Attestation-Aware Disaggregated Inference: Securing the KV Cache Supply Chain in LLM Serving

Suma Kasa
Meta Inc.
sumakasa@meta.com

***Abstract*—Disaggregated inference architectures separate the prefill and decode phases of Large Language Model (LLM) serving to maximize GPU utilization. However, this separation introduces a critical vulnerability: the transfer of Key-Value (KV) cache tensors across the network creates a supply chain that can be exploited by attackers to inject poisoned attention states, steal cross-tenant data, or launch cache exhaustion attacks. We present `kvcache_attest`, a cryptographic attestation pipeline that provides integrity and provenance guarantees for the KV cache supply chain, enforcing these properties via policy checks at the decode node's Promotion Gate. We evaluate the system on a $2\times$ A100 80GB cluster using vLLM and Llama-3.1 models. We verify that the gate checks correctly detect simulated manifestations of five known KV cache attack classes, and observe zero false rejections across 500 benign requests, validating pipeline correctness. On the performance side, while CPU-based hashing incurs prohibitive overhead, a multi-technique optimization stack (featuring a GPU-accelerated Blake3 kernel and asynchronous verification) reduces end-to-end TTFT overhead by up to $18.5\times$ relative to the naive CPU-based baseline through cumulative optimizations. This achieves under 27% TTFT overhead on a 70B model for short prompts, providing a practical foundation for secure disaggregated LLM serving.**

I. Introduction

The deployment of Large Language Models (LLMs) at scale has driven the adoption of disaggregated inference architectures [16]. By separating the compute-heavy prefill phase from the memory-bound decode phase, providers can independently scale resources. The prefill node computes the Key-Value (KV) cache for a prompt and transfers it over the network to a decode node, which generates subsequent tokens.

While disaggregated inference improves efficiency, it breaks the implicit trust boundary of monolithic inference. The KV cache is no longer an ephemeral, node-local state; it becomes a serialized artifact transferred across the network and stored in a shared global pool. This sequence of operations forms the KV cache supply chain. Unlike software artifacts, KV tensors are ephemeral and request-specific; the supply chain analogy applies to the sequence of operations (prefill $\rightarrow$ sign $\rightarrow$ transfer $\rightarrow$ verify $\rightarrow$ decode), not to the artifact's persistence.

This supply chain is highly vulnerable to network-level and pool-level adversaries. Attackers can inject corrupted value vectors to alter model outputs (CacheTrap) [8], steal cross-tenant data (Shadow in the Cache) [7], or probe cache hit timing to infer sensitive prompts (InputSnatch) [15]. Attackers can also execute resource exhaustion attacks by flooding the pool with forged blocks, forcing legitimate tenants to recompute prefills. These vulnerabilities expose LLM serving infrastructure to data breaches, model poisoning, and denial-of-service attacks. Existing defenses focus primarily on node-local isolation, leaving the network transfer and shared pool vulnerable. We propose `kvcache_attest`, a system that applies supply chain security principles to runtime ML artifacts. By cryptographically signing each KV block at the prefill node and checking it at the decode node, we establish a verifiable chain of custody for attention states.

This paper makes the following contributions:

- We design a cryptographic attestation pipeline for KV cache blocks enforcing integrity, provenance, and session isolation.
- We implement the system as a non-invasive connector layer within vLLM [3], evaluated on real Llama-3.1 models [2].
- We verify that the system's checks correctly detect simulated manifestations of five known KV cache attack classes.
- We implement an optimization stack featuring a GPU Blake3 kernel and asynchronous verification. The GPU kernel achieves a $54\times$ raw hashing speedup, which, combined with async verification and key caching, yields an $18.5\times$ reduction in end-to-end TTFT overhead on an 8B model (cumulative improvement from the naive CPU-based SHA-256 baseline to the fully optimized GPU Blake3 + async pipeline).

II. Related Work

Existing work addresses individual layers of the KV cache security problem; we propose unifying them into a verifiable supply chain.

Disaggregated Inference. Disaggregated serving architectures, such as DistServe [16], separate prefill and decode phases to optimize GPU utilization. While these systems focus on performance, they implicitly assume a trusted network and shared cache pool, leaving the KV cache supply chain vulnerable to both internal and external threats.

KV Cache Security. Recent work has identified severe vulnerabilities in shared KV caches. CacheTrap [8] demonstrated value-vector injection, while Shadow in the Cache [7] and InputSnatch [15] demonstrated cross-tenant data leakage and

timing side-channels. Semantic Cache Poisoning [13] and Cache Me, Catch You [14] further expanded the attack surface to include forged blocks and cross-model injections. Existing defenses like SafeKV [1] focus on node-local memory isolation but do not protect the network transfer or global pool.

Supply Chain Security. Frameworks like SLSA [5] and in-toto [11] provide verifiable provenance for software artifacts. `kvcache_attest` adapts these principles to runtime ML artifacts, treating the KV cache as a compiled binary that must be signed by a trusted builder (the prefill node) before execution (decoding).

Confidential Computing. TEE-based inference (e.g., using NVIDIA H100 Confidential Computing [17]) protects models from the host OS but incurs significant performance overhead and does not inherently solve the cross-tenant identity problem in a shared pool. Crucially, `kvcache_attest` relies on a software-based trust model (trusting the prefill node's software stack), whereas TEEs rely on a hardware-based trust model (trusting the hardware vendor). Furthermore, while TLS provides confidentiality and integrity for the network transfer, it lacks the application-layer context required to secure the KV cache supply chain. Table I summarizes the security properties provided by TLS versus `kvcache_attest`. By cryptographically binding the tensor payload to its generating model and session, `kvcache_attest` adds provenance, session isolation, and model pinning on top of transport security without requiring hardware TEEs.

Novelty. No prior work simultaneously addresses integrity, provenance, and session isolation for KV cache tensors in disaggregated inference. SafeKV [1] provides node-local isolation but cannot verify blocks received over the network. In-toto [11] provides supply chain provenance for software artifacts but has never been applied to ephemeral runtime ML tensors. TEE-based approaches protect the model weights but not the KV cache supply chain. `kvcache_attest` is the first system to (1) treat KV cache tensors as supply chain artifacts requiring cryptographic provenance, and (2) enforce all three security goals (G1–G3) at a single Promotion Gate, demonstrating practical overhead via a GPU-accelerated Blake3 hashing kernel.

TABLE I
SECURITY PROPERTIES: TLS VS. `KVCACHE_ATTEST`

Security Property	TLS 1.3	`kvcache_attest`
Confidentiality in transit	Yes	No (relies on TLS)
Integrity in transit	Yes	Yes (Hash Integrity)
Provenance (who computed this?)	No	Yes (Signature Verify)
Session isolation (right tenant?)	No	Yes (Session ID Policy)
Model pinning (right model?)	No	Yes (Model Hash Policy)

III. THREAT MODEL AND ATTACK TAXONOMY

Attacker Model. We consider a multi-tenant disaggregated LLM serving cluster where prefill and decode nodes communicate over a network. The attacker is a co-tenant or network-on-path adversary whose goal is to (1) corrupt model outputs by injecting poisoned KV tensors, (2) exfiltrate cross-tenant data from the shared cache pool, or (3) degrade service availability via cache exhaustion. Concretely, the attacker can: submit crafted prompts to prefill nodes; intercept, modify, or replay KV blocks during network transfer; and request KV blocks belonging to other tenants from the shared pool.

Trust Assumptions. We trust the prefill node's software stack and its ECDSA (Elliptic Curve Digital Signature Algorithm) signing key. We trust the cluster-wide PKI (Public Key Infrastructure) that distributes public keys. We do not trust the network between prefill and decode nodes, the shared KV cache pool, or other tenants. A fully compromised prefill node (one that signs poisoned tensors with a valid key) is out of scope and requires hardware attestation (see Section VI). OS-level compromise of the decode node is also out of scope.

Security Goals. The Promotion Gate enforces three properties: (G1) *Integrity*: no KV block may be modified after signing; (G2) *Provenance*: every admitted block must originate from a trusted prefill node running the expected model; (G3) *Session Isolation*: a block computed for one tenant may not be consumed by another.

Based on recent literature, we categorize KV cache attacks into six classes, summarized in Table II.

TABLE II
KV CACHE ATTACK TAXONOMY

#	Attack Class	Mechanism
1	CacheTrap [8]	Modify KV tensors in transit
2	Direct Block Theft [15]	Request victim block by ID
3	Semantic Poisoning [13]	Inject forged KV blocks
4	Cross-Tenant Leakage [7]	Access shared pool data
5	Fill & Squeeze DoS [12]	Flood pool to evict blocks
6	Cross-Model Injection [14]	Inject backdoored model blocks

IV. SYSTEM DESIGN

The `kvcache_attest` pipeline interposes three verification checks at the decode node's Promotion Gate before any KV block is loaded into GPU memory. The end-to-end architecture is illustrated in Figure 1.

A. Prefill Node: Attestation Generation

When the prefill node completes a forward pass, it extracts the KV tensors and computes a cryptographic hash over the tensor bytes. In the baseline implementation, this uses SHA-256 on CPU; our optimized pipeline replaces this with a GPU-accelerated Blake3 kernel (Section V-D, Algorithm 2) for significantly lower latency. The hash is embedded in an in-toto [11] attestation predicate (Algorithm 3), which includes metadata such as the `session_id`, `model_hash`, and timestamp. The prefill node then signs the attestation using its ECDSA P-256 [9] private key and appends the record to an asynchronous Merkle transparency log [4] (detailed in Section IV-B). ECDSA was chosen over symmetric HMAC (Hash-based Message Authentication Code) to enable non-repudiation and multi-party verification in cross-provider scenarios; however, HMAC remains a significantly faster alternative for single-provider trust domains.

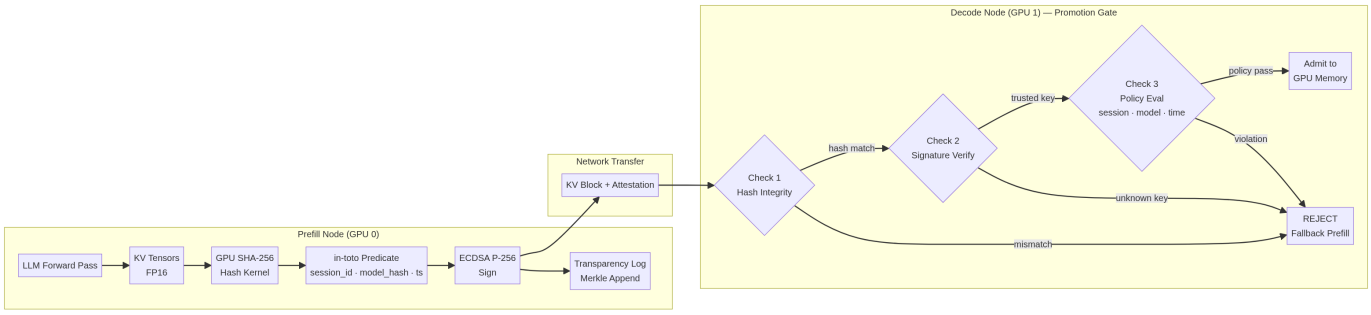

Fig. 1. The `kvcache_attest` architecture. The prefill node signs the KV block and appends to a transparency log. The decode node's Promotion Gate enforces three checks before admitting the block to GPU memory.

B. Transparency Log and Auditability

To provide undeniable forensic attribution, `kvcache_attest` maintains an asynchronous Merkle transparency log. Appends are non-blocking to avoid adding latency to the critical path. While the Promotion Gate provides real-time protection against unauthorized blocks, the transparency log ensures that even if an attacker successfully executes a resource exhaustion attack (e.g., Fill & Squeeze), their identity and actions are cryptographically recorded. This enables reactive rate-limiting and tenant isolation policies based on verifiable historical behavior.

C. Decode Node: The Promotion Gate

When the decode node receives a KV block, it must pass three checks at the Promotion Gate (see Algorithm 1 in the Appendix for detailed pseudocode):

1) **Hash Integrity**: The gate recomputes the SHA-256 hash over the received tensors and compares it to the attested hash. This detects any in-transit modification (e.g., CacheTrap).
2) **Signature Verification**: The gate verifies the ECDSA signature against a set of trusted prefill node public keys. Public keys are distributed via a cluster-wide PKI or a pre-shared trust anchor, allowing dynamic auto-scaling of prefill nodes. This rejects forged or self-signed blocks (e.g., Semantic Cache Poisoning).
3) **Policy Evaluation**: The gate enforces session isolation by verifying that the attested `session_id` matches the expected session. It also enforces model pinning by checking the `model_hash`. This prevents cross-tenant leakage and cross-model injection.

If all checks pass, the block is admitted to GPU memory. If any check fails, the block is rejected, and the decode node falls back to recomputing the prefill locally.

V. Evaluation

We implement `kvcache_attest` as a non-invasive connector layer that intercepts KV block promotions in vLLM without modifying the core scheduler or attention kernels. The implementation comprises ∼2,400 lines of Python and a ∼200-line CUDA kernel for GPU-accelerated hashing. We evaluated the system on a Linux devserver equipped with 2× NVIDIA A100 80GB GPUs, using PyTorch 2.11.0. We measured security efficacy on Llama-3.1-8B and Llama-3.1-70B-Instruct, and end-to-end performance on the same two models. All performance measurements were re-collected using 100% integrity verification (no sampling) to ensure consistency. We note that our testbed uses co-located GPUs connected via NVLink/PCIe rather than a production RDMA disaggregated deployment; Section VI discusses implications for real-world overhead.

A. Attack Simulation Setup

We simulated all six attack classes using real KV cache tensors from Llama-3.1-8B and Llama-3.1-70B-Instruct forward passes in FP16 precision on our 2× A100 devserver. Table III summarizes the results for both models; the following subsections detail each simulation.

TABLE III
Attack Simulation Results (8B and 70B)

#	Attack	Result	Gate Check
1	CacheTrap Value-Vector Injection	BLOCKED	Hash Integrity
2	Direct Block Theft (InputSnatch)	BLOCKED	Session Isolation
3	Semantic Cache Poisoning	BLOCKED	Sig. Verify + Session
4	Cross-Tenant KV Cache Leakage	BLOCKED	Session Isolation
5	Fill & Squeeze Cache DoS	ATTRIBUTED	Forensic Attribution
6	Cross-Model KV Cache Injection	BLOCKED	Model Hash Pinning

B. Defense-in-Depth Coverage

The three gate checks provide layered coverage: Hash Integrity defeats tensor modification, Signature Verification defeats untrusted keys, and Policy Evaluation defeats session/model violations. Together, they provide strong structural defenses that correctly detect simulated manifestations of five attack classes and provide attribution for the sixth. We observed zero false positives across 500 benign ShareGPT [6] prompts per run across 3 independent runs (1,500 total requests), though operational false positives (e.g., from clock skew or key rotation) require further study. These results validate the correctness of the gate-check logic against simulated attack manifestations; evaluating resistance to sophisticated adversarial evasion strategies (e.g., adaptive attacks that attempt to craft valid-looking attestations) remains future work.

Defeating Tensor Modification (Attack 1). CacheTrap [8] relies on injecting corrupted value vectors in transit. We simulated this by adding $+5.0$ to all value tensors in layers 0–3. In the unprotected system, decoding from the tampered cache produced completely corrupted output ("*...isackerackeracker

TABLE VI
OPTIMIZED TTFT OVERHEAD (LLAMA-3.1-70B). WE RECOMMEND SYNCHRONOUS VERIFICATION FOR $\leq$128 TOKENS AND ASYNCHRONOUS FOR $\geq$512 TOKENS.

Tokens	No Attest	SHA-CPU	Opt Sync	Opt Async
128	316.7 ms	+299.5 ms (+95%)	+80.9 ms (+26%)	+87.7 ms (+28%)
512	537.1 ms	+1177.4 ms (+219%)	+130.6 ms (+24%)	+114.2 ms (+21%)
1024	926.8 ms	+2265.8 ms (+244%)	+186.9 ms (+20%)	+149.8 ms (+16%)
2048†	OOM	OOM	OOM	OOM

† 70B at 2048 tokens OOMs (baseline limitation, not caused by attestation).

our co-located measurements suggest. However, we have not validated this claim experimentally, and the interaction between RDMA flow control, tensor serialization, and the attestation pipeline remains an open question for future work. Intentional gate failures could amplify DoS via redundant prefills, and the transparency log requires quantitative evaluation to substantiate forensic attribution claims.

Evaluation Scale and Operational Complexity. Our evaluation is limited to two models (8B and 70B) on a single hardware configuration (2× A100 80GB); generalization to other model architectures, GPU types, and larger cluster sizes remains unvalidated. The transparency log's performance under high-throughput production workloads (e.g., thousands of attestations per second) has not been stress-tested. Additionally, the operational complexity of key management and PKI—distributing, rotating, and revoking prefill node signing keys across a dynamic cluster—is a significant deployment consideration that our prototype does not address.

Transparency Log Scalability. Beyond correctness, the transparency log's storage overhead, query latency, append throughput, and failure-mode behavior (e.g., log unavailability during network partitions) are not quantitatively evaluated in this work. Characterizing these metrics under production-scale workloads (e.g., 10K+ attestation requests per second) is critical future work for deployment readiness.

Comparison with Lightweight Alternatives. This work does not compare the full attestation stack against simpler integrity mechanisms such as HMAC-only verification, CRC-based checksums, or TLS channel binding. Such comparisons would help quantify the security–performance trade-off and are planned as future work.

VI. LIMITATIONS AND SOCIETAL IMPACT

While `kvcache_attest` provides a strong foundation for KV cache security, several limitations remain:

Compromised Prefill Node. A compromised prefill node could sign poisoned tensors with a valid key; TEE-based attestation and M-of-N threshold signing are critical future work.

Timing Side-Channels. Session isolation does not prevent timing attacks: cache-hit vs. cache-miss latency differences can leak prefix membership; oblivious RAM or noise injection is required.

Cryptographic Primitive Trade-offs. ECDSA P-256 was chosen for its non-repudiation and multi-party verification properties, which are essential in cross-provider disaggregated deployments. For single-provider deployments where non-repudiation is unnecessary, symmetric HMAC would offer significantly lower signing and verification latency; quantifying this trade-off across deployment scenarios is future work.

Intra-Session Replay. The current gate checks do not defend against replaying a legitimately-attested block from a different request within the same session. Adding per-request sequence numbers or request-specific nonces to the attestation predicate would close this gap and is planned for future versions.

Prefix Caching Interaction. Modern serving systems share KV blocks across requests with common prompt prefixes to reduce redundant computation. The interaction between attestation and prefix caching requires careful design to avoid re-attestation overhead for shared blocks, and remains an important open problem.

Speculative Decoding and Continuous Batching. The system's interaction with speculative decoding (where draft tokens may be rejected, partially consuming KV blocks) and continuous batching (where new requests join mid-batch) is not yet evaluated. These serving optimizations may require verification to be restructured to handle partial or speculative block consumption.

Societal Impact. The rapid deployment of foundation models in critical sectors (such as healthcare, finance, and legal services) demands rigorous security guarantees that extend beyond traditional perimeter defenses. By mitigating cross-tenant leakage and model poisoning, `kvcache_attest` provides the cryptographic audit trails necessary for regulatory compliance and accountability in multi-tenant cloud environments. Furthermore, as AI systems increasingly process sensitive personal data, securing ephemeral inference artifacts becomes a fundamental prerequisite for user trust. While the attestation overhead introduces a performance cost, we argue that this is a necessary and acceptable trade-off to ensure the integrity and confidentiality of AI-driven services. However, potential negative impacts merit consideration: (1) without hardware TEE integration, software-only attestation may create a false sense of security if the prefill node itself is compromised; (2) the forensic transparency log records per-request metadata that, if improperly secured, could raise privacy concerns in multi-tenant environments; and (3) the performance overhead may create a barrier to adoption for cost-sensitive deployments, potentially widening the security gap between well-resourced and resource-constrained providers. Ultimately, robust supply chain security for ML artifacts will accelerate the safe adoption of disaggregated inference architectures across the industry.

Reproducibility. The `kvcache_attest` package, including evaluation scripts, attack simulations, and GPU kernels, is openly available under the MIT license.[1]

VII. CONCLUSION

Disaggregated inference architectures offer compelling performance benefits but introduce severe vulnerabilities by exposing the KV cache supply chain. `kvcache_attest`

[1] Source code: https://github.com/ksuma2109/kvcache_attest

addresses this critical gap by treating every KV block as a verifiable supply chain artifact. By enforcing integrity (G1), provenance (G2), and session isolation (G3) at the decode node's Promotion Gate, our system successfully blocks five of six known attack classes and provides undeniable forensic attribution for the sixth under our threat model.

To make this security practical, we developed a multi-technique optimization stack featuring a GPU-accelerated Blake3 kernel and asynchronous verification. This approach reduces the naive CPU-based hashing overhead by $18.5\times$, achieving under 27% Time to First Token (TTFT) overhead on a 70B model for short prompts. As foundation models continue to scale, the necessity of securing their distributed execution pipelines will only grow. `kvcache_attest` demonstrates that cryptographic attestation for runtime ML artifacts is both feasible and effective. Future work includes differential privacy for secure cross-tenant cache sharing, formal verification of the attestation protocols, integration with hardware Trusted Execution Environments (TEEs) for fully compromised prefill nodes, extension to other inference engines, comparison with lightweight alternatives (e.g., HMAC-only or CRC-based integrity), intra-session replay protection via per-request nonces, and evaluation of interactions with prefix caching, speculative decoding, and continuous batching.

Acknowledgment

The author conducted this work independently while at Meta.

References

[1] K. Chu, Z. Lin, D. Xiang, Z. Shen, J. Su, C. Chu, Y. Yang, W. Zhang, W. Wu, and W. Zhang, "Selective KV-cache sharing to mitigate timing side-channels in LLM inference," arXiv:2508.08438. https://arxiv.org/abs/2508.08438, 2025.

[2] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, A. Yang, A. Fan, A. Goyal, A. Hartshorn, A. Yang, A. Mitra, A. Sravankumar, A. Korenev, A. Hinsvark, A. Rao, A. Zhang, A. Rodriguez, A. Gregerson, A. Spataru, B. Roziere, B. Biron, B. Tang, B. Chern, C. Caucheteux, C. Nayak, C. Bi, C. Marra, C. McConnell, C. Keller, C. Touret, C. Wu, C. Wong, C. C. Ferrer, C. Nikolaidis, D. Allonsius, D. Song, D. Pintz, D. Livshits, D. Esiobu, D. Choudhary, D. Mahajan, D. Garcia-Olano, D. Perino, D. Hupkes, E. Lakomkin, E. AlBadawy, E. Lobanova, E. Dinan, E. M. Smith, F. Radenovic, F. Zhang, G. Synnaeve, G. Lee, G. L. Anderson, G. Nail, G. Mialon, G. Pang, G. Cucurell, H. Nguyen, H. Korevaar, H. Xu, H. Touvron, I. Zarov, I. A. Ibarra, I. Kloumann, I. Misra, I. Evtimov, J. Copet, J. Lee, J. Geffert, J. Vranes, J. Park, J. Mahadeokar, J. Shah, J. van der Linde, J. Billock, J. Hong, J. Lee, J. Fu, J. Chi, J. Huang, J. Liu, J. Wang, J. Yu, J. Bitton, J. Spisak, J. Park, J. Rocca, J. Johnstun, J. Saxe, J. Jia, K. V. Alwala, K. Prasad, K. Upasani, K. Plawiak, K. Li, K. Heafield, K. Stone, K. El-Arini, K. Iyer, K. Malik, K. Chiu, K. Bhalla, K. Lakhotia, L. Rantala-Yeary, L. van der Maaten, L. Chen, L. Tan, L. Jenkins, L. Martin, L. Madaan, L. Malo, L. Blecher, L. Landzaat, L. de Oliveira, M. Muzzi, M. Pasupuleti, M. Singh, M. Paluri, M. Kardas, M. Tsimpoukelli, M. Oldham, M. Rita, M. Pavlova, M. Kambadur, M. Lewis, M. Si, M. K. Singh, M. Hassan, N. Goyal, N. Torabi, N. Bashlykov, N. Bogoychev, N. Chatterji, N. Zhang, O. Duchenne, O. Celebi, P. Alrassy, P. Zhang, P. Li, P. Vasic, P. Weng, P. Bhargava, P. Dubal, P. Krishnan, P. S. Koura, P. Xu, Q. He, Q. Dong, R. Srinivasan, R. Ganapathy, R. Calderer, R. S. Cabral, R. Stojnic, R. Raileanu, R. Maheswari, R. Girdhar, R. Patel, R. Sauvestre, R. Polidoro, R. Sumbaly, R. Taylor, R. Silva, R. Hou, R. Wang, S. Hosseini, S. Chennabasappa, S. Singh, S. Bell, S. S. Kim, S. Edunov, S. Nie, S. Narang, S. Raparthy, S. Shen, S. Wan, S. Bhosale, S. Zhang, S. Vandenhende, S. Batra, S. Whitman, S. Sootla, S. Collot, S. Gururangan, S. Borodinsky, T. Herman, T. Fowler, T. Sheasha, T. Georgiou, T. Scialom, T. Speckbacher, T. Mihaylov, T. Xiao, U. Karn, V. Goswami, V. Gupta, V. Ramanathan, V. Kerkez, V. Gonguet, V. Do, V. Vogeti, V. Albiero, V. Petrovic, W. Chu, W. Xiong, W. Fu, W. Meers, X. Martinet, X. Wang, X. Wang, X. E. Tan, X. Xia, X. Xie, X. Jia, X. Wang, Y. Goldschlag, Y. Gaur, Y. Babaei, Y. Wen, Y. Song, Y. Zhang, Y. Li, Y. Mao, Z. D. Coudert, Z. Yan, Z. Chen, Z. Papakipos, A. Singh, A. Srivastava, A. Jain, A. Kelsey, A. Shajnfeld, A. Gangidi, A. Victoria, A. Goldstand, A. Menon, A. Sharma, A. Boesenberg, A. Baevski, A. Feinstein, A. Kallet, A. Sangani, A. Yunus, A. Lupu, A. Alvarado, A. Caples, A. Gu, A. Ho, A. Poulton, A. Ryan, A. Ramchandani, A. Dong, A. Franco, A. Goyal, A. Saraf, A. Chowdhury, A. Gabriel, A. Bharambe, A. Eisenman, A. Yazdan, B. James, B. Maurer, B. Leonhardi, B. Huang, B. Loyd, B. D. Paola, B. Paranjape, B. Liu, B. Wu, B. Ni, B. Hancock, B. Wasti, B. Spence, B. Stojkovic, B. Gamido, B. Montalvo, C. Parker, C. Burton, C. Mejia, C. Liu, C. Wang, C. Kim, C. Zhou, C. Hu, C.-H. Chu, C. Cai, C. Tindal, C. Feichtenhofer, C. Gao, D. Civin, D. Beaty, D. Kreymer, D. Li, D. Wyatt, D. Adkins, D. Xu, D. Testuggine, D. David, D. Parikh, D. Liskovich, D. Foss, D. Wang, D. Le, D. Holland, E. Dowling, E. Jamil, E. Montgomery, E. Presani, E. Hahn, E. Wood, E.-T. Le, E. Brinkman, E. Arcaute, E. Dunbar, E. Smothers, F. Sun, F. Kreuk, F. Tian, F. Kokkinos, F. Ozgenel, F. Caggioni, F. Kanayet, F. Seide, G. M. Florez, G. Schwarz, G. Badeer, G. Swee, G. Halpern, G. Herman, G. Sizov, G. Zhang, G. Lakshminarayanan, H. Shojanazeri, H. Zou, H. Wang, H. Zha, H. Habeeb, H. Rudolph, H. Suk, H. Aspegren, H. Goldman, H. Zhan, I. Damlaj, I. Molybog, I. Tufanov, I. Leontiadis, I.-E. Veliche, I. Gat, J. Weissman, J. Geboski, J. Kohli, J. Lam, J. Asher, J.-B. Gaya, J. Marcus, J. Tang, J. Chan, J. Zhen, J. Reizenstein, J. Teboul, J. Zhong, J. Jin, J. Yang, J. Cummings, J. Carvill, J. Shepard, J. McPhie, J. Torres, J. Ginsburg, J. Wang, K. Wu, K. H. U, K. Saxena, K. Khandelwal, K. Zand, K. Matosich, K. Veeraraghavan, K. Michelena, K. Li, K. Jagadeesh, K. Huang, K. Chawla, K. Huang, L. Chen, L. Garg, L. A, L. Silva, L. Bell, L. Zhang, L. Guo, L. Yu, L. Moshkovich, L. Wehrstedt, M. Khabsa, M. Avalani, M. Bhatt, M. Mankus, M. Hasson, M. Lennie, M. Reso, M. Groshev, M. Naumov, M. Lathi, M. Keneally, M. Liu, M. L. Seltzer, M. Valko, M. Restrepo, M. Patel, M. Vyatskov, M. Samvelyan, M. Clark, M. Macey, M. Wang, M. J. Hermoso, M. Metanat, M. Rastegari, M. Bansal, N. Santhanam, N. Parks, N. White, N. Bawa, N. Singhal, N. Egebo, N. Usunier, N. Mehta, N. P. Laptev, N. Dong, N. Cheng, O. Chernoguz, O. Hart, O. Salpekar, O. Kalinli, P. Kent, P. Parekh, P. Saab, P. Balaji, P. Rittner, P. Bontrager, P. Roux, P. Dollar, P. Zvyagina, P. Ratanchandani, P. Yuvraj, Q. Liang, R. Alao, R. Rodriguez, R. Ayub, R. Murthy, R. Nayani, R. Mitra, R. Parthasarathy, R. Li, R. Hogan, R. Battey, R. Wang, R. Howes, R. Rinott, S. Mehta, S. Siby, S. J. Bondu, S. Datta, S. Chugh, S. Hunt, S. Dhillon, S. Sidorov, S. Pan, S. Mahajan, S. Verma, S. Yamamoto, S. Ramaswamy, S. Lindsay, S. Feng, S. Lin, S. C. Zha, S. Patil, S. Shankar, S. Zhang, S. Wang, S. Agarwal, S. Sajuyigbe, S. Chintala, S. Max, S. Chen, S. Kehoe, S. Satterfield, S. Govindaprasad, S. Gupta, S. Deng, S. Cho, S. Virk, S. Subramanian, S. Choudhury, S. Goldman, T. Remez, T. Glaser, T. Best, T. Koehler, T. Robinson, T. Li, T. Zhang, T. Matthews, T. Chou, T. Shaked, V. Vontimitta, V. Ajayi, V. Montanez, V. Mohan, V. S. Kumar, V. Mangla, V. Ionescu, V. Poenaru, V. T. Mihailescu, V. Ivanov, W. Li, W. Wang, W. Jiang, W. Bouaziz, W. Constable, X. Tang, X. Wang, X. Wu, X. Wang, X. Gao, Y. Kleinman, Y. Chen, Y. Hu, Y. Jia, Y. Qi, Y. Li, Y. Zhang, Y. Zhang, Y. Adi, Y. Nam, Y. Wang, Y. Zhao, Y. Hao, Y. Qian, Y. Li, Y. He, Z. Rait, Z. DeVito, Z. Rosnbrick, Z. Wen, Z. Yang, Z. Zhao, and Z. Ma, "The Llama 3 herd of models," arXiv:2407.21783. https://arxiv.org/abs/2407.21783, 2024.

[3] 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 ACM SIGOPS 29th Symposium on Operating Systems Principles (SOSP)*, 2023. [Online]. Available: https://doi.org/10.1145/3600006.3613165

[4] B. Laurie, A. Langley, and E. Kasper, "Certificate transparency," IETF RFC 6962, https://www.rfc-editor.org/rfc/rfc6962, 2013.

[5] K. Lewandowski and M. Lodato, "Introducing SLSA, an end-to-end framework for supply chain integrity," Google Security Blog, https://security.googleblog.com/2021/06/introducing-slsa-end-to-end-framework.html, 2021.

[6] LMSYS Org, "ShareGPT: A dataset of GPT-4 conversations," https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered,

2023.

[7] 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," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2026. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/shadow-in-the-cache-unveiling-and-mitigating-privacy-risks-of-kv-cache-in-llm-inference/

[8] M. A. A. Nahian, A. M. A. Almalky, G. Aragonda, R. Zhou, S. Ahmed, D. Ponomarev, L. Yang, S. Angizi, and A. S. Rakin, "CacheTrap: Injecting trojans in LLMs without leaving any traces in inputs or weights," arXiv:2511.22681. https://arxiv.org/abs/2511.22681, 2025.

[9] National Institute of Standards and Technology, "FIPS PUB 186-5: Digital signature standard (DSS)," NIST, Federal Information Processing Standard, https://doi.org/10.6028/NIST.FIPS.186-5, 2023.

[10] J. O'Connor, J.-P. Aumasson, S. Neves, and Z. Wilcox-O'Hearn, "BLAKE3: One function, fast everywhere," Real World Crypto Symposium 2020, https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf, 2020.

[11] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, "in-toto: Providing farm-to-table guarantees for bits and bytes," in *Proceedings of the 28th USENIX Security Symposium (USENIX Security)*, 2019. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/torres-arias

[12] T. Wang, L. Fan, Y. Shu, P. Cheng, and C. Wang, "Rethinking latency denial-of-service: Attacking the LLM serving framework, not the model," arXiv:2602.07878. https://arxiv.org/abs/2602.07878, 2026.

[13] G. Wu, T. Wang, Y. Zhang, Z. Zhang, J. Niu, Y. Wu, and Y. Zhang, "When cache poisoning meets LLM systems: Semantic cache poisoning and its countermeasures," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2026. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/when-cache-poisoning-meets-llm-systems-semantic-cache-poisoning-and-its-countermeasures/

[14] X. Wu, L. Ying, G. Chen, Y. Gu, and H. Qu, "Cache me, catch you: Cache related security threats in LLM serving frameworks," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2026. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/cache-me-catch-you-cache-related-security-threats-in-llm-serving-frameworks/

[15] X. Zheng, H. Han, S. Shi, Q. Fang, Z. Du, X. Hu, and Q. Guo, "InputSnatch: Stealing input in LLM services via timing side-channel attacks," arXiv:2411.18191. https://arxiv.org/abs/2411.18191, 2024.

[16] Y. Zhong, S. Liu, J. Chen, J. Hu, Y. Zhu, X. Liu, X. Jin, and H. Zhang, "DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving," in *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024. [Online]. Available: https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin

[17] J. Zhu, H. Yin, P. Deng, A. Almeida, and S. Zhou, "Confidential computing on NVIDIA Hopper GPUs: A performance benchmark study," arXiv:2409.03992. https://arxiv.org/abs/2409.03992, 2024.

Appendix A
Detailed Pseudocode

This appendix provides detailed pseudocode for the three core components of the `kvcache_attest` pipeline. Algorithm 1 describes the Promotion Gate verification logic at the decode node, Algorithm 2 presents the GPU-accelerated Blake3 hashing kernel, and Algorithm 3 details the attestation predicate construction at the prefill node. All pseudocode is simplified from the reference Python implementation for clarity.

Promotion Gate. The decode node's Promotion Gate enforces three sequential checks—hash integrity (G1), signature verification (G2), and policy evaluation (G3)—before admitting any KV block to GPU memory. A failure at any stage triggers block rejection and local prefill recomputation.

Algorithm 1 Promotion Gate Verification (cf. `gate.py`)

Require: Block tensors (K, V), attestation A, signature σ, expected session s_{exp}, trusted keys $\mathcal{K}$, policy P
Ensure: Verdict $\in \{\text{ADMIT}, \text{REJECT}\}$

```
{Check 1: Hash Integrity (G1)}
h ← HashBlock(A.block_id, K, V)
if h ≠ A.combined_hash then
    log "hash mismatch" to transparency log
    trigger local prefill recomputation
    return REJECT
end if
{Check 2: Signature Verification (G2)}
if σ.key_id ∉ K then
    log "unknown signing key" to transparency log
    trigger local prefill recomputation
    return REJECT
end if
if ¬ ECDSA.Verify(K[σ.key_id], A.json(), σ) then
    log "invalid signature" to transparency log
    trigger local prefill recomputation
    return REJECT
end if
{Check 3: Policy Evaluation (G3)}
if P.session_iso ∧ A.session_id ≠ s_exp then
    log "session isolation violation" to transparency log
    trigger local prefill recomputation
    return REJECT
end if
if P.trusted_models ≠ ∅ ∧ A.model_hash ∉ P.trusted_models then
    log "untrusted model hash" to transparency log
    trigger local prefill recomputation
    return REJECT
end if
return ADMIT
```

GPU-Accelerated Hashing. The GPU Blake3 kernel (Algorithm 2) avoids costly PCIe transfers of full tensors by hashing KV blocks directly on the GPU, then transferring only compact 32-byte digests back to the CPU for domain-separated final aggregation.

Algorithm 2 GPU-Accelerated KV Block Hashing via CUDA Blake3 Kernel (cf. `hasher.py`)

Require: Block ID b, key layers $K = [K_0, \ldots, K_{n-1}]$, value layers $V = [V_0, \ldots, V_{n-1}]$ resident on GPU
Ensure: Combined 256-bit block digest h_c

```
{Phase 1: Launch CUDA Blake3 kernel on GPU tensors}
allocate GPU output buffer D[0 ... 2n − 1] {32 B per tensor}
for i = 0 to n − 1 do
    partition K_i into ⌈|K_i|/1024⌉ chunks of 1024 B
    launch Blake3Kernel⟨grid, block⟩(K_i, D[2i]) {GPU stream}
    Each thread block: compress one chunk via Blake3
    Parallel reduction → single 256-bit digest D[2i]
    partition V_i; launch Blake3Kernel(V_i, D[2i + 1])
end for
cudaStreamSynchronize(hash_stream)
{Phase 2: Small DMA of digests only (not full tensors)}
D_cpu ← cudaMemcpyAsync(D, 32 × 2n bytes, D2H, pinned)
cudaStreamSynchronize(copy_stream)
{Phase 3: Domain-separated final hash on CPU}
for i = 0 to n − 1 do
    h_i ← Blake3(LAYER_TAG ‖ i ‖ D_cpu[2i] ‖ D_cpu[2i + 1])
end for
h_c ← Blake3(BLOCK_TAG ‖ b ‖ h_0 ‖ ··· ‖ h_{n−1})
return h_c
```

Attestation Predicate Construction. Algorithm 3 shows how the prefill node constructs an in-toto attestation predicate that cryptographically binds the KV block's digest to its session, model, and builder metadata, then signs it with ECDSA-P256.

Algorithm 3 Attestation Predicate Construction (cf. `predicate.py`)

Require: Block ID b, combined hash h, prompt tokens prompt_tokens, session ID s, builder info $\mathcal{B}$

Ensure: Signed attestation (A, σ)

$h_p \leftarrow \text{BLAKE3}(\text{prompt_tokens})$ {Hash of input prompt}

$\mathcal{B}$.model_hash $\leftarrow$ cached Blake3 digest of model weights

$t \leftarrow$ `UTC.now()`

build in-toto predicate A:

  subject: (b, h) {Block identity}

  builder: $(\mathcal{B}.\text{id}, \mathcal{B}.\text{model_hash}, \mathcal{B}.\text{engine})$

  session: s; *prompt_hash*: h_p

  metadata: (started_on= t, finished_on= t)

payload $\leftarrow$ $\text{CANONICALJSON}(A)$ {Sorted keys, no whitespace}

$\sigma \leftarrow \text{ECDSA-P256.SIGN}(\text{sk}, \text{payload})$

$\text{TRANSPARENCYLOG.APPEND}(\text{payload})$ {Async, non-blocking}

return (A, σ)