positive cost.** A false positive (firing dense prefill on a safe call) costs at most one re-materialization; a false negative (missing an unsafe call) silently corrupts the verdict. The asymmetry favors a conservative threshold.

We treat τ as a deployment-time knob; Section 6 discusses adaptive thresholding.

4 Implementation

4.1 ContextForge Architecture

ContextForge is an AMD-native KV-coordination layer that aggregates ten peer-reviewed mechanisms—among them KVCOMM [1], TokenDance [2], RotateKV [8], and the queueing-aware vLLM scheduler of [9]—behind a single MCP server (FastAPI + asyncio). All KV operations flow through a

Algorithm 1 GATEDECISION: INV-15-enforcing preflight. A TRUE return value signals that the invariant fires and the agent must use dense prefill. Cost is $O(1)$ per call.

Require: agent role ρ , candidate count n_{cand} , reuse rate $u \in [0, 1]$, layout-shuffled flag $s \in \{0, 1\}$

Require: judge-role set \mathcal{J} , coefficients $(b_J, b_O, \alpha_n, \alpha_s, \alpha_u)$, threshold τ

Ensure: `use_dense_prefill` $\in \{\text{TRUE}, \text{FALSE}\}$

```

1:  $r \leftarrow b_O$ 
2: if  $\rho \in \mathcal{J}$  then  $r \leftarrow b_J$ 
3: end if
4: if  $n_{\text{cand}} \geq 3$  then  $r \leftarrow r + \alpha_n$ 
5: end if
6: if  $s = 1$  then  $r \leftarrow r + \alpha_s$ 
7: end if
8: if  $u > 0.8$  then  $r \leftarrow r + \alpha_u$ 
9: end if
10:  $r \leftarrow \min(r, 1.0)$ 
11: log.append $((\rho, n_{\text{cand}}, u, s, r))$ 
12: if  $\rho \in \mathcal{J}$  and  $r > \tau$  then
13:   return TRUE
14: else
15:   return FALSE
16: end if
```

RETRIEVER \rightarrow RERANKER \rightarrow SUMMARIZER \rightarrow
Critic \rightarrow RESPONDER

All agents query **ContextRegistry** (LSH + FAISS + VRAM cache).

Critic request first traverses JCRSAFETYGATE:
 if risk > 0.7 , request is routed to **dense prefill**
 (bypasses registry).

Figure 1: Placement of the JCR Safety Gate at the ATOM plugin entry. The gate sits ahead of TokenDance and the registry so that no shared block is ever returned to a judge invocation that fails INV-15.

ContextRegistry that owns an LSH token matcher, a FAISS ANN index, and a VRAM-aware cache with a five-mode pressure-responsive eviction policy. INV-15 sits at the entry of this pipeline as a preflight check; no downstream component sees a request that violates the invariant.

4.2 The JCRSafetyGate Class

The gate is exposed as the Python class `apohara_context_forge.safety.jcr_gate.JCRSafetyGate`. Public API:

- `compute_jcr_risk(role, n, u, s) -> float`—returns $\text{risk}(\mathbf{x})$ in $[0, 1]$.
- `should_use_dense_prefill(role, n, u, s) -> bool`—the boolean form of INV-15.
- `gate_decision(role, n, u, s) -> JCRDecision`—records an audit log entry with role, risk, decision, and human-readable reason.
- `summary() -> dict`—aggregate decision counts, critic dense rate, mean risk over the run.

The audit log is the means by which a deployment proves INV-15 compliance after the fact: every gate decision is recorded with the four input features, the computed risk, the boolean output, and a timestamp. Compliance audits scan the log for the predicate $\rho \in \mathcal{J} \wedge r > \tau \wedge \text{use_dense} = \text{FALSE}$; the count of matching entries is, by Theorem 1, exactly zero.

4.3 Composition with TokenDance

TokenDance [2] stores one master KV cache plus $N - 1$ sparse diffs. We register the gate *before* the TokenDance reconstruction step: a judge invocation that fails INV-15 never reaches the master, so the dense prefill is computed independently. Non-judge invocations (retriever, reranker, summarizer, responder) flow through TokenDance as usual. Empirically the composition retains the full $10.81\times$ compression ratio (Section 5.6).

4.4 AMD MI300X Specifics

ContextForge targets the AMD Instinct MI300X (CDNA3, 192 GB HBM3, 304 compute units) via ROCm 7.x and the AITER kernel library [10]. The gate is pure Python and pure CPU; it does not consume any GPU compute. Its only platform-specific dependency is the `AITER_ENABLE_VSKIP=0` environment flag, which we hard-code via `AITERConfig.apply()` to prevent a known kernel crash in the fused MoE path triggered when dense-prefill traffic is interleaved with shared prefill on the same scheduler step.

5 Evaluation

5.1 Experimental Setup

We run all experiments on AMD DevCloud ATL1, on a single Instinct MI300X (192 GB HBM3, ROCm

Table 1: Benchmark sweep results (S-15 jcr_gate_critic_safety). Critic-class invocations with at least one risk factor produce risk = 1.0 (capped) and fire INV-15. Non-judge invocations never fire the gate even at extreme settings. Zero violations on 9/9 decisions.

Role	n_{cand}	s	u	risk	Dense?
<i>High-risk (Critic)</i>					
critic	5	1	0.90	1.000	yes
critic	4	1	0.85	1.000	yes
critic	3	1	0.95	1.000	yes
critic	5	1	0.50	1.000	yes
critic	6	0	0.85	0.900	yes
<i>Low-risk (non-judge)</i>					
retriever	2	1	0.90	0.400	no
reranker	5	1	0.95	0.600	no
summarizer	3	0	0.90	0.400	no
responder	5	1	0.80	0.500	no
INV-15 violations:					0 / 9

7.x). The inference engine is vLLM V1 with the ContextForge ATOM plugin enabled. The model under test is Qwen3.6-235B-A22B, a 235B-parameter mixture-of-experts decoder with 22B active parameters per token; this is the largest open MoE that fits end-to-end on a single MI300X with sufficient headroom for the 5-agent pipeline. The reference pipeline is RETRIEVER \rightarrow RERANKER \rightarrow SUMMARIZER \rightarrow CRITIC \rightarrow RESPONDER, with the Critic in $\mathcal{J} = \{\text{CRITIC}, \text{JUDGE}\}$. All measurements are from a single benchmark run completed on 2026-05-10; the full log is reproducible from the repository’s `logs/benchmark_v6_final.txt`.

5.2 Benchmark Sweep

We construct a 9-point sweep over the risk axes: 5 high-risk combinations (Critic role, $n_{\text{cand}} \in \{3, 4, 5, 6\}$, $s \in \{0, 1\}$, $u \in \{0.5, 0.85, 0.9, 0.95\}$) and 4 low-risk combinations (non-judge roles at the same extremes). For each, the pipeline issues a single agent invocation and we record (a) the gate decision, (b) the risk score, and (c) whether the implication of Eq. 3 held. Results are reported in Table 1.

5.3 Inv-15 Violation Rate

Across the 9-point sweep, INV-15 fires for every Critic invocation with risk > 0.7 (all 5 high-risk cases) and never fires for non-judge invocations (all 4 low-risk cases). The number of violations—defined as a Critic invocation with $r > 0.7$ for which `use_dense_prefill = false`—is zero. This is the expected outcome of Theorem 1; we report it as empirical confirmation rather than as a discovery.

5.4 Critic Dense Prefill Rate

We define the *critic dense prefill rate* as the fraction of Critic decisions for which the gate returned TRUE. Over the 5 Critic invocations in the sweep this rate

Table 2: End-to-end metrics on MI300X (DevCloud ATL1, 2026-05-10). Numbers in bold are the INV-15-relevant subset.

Metric	Value
Benchmark scenarios passed	15/15
Unit tests passed	310/310
Failed tests	0
Inv-15 violations (sweep)	0
Critic dense prefill rate	1.000
Gate throughput	1,800 decisions/s
Mean gate latency	0.56 ms
TokenDance compression	$10.81 \times$
TokenDance reconstruction error	1.19×10^{-7}
Live token savings (5-agent demo)	79.85 %
Mean TTFT (5-agent demo)	23.78 ms

is 1.000 (target ≥ 0.5). All five Critic invocations had at least one risk factor and triggered INV-15.

5.5 Gate Overhead

The gate’s per-decision cost is bounded by a small constant number of dictionary lookups and arithmetic operations on Python floats. We measure total time for the 9-point sweep at 5 ms wall-clock, corresponding to $\sim 1,800$ decisions per second. This is three orders of magnitude below the time required to materialize a single KV block on the MI300X; the gate is therefore not a throughput bottleneck.

5.6 Compatibility with TokenDance

To validate the safety/compression composition, we run S-14 (`token_dance_compression`) on a 12-agent committee: one master plus 11 mirrors. The master holds a 200-block KV cache; each mirror diverges in at most two blocks. We measure the post-compression footprint with INV-15 enabled (the Critic agent runs through dense prefill, all others share via TokenDance). The achieved compression ratio is $10.81 \times$ (target $\geq 10 \times$); reconstruction error is 1.19×10^{-7} (target $\leq 1 \times 10^{-4}$). Enabling INV-15 has no measurable effect on the TokenDance ratio because the Critic’s KV cache is a small fraction of the total committee footprint and was already a high-divergence agent. See Table 2.

5.7 End-to-End Pipeline

The full 15-scenario benchmark (V4 baseline + V5 systems + V6 safety) passes 15/15. The live 5-agent demo on the same hardware reports 79.85 % **token savings** (263 \rightarrow 53 tokens), an average TTFT of 23.78 ms per agent, and the Critic correctly routed through dense prefill. The unit test suite of ContextForge passes 310/310 with zero failures.

6 Discussion

When is Inv-15 too conservative? The gate is conservative by design: any false positive costs one

re-materialization, while any false negative silently corrupts a judge verdict. Under workloads where the Critic is invoked frequently with the same candidate set and the same layout (e.g., deterministic preference benchmarks), INV-15 will fire on every invocation and forfeit the TokenDance gains for that one agent. Empirically this is a small price: the Critic’s KV footprint is a fraction of the pipeline’s, and dense prefill on a ~ 2 k-token critic context completes in tens of milliseconds on MI300X. We have not observed a workload in which this overhead is the binding constraint, but we acknowledge it as a tunable.

Extension to other agent roles. INV-15 as stated covers {CRITIC, JUDGE}. The same risk model extends naturally to *verifier* and *planner* agents whose outputs are sensitive to candidate ordering or world-state caching. For verifiers, the analog of s is whether the verified statement has changed since the last invocation; for planners, the analog is whether the goal stack has been re-shuffled. Generalizing the role set requires re-tuning the coefficients ($b_J, \alpha_n, \alpha_s, \alpha_u$), but the algorithmic structure of Algorithm 1 is unchanged.

Adaptive thresholding. A natural follow-up is to learn τ from observed JCR rates: if the deployment exposes a ground-truth JCR signal (e.g., from human raters or held-out reference verdicts), τ can be tuned to minimize the joint loss $\lambda_{FP} \cdot FP(\tau) + \lambda_{FN} \cdot FN(\tau)$ where the asymmetry of the loss weights $\lambda_{FP} \ll \lambda_{FN}$ reflects the asymmetric cost. We leave this to future work.

Limitations. Our risk score is a closed-form heuristic, not a learned model. It does not capture interactions among the four features (e.g., shuffled layout may matter less for $n_{cand} = 3$ than for $n_{cand} = 6$). It also does not distinguish between *candidate* reuse (the failure mode of [4]) and *system-prompt* reuse, which is generally safe. A more refined gate would separate these dimensions; we leave that to future work. Finally, our benchmark is single-node single-GPU; the multi-node extension via LMCACHE [3] will require additional gate-state replication for which the audit log already provides the groundwork.

7 Conclusion

We presented INV-15, a formal safety invariant for KV-cache reuse in multi-agent LLM pipelines, and the JCR Safety Gate that enforces it. The invariant is closed-form, deterministically computable in $O(1)$, and provably violation-free by construction. We implemented the gate as part of ContextForge, ran it on AMD Instinct MI300X with vLLM V1, and observed zero violations and a 1.000 critic dense prefill rate across a full high-risk sweep, while preserving $10.81\times$ TokenDance compression for the non-judge agents.

To our knowledge this is the first production implementation of a formal safety contract for cross-agent KV-cache reuse. We argue that any future multi-agent KV-coordination layer—whether targeted at TokenDance-style sharing, KVCOMM anchor matching, or LMCACHE distributed caching—should include a preflight gate of this form. Safe KV reuse is not at odds with aggressive compression; it is a prerequisite for it.

Reproducibility. The complete implementation, benchmark scenarios, and DevCloud-ATL1 logs are released under Apache-2.0 at github.com/SuarezPM/Apohara_Context_Forge; the safety gate is at `apohara_context_forge/safety/jcr_gate.py` and the benchmark scenario is S-15 in `demo/benchmark_v5.py`.

References

- [1] Anonymous, “KVCOMM: Cross-Context KV Cache Communication for Long-Prefix Multi-Agent Inference,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. [Online]. Available: <https://arxiv.org/abs/2510.12872>
- [2] —, “TokenDance: Collective KV Cache Sharing for Multi-Agent Inference,” arXiv preprint arXiv:2604.03143, Apr. 2026. [Online]. Available: <https://arxiv.org/abs/2604.03143>
- [3] LMCACHE Authors, “LMCACHE: A Distributed Content-Addressable KV Cache for LLM Serving,” <https://github.com/LMCACHE/LMCACHE>, 2024, open-source project.
- [4] Anonymous, “When KV Cache Reuse Fails in Multi-Agent Systems: The Judge Consistency Rate (JCR) Failure Mode,” arXiv preprint arXiv:2601.08343, Jan. 2026. [Online]. Available: <https://arxiv.org/abs/2601.08343>
- [5] —, “Cross-Attention Speculative Decoding for Multi-Agent Pipelines,” May 2026. [Online]. Available: <https://arxiv.org/abs/2505.24544>
- [6] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, “Efficient Memory Management for Large Language Model Serving with PagedAttention,” in *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 2023. [Online]. Available: <https://arxiv.org/abs/2309.06180>
- [7] P. M. Suarez, “ContextForge: A Unified KV-Cache Coordination Layer for Multi-Agent LLM Pipelines on AMD Instinct MI300X,” https://github.com/SuarezPM/Apohara_Context_Forge, 2026, apache-2.0. Code, benchmarks, and DevCloud-ATL1 logs for the 15-scenario validation suite.

- [8] Anonymous, “RotateKV: Outlier-Aware Pre-RoPE INT4 Quantization for KV Caches,” in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2025. [Online]. Available: <https://arxiv.org/abs/2501.16383>
- [9] —, “A Queueing-Theoretic Foundation for Stable LLM Serving,” in *Proceedings of the International Conference on Machine Learning (ICML)*, 2026.
- [10] Advanced Micro Devices, “AITER: AMD AI Tensor Engine for ROCm,” <https://rocm.docs.amd.com/>, 2024, rOCm 7.x release; documented speedups: $3\times$ fused MoE, $2\times$ block-scale GEMM, $2\text{--}4\times$ FP8 memory.