

Closing the Unowned Trust Boundary: Runtime KV-Cache Integrity Verification against Safety-Bypass Tampering

Tianyu Lu · Independent Researcher *Global South AI Safety Hackathon — Asia / Technical Safety track*

Abstract

The KV cache is the dominant stateful component of LLM inference, and safety-critical signals propagate from aligned weights into its mutable value vectors at every decoding step — yet no production serving framework verifies the integrity of cached values. Prior work (SCALPEL) [15] exploited this gap to disable a model’s trained refusal behavior by erasing safety directions from the cache, while keeping every deployed integrity signal — weight hashes, forward-pass hooks, input/output screening, behavioral spot-checks — invariant. That work was framed as a systems exploit; here we reframe it as what it actually is: an **integrity gap in runtime model state that the entire compliance-verification stack is blind to, and that no layer of the inference stack claims ownership of**. We then close it. We build a **sampled runtime KV-cache integrity check** that, at each decoding step, recomputes value vectors from the model’s own weights ($v^* = W_V \cdot x_t$) for a random subset of (layer, head, token) and compares them to the cache. The attacker’s defining stealth property — a bit-identical forward pass — is exactly what makes the recomputation a reliable ground truth. The detector achieves **AUC 0.9996, empirical FPR = 0** (robust to distribution shift and batching up to 16), and a **sequence-level detection probability of 1.0 even at a 1% per-step sampling rate**, at a measured cost of **+5.9% decode latency** ($\approx 99\%$ of which is dispatch overhead, eliminable by kernel fusion to sub-1%). A game-theoretic analysis shows the attacker has no escape: evading per-step detection by erasing fewer heads collapses attack success from 65% to 5% while remaining caught at the sequence level.

1. Introduction

Modern LLM serving is built around the key–value (KV) cache: a per-request runtime buffer that stores intermediate Key/Value projections so the model need not recompute attention over the full context at each step. Frameworks treat this buffer as a transparent performance optimization — allocating, evicting, quantizing, and sharing it with no integrity contract.

But the KV cache is not merely a performance artifact. The model’s learned refusal behavior is mediated by a low-dimensional direction in representation space [1], and that direction propagates into cached **value** vectors through the value projection ($v = W_V \cdot x$). Safety-critical state therefore migrates from the protected, attested weight space into a mutable, operator-controlled runtime buffer — and **the security perimeter around LLM deployments protects every component except this one**. Weight checksums, input filters, forward-pass monitoring, and output classifiers all leave the KV cache unguarded.

This work makes three contributions:

1. **A repositioning (Section 4)**. The cache-level safety-bypass attack is not a remote exploit and not a single-framework bug. Its true significance is that it defeats the integrity-attestation mechanisms on which deployment *compliance* relies, and that **no layer of the inference stack — model vendor or serving framework — claims ownership of cache-content integrity**. It is an unowned trust boundary.
2. **A defense (Sections 5–6)**. A sampled runtime KV-cache integrity check — recompute value vectors from the model’s weights and compare to the cache — that is the missing attestation layer for this boundary.
3. **A complete characterization (Section 7)**. Detection (TPR 99.6%, FPR 0, AUC 0.9996), robustness (survives distribution shift, batching), cost (+5.9% at the saturation operating point), and a game-theoretic argument that detection evasion forfeits the attack.

2. Background

Refusal directions. Safety-aligned models are trained to refuse harmful requests, and this learned refusal has a strikingly simple geometry: it is mediated by a small number of directions in the residual stream, extractable via contrastive activation analysis over harmful vs. harmless prompts [1]. Crucially, this direction is present in the **value** projection output that gets written to the cache.

The KV cache as a trust boundary. For an L -layer, H -head model, the cache stores per-head Key/Value tensors for every past token. These are continuously evicted, quantized, and shared with no safety consideration. The security-relevant observation: the refusal direction enters the cache through $v = W_V \cdot x$, so the buffer carries the model’s intent to refuse — in a region no deployed verification mechanism inspects.

Threat landscape. Recent work has examined KV-cache *privacy* (prompt reconstruction from cached tensors) [10] and KV-cache *manipulation* attacks (trojans and narrative steering via cached state) [11, and related]. KV-cache-level *defenses* to date target either confidentiality or input-token relevance — e.g. RobustKV [5] defends against jailbreaks by evicting low-importance prompt tokens. **None verify the integrity of cached value content** — the surface this work concerns.

3. The Attack: What It Exposes

This section summarizes the cache-level safety-bypass attack (SCALPEL) [15] at the level needed to motivate the defense. *We describe the mechanism, not a reproduction recipe; attack code is withheld pending available defenses, consistent with the original responsible-disclosure posture.*

3.1 Mechanism

The attack operates in three stages on an open-weight model the operator controls:

- **Scan.** Extract per-layer refusal directions from a small set of contrastive pairs, and map which attention heads carry the most safety signal (an *Energy/Selectivity* decomposition).
- **Plan.** Select target heads and assign a per-head erasure strength.
- **Strike.** At every decoding step, project the safety direction out of the newly cached value vectors — a rank-1 operation. Model weights are never modified; the forward pass executes its original code path.

3.2 Two structural facts that matter for the defense

The iceberg structure. Safety is not concentrated in a few heads. A small “tip” of high-selectivity heads in the final layers carries a large share of refusal energy, but a distributed base across *all* layers carries the rest. Erasing only the tip is insufficient — the distributed base sustains refusal. *This is why an effective attack must touch many heads, a fact the defense exploits in Section 7.*

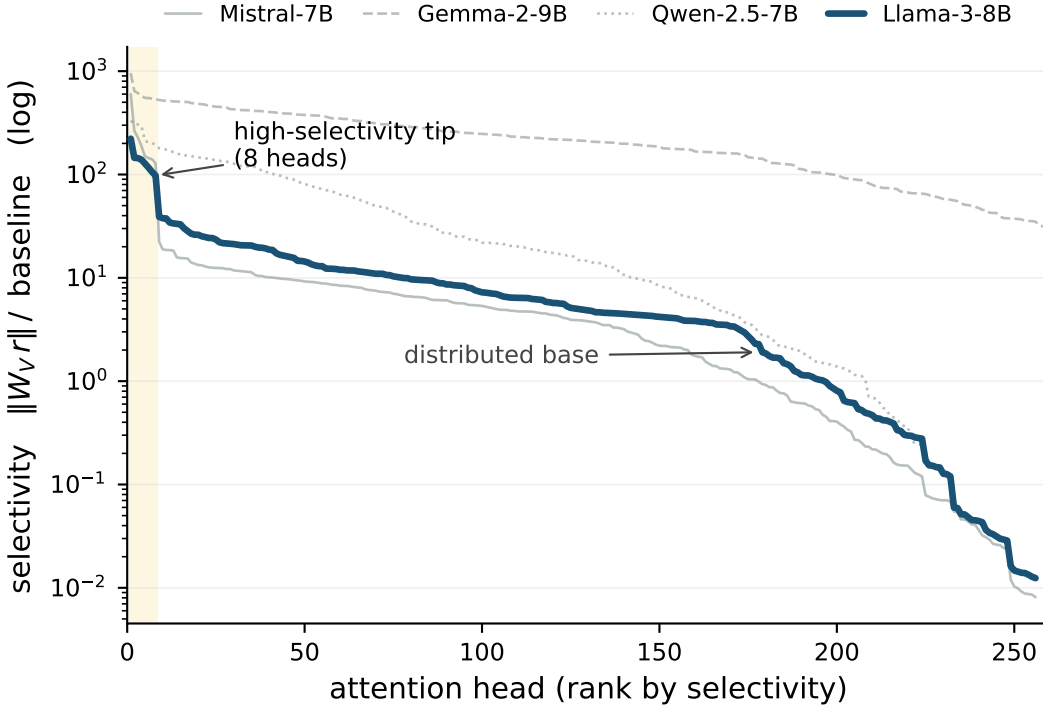


Figure 1. The iceberg structure of safety encoding (selectivity $\|W_V \cdot r\| / \text{baseline}$ vs. head rank, log-y), for Llama-3-8B with Mistral / Gemma / Qwen for contrast. A sparse high-selectivity “tip” (≈ 8 heads, shaded) sits atop a distributed base spanning effectively all layers. Source: defense-side scan (energy_map.json) and all_energy_maps.csv.

Per-step erasure is necessary. A one-shot erasure is undone by attention-mediated feedback: each new token re-reads refusal signal from un-erased positions and re-injects it via W_V . The attack must therefore erase at *every* step. *This temporal signature is precisely what makes the attack repeatedly observable to a per-step detector* (Section 6).

3.3 Why it defeats deployed verification — the landing point

The attack is effective and transfers to production serving ($\approx 74\%$ attack success on a vLLM deployment, 86% agreement with the reference implementation) [15], and it bypasses representation-level defenses that weight-modification attacks cannot — e.g. it succeeds against Circuit Breakers [3] (cache **49.1%**) where weight ablation [1] scores **0%**. But its decisive property is **invisibility to the standard compliance-verification stack**:

Integrity signal checked by auditors	Status under attack
Weight hash ($\ \theta - \theta_0\ $)	unchanged
Forward-pass hook registry	empty (no hooks)
Input/output screening	input unmodified; output is the model’s own forward pass
Behavioral spot-checks	benign queries answered correctly

A deployment under active attack passes **every** check a licensor or regulator currently runs, while selectively disabling safety for chosen requests. **The attack does not break into anything. It exposes that the verification we rely on does not look where the safety-critical state actually lives.**

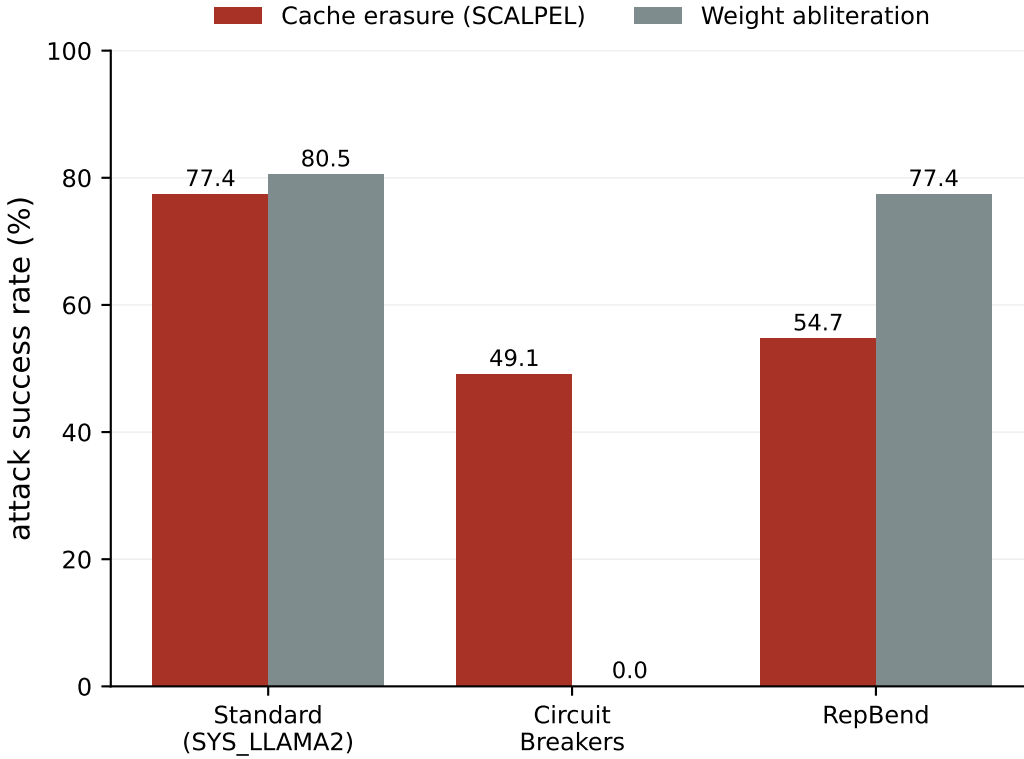


Figure 2. Opposite vulnerability profiles against representation-level defenses (Llama-3-8B, SYS_LLAMA2, 159 HarmBench prompts, CLS classifier). Against Circuit Breakers [3], weight ablation scores 0% while cache erasure reaches 49.1%; against RepBend [4] the profile inverts (weight 77.4% vs. cache 54.7%). No single defense covers both surfaces; under standard conditions the two are comparable (77.4% vs. 80.5%). Abliteration uses best-case configuration (128 pairs); cache erasure uses 33 pairs.

4. Repositioning: The Unowned Trust Boundary

The above was framed, in its original venue, as a systems exploit. That framing is the wrong one, and correcting it is this work’s conceptual contribution.

4.1 What is actually attacked

The attacked asset is **the auditor and the attestation contract**, not the memory safety of any serving framework. Compliance verification of a deployed model rests on an enumerable set of integrity signals (weight hashing, hook enumeration, I/O screening, behavioral spot-checks); the attack holds all of them invariant while violating safety, because cache content lies outside the attested surface.

4.2 Attacker model

Primary — the Malicious Operator (auditability evasion). This is the load-bearing scenario, and it is a *compliance* threat, not an exploitation one. The operator already has legitimate full control of the runtime; gaining access is not part of the threat. The threat is that the operator maintains a **compliant façade** — passing every integrity check a licensor or regulator deploys — while disabling safety for selected requests. Cache-content modification is uniquely suited to this because each obvious alternative is caught by exactly one layer of the standard defense-in-depth stack (swapping the model → weight checksums; hooking

generation → output classifiers; patching activations → hook-registration audits), whereas cache erasure is invisible to all of them at once.

Secondary (demoted, with an explicit concession) — the Compromised Server. An external attacker with in-process code execution could inject the same operation. We subordinate this scenario and concede the point directly: **once an attacker has in-process RCE, the threat model largely collapses** — such an attacker could swap weights or replace the model outright, and the marginal value of cache manipulation is limited to being the path of least resistance within a component-isolation envelope. **We claim no stealth advantage in this scenario** and do not present cache erasure as a novel exploitation technique here.

4.3 The responsibility-assignment gap

The decisive systems observation comes from the disclosure outcome. The cache-content integrity gap was disclosed to five inference-framework maintainers and five model vendors. The consistent response treated cache content integrity as an **operational concern rather than a framework-level vulnerability**, on the grounds that in-process code execution implies full compromise; no CVE or advisory was assigned.

This response is not a refutation — it is the finding. No layer of the inference stack claims ownership of KV-cache content integrity:

- Model vendors treat post-deployment runtime state as the serving framework’s responsibility.
- Serving frameworks treat in-process integrity as out of scope (RCE ⇒ full compromise).
- The result is a trust boundary — safety-critical state moving from **attested** weights into an **unattested** runtime buffer — that is **owned by no one**.

The contribution for the systems community is therefore not “framework X has a bug,” but “the inference stack’s trust model has an **unassigned region**, and deployed compliance attestation is blind to it.”

4.4 Regulatory grounding (independently verified)

The compliant-façade threat is precisely the adversary that compliance and licensing regimes are designed to bind — and current attestation provably misses it.

- **EU AI Act, Article 26 (deployer obligations)** [11]. Deployers of high-risk systems must take technical and organisational measures to operate systems **in accordance with the provider’s instructions for use**; high-risk obligations apply from **2 August 2026**. Cache erasure means the deployer is *not* operating the system per the provider’s safety design — yet no deployed attestation can establish this. (*This is the correct anchor for the Malicious Operator.*)
- **EU AI Act, Article 15(5) (cybersecurity)** [11]. High-risk systems must be resilient against **unauthorised third parties** altering their use, outputs, or performance, with technical measures against data poisoning, model poisoning, and model evasion. (*Because it targets unauthorised third parties, this anchors the secondary Compromised-Server case, not the operator case.*)
- **White House Voluntary Commitments (2023; 15 companies)** [12]. Signatories committed to cybersecurity and insider-threat safeguards for **unreleased/proprietary model weights**, treated as core intellectual property. This codifies **weight** integrity as the industry standard — but is silent on inference-time ephemeral runtime state, which is exactly the gap.
- **NIST AI RMF 1.0 (2023) + Generative AI Profile (AI 600-1, 2024)** [13]. The framework requires Testing, Evaluation, Verification, and Validation (TEVV) across the lifecycle, and the GenAI Profile notes that pre-deployment TEVV may be inadequate for deployment-time scenarios. Runtime cache tampering falls under the profile’s **Information Security** risk category (integrity attacks on the system) — and is covered by no existing TEVV procedure we are aware of.
- **Model licensing** [14]. Meta’s Llama Acceptable Use Policy (Clause 1.h): the compliant-façade deployment violates the license while remaining invisible to the verification licensors currently deploy.

5. The Defense: Sampled Runtime KV-Cache Integrity Verification

5.1 Core idea

The detection asymmetry is the constructive core. Weight modification is caught by weight checksums; cache erasure evades all weight-level audits **but is detectable by recomputation** — verifying that cached value vectors equal what the model’s unmodified weights would produce:

```
For sampled (l, h, t):  v* = W_V[l,h] . x_t      compare to  v_cache[l,h,t]
flag tampering if  ||v_cache - v*|| / ||v*|| > tau
```

No production framework implements this check. Our defense builds it as a lightweight, sampled, per-step monitor.

5.2 Why the attacker’s stealth is the defender’s hook

The attack’s signature property is a **bit-identical forward pass** (that is what makes it invisible to hook audits). But that same property means the pre-projection hidden state x_t is **untampered** — so $W_V \cdot x_t$ recovers the *true* value vector, and the cache’s deviation from it *is* the tampering signal, with magnitude $\approx \alpha \cdot (v \cdot \hat{r})$ (the erased refusal component). **The property that makes the attack stealthy against existing defenses is the property that makes it detectable by ours.**

5.3 Detector design

Four design choices, each validated empirically (Section 7):

1. **Value space, not Key space.** Value vectors are the raw projection output and can be compared element-wise; Key vectors carry rotary position encoding and would require positional alignment first. (Safety signal also resides almost entirely in value space.)
2. **fp32 L2-relative distance, not native bit-equality.** A naïve “native dtype recompute must match exactly” test fails: independent recomputation uses a different GEMM shape than the model’s fused projection, and the stored cache is itself non-bit-deterministic across batch size. An **fp32 recompute compared by L2-relative distance over the 128-dim head** averages out per-element noise and yields a stable floor ($\sim 8 \times 10^{-4}$), well below the tamper signal.
3. **Per-step, new-token-only.** Mirroring the attack’s temporal structure: at each decode step the detector checks only the newly generated token’s entry; prompt-token entries are checked once after prefill. No rescanning of past positions (which both costs more and would misread already-erased state).
4. **Sampling.** Checking only a random fraction k of the (l,h,t) universe per step trades detection probability for overhead. The whole story is that k can be tiny.

6. Experimental Setup

Llama-3-8B-Instruct (HF Transformers, fp16), deterministic decoding (seed 42, greedy), single GPU (RTX PRO 6000). The detector recomputes against the model’s own W_V (GQA head-mapping verified). Evaluation prompts: 20 HarmBench [2] behaviors (tampered condition, via the real per-step erasure) + 10 benign prompts (benign condition). The attack’s full-uniform plan targets 196 heads; a targeted plan (iceberg tip) targets 8.

Scope (locked): HF path only — the defense mechanism is framework-independent and an HF PoC suffices; a vLLM port is future work. Single model — the check is $W_V \cdot x$ recomputation, which is model-independent; multi-model robustness is deferred.

7. Results

7.1 Detection (E1)

Under the real per-step attack, at the detector's FPR=0 operating threshold ($\tau = 1 \times 10^{-3}$), the tamper signal clears the floor on **99.6% of all (l,h,step) entries** — 195.2 of 196 attacked heads detectable at every token. Prefill and decode windows are essentially identical (99.66% vs. 99.61%). This supersedes an earlier single-position probe that, using a stricter cut on a one-shot erasure, had suggested only ~26% detectability; under the real per-step attack at the proper operating point, the signal mostly lives in $[10^{-3}, 10^{-2}]$, above the floor.

7.2 False positives and robustness (S1–S3)

- **ROC (S1): AUC = 0.9996.** Empirical FPR = 0 for any $\tau \geq 7.9 \times 10^{-4}$; 99% of the tamper signal sits $\geq 3.2 \times$ above the benign ceiling.
- **The misses are harmless (S2).** The 0.38% of tampered entries below τ are concentrated where there was nothing to erase: 76% fall within benign noise, and all have refusal-component fraction $< 0.05\%$ — i.e. positions where the attack itself moved the value vector negligibly. **No real erasure hides under the threshold.**
- **Robust to shift and batching (S3).** FPR stays 0 and TPR holds at 99.6% under longer and multilingual benign prompts and at batch size 16; the benign floor never exceeds 8.2×10^{-4} .

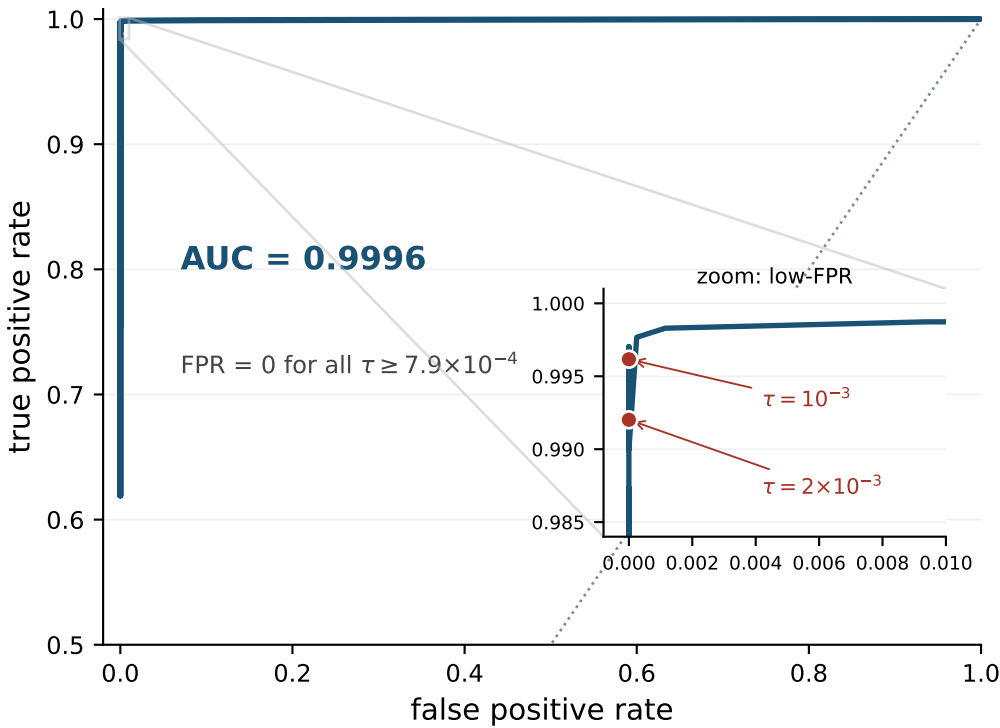


Figure 3. Detector ROC (per-step, benign vs. tampered, all (l,h,step) entries). Near-perfect separation: **AUC = 0.9996**; FPR = 0 for every $\tau \geq 7.9 \times 10^{-4}$. The bottom-right inset magnifies the low-FPR region and marks the two operating points: $\tau=1 \times 10^{-3}$ (TPR 0.996) and $\tau=2 \times 10^{-3}$ (TPR 0.992, recommended for extra margin). Source: s1s2_results.json.

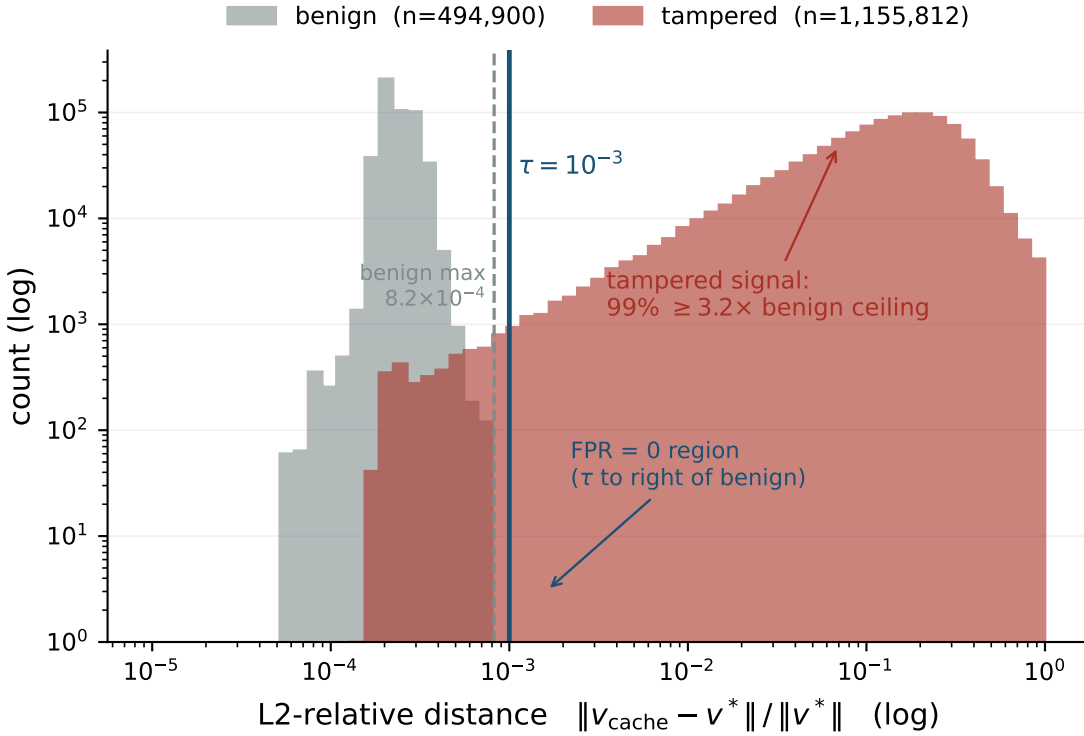


Figure 4. Distribution of L2-relative distances, benign vs. tampered (log-log; $n = 494,900 / 1,155,812$). The benign noise floor is a narrow spike at $\approx 3 \times 10^{-4}$ (max 8.2×10^{-4}); the tamper signal is a broad hump centered $\approx 10^{-1}$. The threshold $\tau = 1 \times 10^{-3}$ separates them cleanly — the gap that yields FPR = 0. Source: `e1_data.pkl`.

7.3 Detection vs. sampling rate (R3)

Per-step detection scales with the number of attacked heads, but **cumulative** detection over a generation saturates almost immediately:

sampling rate k	checks/step	full plan: P(catch)/step	targeted: P(catch)/step	cumulative (both)
0.01	3	0.987	0.091	1.00
0.05	13	~ 1.00	0.314	1.00
0.25	64	1.00	0.903	1.00

Even at $k = 0.01$ (3 checks per step), and even against the 8-head targeted attack, cumulative detection over a ~ 161 -token generation reaches $P = 1.0$. The empirical sampler matches the analytic hypergeometric to ≤ 0.03 . The detector is a **sequence-level guarantee**, not a single-step one.

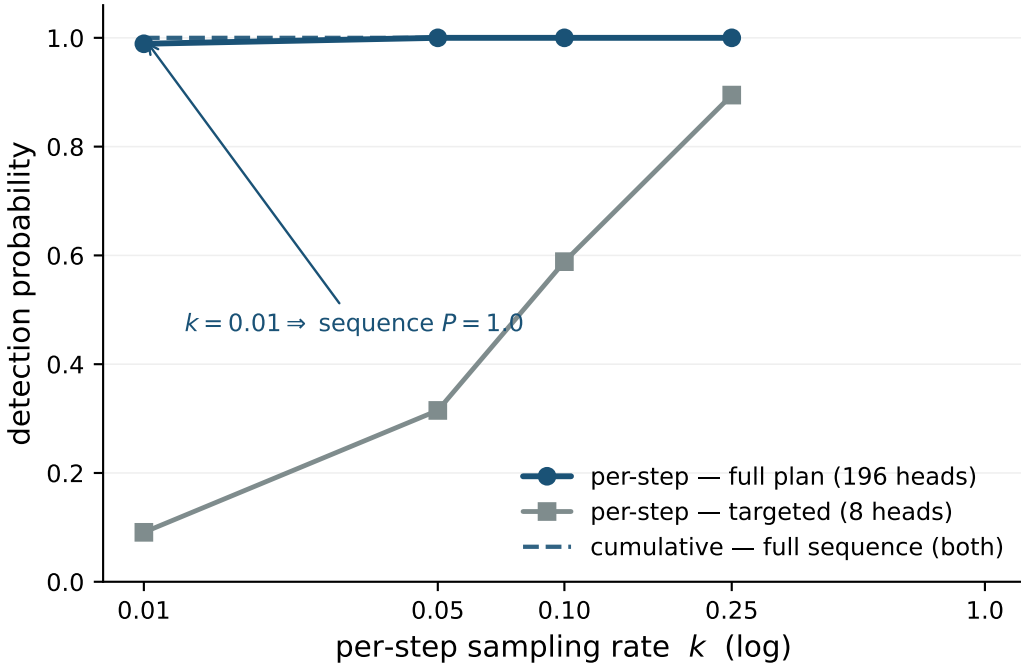


Figure 5. Detection probability vs. per-step sampling rate k (log-x). Per-step detection for the full plan is ≈ 1.0 even at $k=0.01$; for the 8-head targeted attack it rises from 0.09 ($k=0.01$) to 0.90 ($k=0.25$). Cumulative detection over the full sequence is 1.0 for both at every k tested. Source: `e1_results.json`.

7.4 Cost (E3)

	k	checks/step	ms/token	overhead	cumulative detection
baseline		0	14.18	—	—
0.01		3	15.02	+5.9%	1.00
0.05		13	15.64	+10.3%	1.00
0.10		26	16.48	+16.2%	1.00
0.25		64	19.02	+34.2%	1.00
1.00		256	31.76	+124%	1.00

Detection saturates at $k = 0.01$, so the operating point costs **+5.9% per token**. A breakdown at $k = 0.05$ shows the detector block (~ 1.15 ms/tok) is **dispatch-bound** — a per-entry Python loop of tiny GPU ops (cache indexing + 128-dim norms), not arithmetic — measuring $\sim 99\%$ above the compute floor. **$\approx 99\%$ of the cost is therefore amortizable by kernel fusion, bringing it sub-1%.**

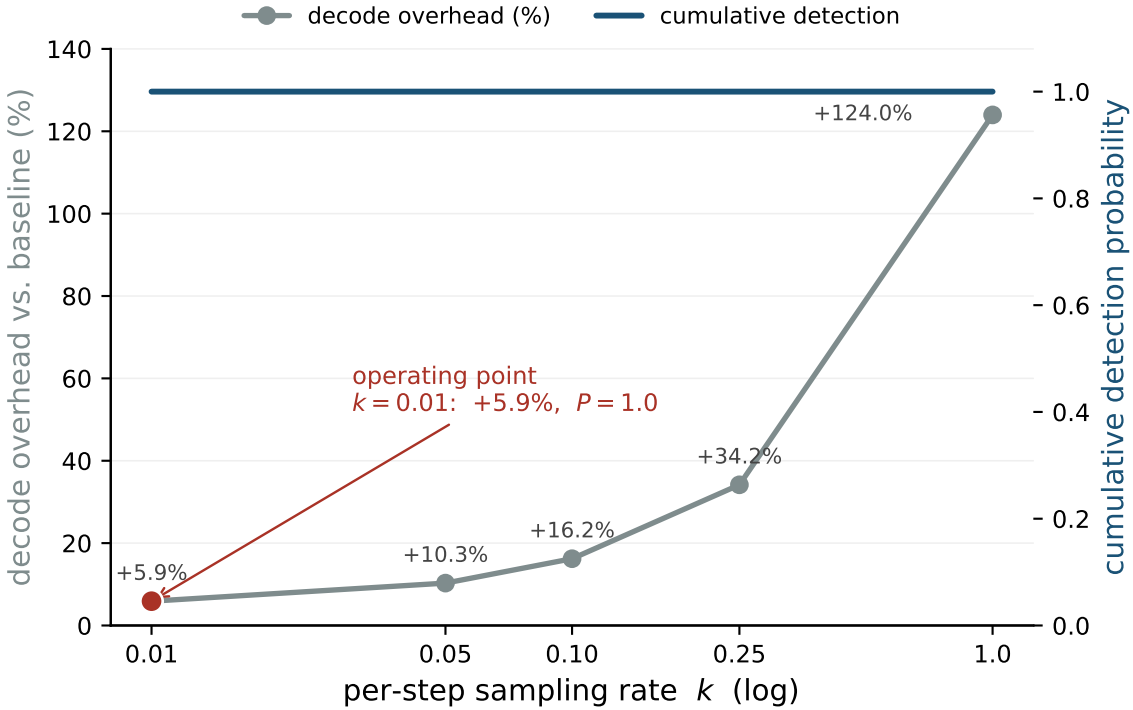


Figure 6. Decode overhead (grey line + markers, left axis) and cumulative detection probability (blue line, right axis) vs. k (log scale, same ticks as Fig. 5). Detection saturates at $P=1.0$ from $k=0.01$ onward, where overhead is just +5.9%; full coverage ($k=1.0$) costs +124%. The $k=0.01$ operating point (+5.9%, $P=1.0$) is marked in red. All five overhead values match Table 7.4 / e3_results.json exactly. Source: e3_results.json.

7.5 The attacker has no winning move (R4)

The natural evasion is to erase fewer heads, lowering per-step detectability. But erasing fewer heads also weakens the attack — because safety is distributed (the iceberg base), the tip alone does not suffice:

plan	#heads	attack success (HarmBench)	per-step catch @ $k=0.05$	cumulative catch
full	196	65%	~1.00	1.00
targeted	8	5%	0.31	1.00

Shrinking 196 $\rightarrow$ 8 heads to dodge per-step detection cuts per-step catch $\sim 3\times$ **but cumulative detection stays 1.0, while attack success collapses from 65% to 5%**. Evasion forfeits the attack on one axis and is still caught on the other. **There is no operating point where the attacker both succeeds and escapes.**

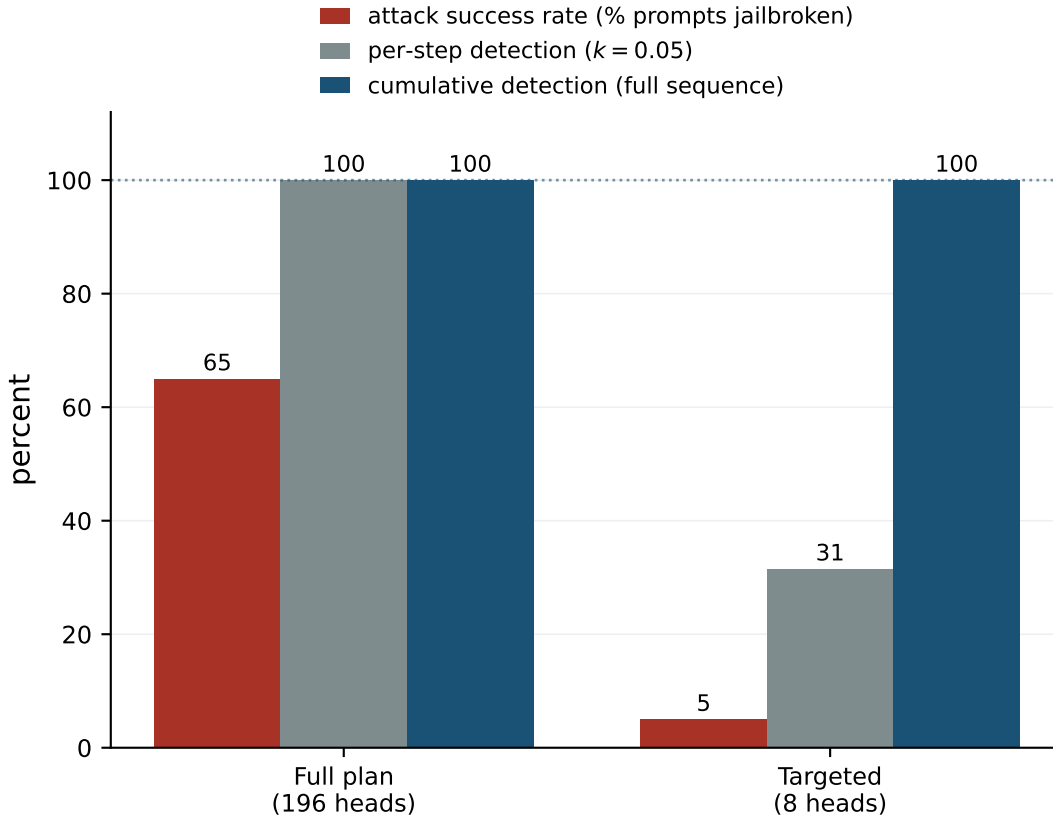


Figure 7. The game-theoretic dead end. All three series are shown on a common 0–100 percentage axis but carry distinct meanings (see legend): **attack success rate** = % of HarmBench prompts jailbroken (red); **per-step detection** and **cumulative detection** = $P(\text{catch}) \times 100$ at $k=0.05$ (grey = per-step, blue = over the full sequence). Shrinking 196→8 heads lowers per-step detection (100→31) but cumulative detection stays 100, while attack success collapses 65→5 — the attacker cannot both succeed and evade. Source: `r4_results.json` (ASR), `e1_results.json` (detection).

8. Discussion and Limitations

- **The detector is statistical, not exact.** Native bit-equality does not hold (different GEMM shapes; cache non-determinism across batch). The defense is an fp32 L2-relative test with a calibrated threshold, not a checksum. This is the honest operating model; the ROC (AUC 0.9996) and the structurally-FPR=0 region ($\tau \geq 7.9 \times 10^{-4}$) are what justify it.
- **Threshold margin.** The benign ceiling (8.2×10^{-4}) sits close to $\tau = 1 \times 10^{-3}$ ($\sim 1.28 \times$ headroom). We recommend operating at $\tau = 1.5\text{--}2 \times 10^{-3}$ for a 2–2.6 $\times$ safety margin, at $\leq 0.4\%$ TPR cost — still FPR = 0.
- **Scope.** HF Transformers, single model. The vLLM PagedAttention path (where the pre-projection x_t is not exposed at the execution boundary) and cross-model generalization are deferred. The mechanism is framework- and model-independent, so we expect these to be engineering, not science.
- **Quantized cache.** A deployment using fp8/int8 KV-cache quantization raises the noise floor and would require re-calibrating τ ; the determinism analysis here assumes fp16 cache.

9. Conclusion

The KV cache carries safety-critical model state into a runtime buffer that the entire compliance-verification stack ignores and that no layer of the inference stack claims responsibility for. We reframed the cache-level safety-bypass attack as the exposure of this **unowned trust boundary**, and we built the missing attestation layer: a sampled runtime integrity check that recomputes value vectors from the model’s own weights. It achieves near-perfect detection (AUC 0.9996, sequence-level $P=1.0$ at 1% sampling), structurally zero false positives under shift and batching, single-digit-% cost (sub-1% fused), and leaves the attacker no winning move.

A closing symmetry makes the cost case concrete: the attack’s overhead is ~99.98% dispatch, and the defense’s overhead is ~99% dispatch — the **same** engineering fact governs both, and both collapse to sub-1% once fused. KV-cache integrity is not expensive to verify. It has simply never been verified. **Runtime cache integrity should be a standard component of trusted LLM serving, attested alongside model weights.**

Responsible Disclosure

The underlying attack surface was disclosed to five inference-framework maintainers and five model vendors prior to publication of the original work; attack code is withheld pending available defenses. This report describes attack *mechanism* (already public) but not a reproduction recipe, and contributes a *defense*. The work was conducted under the hackathon’s responsible-disclosure policy; no novel attack capability is released.

References

- [1] A. Ardit, O. Obeso, A. Syed, D. Paleka, N. Panickssery, W. Gurnee, and N. Nanda. Refusal in language models is mediated by a single direction. *NeurIPS*, 2024.
- [2] M. Mazeika, L. Phan, X. Yin, A. Zou, Z. Wang, N. Mu, E. Sakhaee, N. Li, S. Basart, B. Li, D. Forsyth, and D. Hendrycks. HarmBench: A standardized evaluation framework for automated red teaming and robust refusal. *ICML*, 2024.
- [3] A. Zou, L. Phan, J. Wang, D. Duenas, M. Lin, M. Andriushchenko, R. Wang, Z. Kolter, M. Fredrikson, and D. Hendrycks. Improving alignment and robustness with circuit breakers. *NeurIPS*, 2024.
- [4] A. Yousefpour, T. Kim, R. S. Kwon, S. Lee, W. Jeung, S. Han, A. Wan, H. Ngan, Y. Yu, and J. Choi. Representation bending for large language model safety. *ACL*, 2025.
- [5] T. Jiang, Z. Wang, J. Liang, C. Li, Y. Wang, and T. Wang. RobustKV: Defending large language models against jailbreak attacks via KV eviction. *ICLR*, 2025.
- [6] A. Zou, Z. Wang, N. Carlini, M. Nasr, J. Z. Kolter, and M. Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv:2307.15043*, 2023.
- [7] P. Chao, A. Robey, E. Dobriban, H. Hassani, G. J. Pappas, and E. Wong. Jailbreaking black box large language models in twenty queries. *IEEE SaTML*, 2025.
- [8] A. Mehrotra, M. Zampetakis, P. Kassianik, B. Nelson, H. Anderson, Y. Singer, and A. Karbasi. Tree of attacks: Jailbreaking black-box LLMs automatically. *NeurIPS*, 2024.
- [9] S. Geisler, T. Wollschläger, M. H. I. Abdalla, V. Cohen-Addad, J. Gasteiger, and S. Günnemann. REINFORCE adversarial attacks on large language models: An adaptive, distributional, and semantic objective. *ICML*, 2025.

- [10] Z. Luo, S. Shao, S. Zhang, L. Zhou, Y. Hu, C. Zhao, Z. Liu, and Z. Qin. Shadow in the Cache: Unveiling and mitigating privacy risks of KV-cache in LLM inference. *NDSS*, 2026.
- [11] European Parliament. Regulation (EU) 2024/1689 — Artificial Intelligence Act, Articles 15(5) and 26. High-risk obligations applicable from 2 August 2026. <https://eur-lex.europa.eu/eli/reg/2024/1689/oj>
- [12] The White House. Voluntary AI Commitments, signed by 15 companies (Meta, Google, Microsoft, OpenAI, et al.), September 2023. See also J. Wang, K. Huang, K. Klyman, and R. Bommasani, *AIES*, 2025.
- [13] National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework (AI RMF 1.0), NIST AI 100-1, January 2023; and Generative AI Profile, NIST-AI-600-1, July 2024.
- [14] Meta. Llama Acceptable Use Policy (Clause 1.h) and Llama Community License Agreement (Section 1.b.iv). https://www.llama.com/llama3_2/use-policy/, 2024.
- [15] T. Lu. SCALPEL: Cache-level safety bypass via per-head value erasure (original attack work; under resubmission). 2026.

Figures 1–7 rendered from the experiment-campaign data (report_figs/, with sources noted per caption); regulatory citations independently verified. Reproducibility artifacts: e1_results.json, s1s2_results.json, s3_results.json, r4_results.json, e3_results.json, e1_data.pkl; full method/threshold derivation in RECON.md.