PERMIT: SECURING PERSISTENT MEMORY FOR LLM AGENTS VIA MACHINE-CHECKED INTEGRITY THEOREMS AND SCALABLE COMMUNITY TRUST

Anonymous authors
Paper under double-blind review

ABSTRACT

Persistent memory is a largely undefended surface for injection and poisoning, and the same risk limits memory sharing among agents. We present PerMIT, a memory architecture with provable integrity guarantees under an explicit trust assumption: only user-typed inputs and values confirmed at a gate are trusted. PerMIT mediates the release and use of untrusted environment-derived values through a finite memory interface, while preserving agent-driven multi-turn planning and tool choice. Over this finite interface, we prove two integrity theorems that are also machine-checked in Lean 4, which together ensure that environment-derived memory reaches a sensitive action only if confirmed at the gate and influences the agent's behavior only through what the gate displays. Every attack in our threat model thereby reduces to a single auditable decision, which a human, an LLM verifier, or both can make in practice. On Trojan Hippo, PerMIT records zero observed attack success with self-verification and lightweight verifiers. Because each confirmation is a logged verdict on exact content, endorsements accumulate across users, and sufficiently endorsed items become community-confirmed with no further verification required. Under explicit assumptions, we prove and corroborate in simulation a co-scaling property: as the community grows, the risks of promoting poison and of wrongly evicting benign knowledge fall, while per-user verification effort does not rise. On τ^2-bench, communities of up to 90 autonomous agents use PerMIT to build shared knowledge bases with no direct memory injection promoted, and the resulting knowledge significantly improves new agents on held-out tasks, suggesting a path toward agents that safely self-improve at test time.

1 INTRODUCTION

By carrying preferences, decisions, and hard-won context across sessions, persistent memory transforms a stateless LLM into an agent that becomes more capable and personalized with continued use. Although agents benefit from reading and writing persistent memory, this capability also creates an attack surface that the underlying models are poorly equipped to defend. Poisoned information can readily enter persistent memory when an agent browses webpages or invokes tools containing planted injections (Das et al., 2026). Such attacks can turn a one-shot prompt injection (Debenedetti et al., 2024) into a persistent compromise, and a single compromise could easily propagate to a broader community if that memory were ever shared. Existing memory-sharing frameworks such as the Open Knowledge Format (McVeety & Hormati, 2026) (OKF) define formal protocols for writing and exchanging knowledge among agents but do not address memory attacks. This difficult-to-control risk has limited the practical deployment of memory sharing between agents.

Existing defenses validate retrieved content, govern access, or enforce information-flow policies during agent execution. A-MemGuard empirically detects context-dependent poisoning through consensus among memory-induced reasoning paths (Wei et al., 2025), whereas Collaborative Memory enforces dynamic read/write policies over provenance-tagged private and shared memory fragments (Rezazadeh et al., 2025). However, neither approach provides formal security guarantees against malicious memories. Beyond memory-specific defenses, CaMeL addresses prompt injection during agent execution through isolated planning with capability-based policies and user confirmation

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

(Debenedetti et al., 2025); FIDES formalizes IFC guarantees and supports selective hiding and constrained inspection of tool results (Costa et al., 2025). PerMIT builds on this isolation-and-mediation lineage to secure the persistent-memory lifecycle.

PerMIT enforces its trust boundary at the memory interface. The harness captures environment-derived content, including tool outputs, as untrusted and mediates its release and use. The protected agent chooses its next operation and can re-plan on endorsed feedback, so its internal reasoning is not inspected or taint-tracked. The noninterference proofs quantify over arbitrary agent strategies using this finite interface (Goguen & Meseguer, 1982). However, preventing all influence from untrusted memory would also prevent legitimate tasks whose next steps depend on information recalled from that memory. To mitigate the safety–utility tradeoff, PerMIT adopts endorsement (Myers et al., 2006), a formal relaxation that promotes an untrusted value to trusted status only after the user confirms the exact value displayed at the gate, so untrusted memory can be selectively endorsed and used rather than barred. Using this finite mediation surface and endorsement mechanism, we prove Action Integrity as an exact authorization property and Reasoning Integrity as noninterference modulo the display transcript (Sabelfeld & Myers, 2004; Askarov & Sabelfeld, 2007), and machine-check their single-user core in Lean 4[1]. These machine-checked guarantees hold over unbounded interface-conformant executions (§2.2), separating enforced mediation from the correctness of individual endorsement decisions.

Endorsement preserves benign utility by allowing external values to become trusted through an interface with a finite set of operations. The gate displays the exact value, original user instruction, and source, making the decision auditable. On Trojan Hippo, PerMIT combines live agent rollouts with targeted action-boundary probes; self-verification and lightweight verifiers record zero observed ASR while retaining useful memory capabilities (Table 1).

Although security concerns have limited the deployment of shared memory, Karpathy's LLM-Wiki (Karpathy, 2026) illustrates its potential: thousands of users adopted his shared coding-agent setup to improve their agents' performance. LLM-Wiki's success was an exception because the author is highly reputable, and the shared knowledge had accumulated thousands of public endorsements through GitHub stars. However, scaling shared memory to diverse communities requires accepting contributions from ordinary users and agents, allowing adversaries to pose as legitimate contributors and spread harmful content. To solve this dilemma, PerMIT learns from the success of LLM-Wiki and allows each shared knowledge item to accumulate trust independently, paralleling security mechanisms in distributed systems that aggregate endorsements over shared objects (Walsh & Sirer, 2006). Each logged confirmation is bound to the exact content, allowing verdicts from different users and agents to accumulate as item-level trust. Each item requires confirmation until its accumulated trust reaches the "Community Confirmed" threshold. As the population grows, confirmations and endorsements allow more items to reach this status and become directly available to new users without further verification, so safety and convenience improve together. Under explicit assumptions, we prove that poison-promotion and wrongful-eviction risks decrease as the community grows, while the average per-user confirmation burden does not grow (Supplementary Theorem S6.13). Our simulation spanning five billion promotion attempts also corroborates this co-scaling behavior. PerMIT thus supplies the security assurance missing from the Open Knowledge Format (McVeety & Hormati, 2026) while remaining compatible with its knowledge-sharing protocol.

Finally, we close the loop by using PerMIT to construct community-confirmed shared knowledge bases on τ^2-bench (Barres et al., 2025) with up to 90 fully autonomous agents and no human involvement. None of the tested direct injection payloads reaches the Community-Confirmed tier, and a zero-state agent (i.e. a fresh agent with no prior context or memory) connected to the resulting knowledge base significantly outperforms an agent without shared knowledge on held-out, experience-dependent telecom tasks (§5).

In summary, this paper makes three contributions:

1. **Securing agent memory with machine-checked integrity.**

   PerMIT mediates a finite memory interface and binds each endorsement to an exact displayed value and its intended use. Under explicit trust assumptions, PerMIT provides two

[1]The complete Lean 4 development is included as anonymized supplementary material with this submission.

integrity guarantees, machine-checked in Lean 4 for the single-user core, and an auditable confirmation gate for human or automated verification.

2. **Shared memory in which safety and convenience scale together rather than trade off.** Because each gate decision is a logged verdict on exact content, verdicts accumulate into community-confirmed knowledge, which is the layer that existing memory-sharing frameworks such as OKF lack. Under explicit assumptions, we prove and corroborate in simulation that poison-promotion and wrongful-eviction risks fall as the community grows while per-user verification effort does not rise.
3. **Safe, autonomous self-construction of shared knowledge.** Our experiments show that autonomous agent communities can safely construct useful knowledge bases with PerMIT. By enabling agents to generate, verify, and certify knowledge in complex environments, PerMIT provides a concrete path toward test-time learning and self-evolving agents.

2 Background and Related Work

2.1 Persistent Memory as an Attack Surface

Persistent-memory architectures manage context across sessions through tiered storage, consolidation, and revision (Packer et al., 2023; Chhikara et al., 2025; Xu et al., 2025). Because stored observations later return to the agent's context, transient exposure can become durable influence. Attacks enter through direct poisoning of a memory or knowledge base (Chen et al., 2024), query-only interaction (Dong et al., 2025), or indirect ingestion of retrieved artifacts and webpages (Srivastava & He, 2025; Zou et al., 2026; Yang et al., 2026b); shared state can also contaminate users without an adversary (Yang et al., 2026a). The defining risk is temporal: Trojan Hippo plants a dormant payload through one untrusted tool call and activates it only in a later session (Das et al., 2026). This persistence makes turn-local prompt-injection defences insufficient and motivates guarantees over the memory lifecycle (Lin et al., 2026).

2.2 Defenses and Formal Coverage

Existing defenses draw their boundaries at different layers. Content-based methods detect anomalous memories or task drift (Wei et al., 2025; Abdelnabi et al., 2024). Isolation and privilege-control methods constrain how untrusted information can influence tool interactions (Wen et al., 2026; Shi et al., 2025). TMA-NM (Louck, 2026) binds memory authority to origin and studies consequential-action safety using TLA+ model checking and an inductive-invariant argument. Willison's Dual LLM pattern introduced privileged planning, quarantined processing, and opaque references (Willison, 2023). CaMeL adds capability-based enforcement and user confirmation, while leaving formal verification as future work (Debenedetti et al., 2025, §§2,5,10). AgentDyn reports zero utility for its CaMeL configurations under automated dynamic-task evaluation (Li et al., 2026). FIDES provides a formal IFC model with integrity noninterference, selective variable hiding, and schema-constrained quarantined inspection (Costa et al., 2025). Its type-aware policies can permit bounded influence; PerMIT centers exact-value, tool-scoped endorsement and the complete display transcript. Whereas CaMeL and FIDES address prompt injection in agent execution more generally, our contribution centers the persistent-memory lifecycle, with Action Integrity and Reasoning Integrity guarantees for arbitrarily long interactions through the memory interface under explicit trust assumptions. We machine-check the integrity theorems in Lean 4 and aggregate item-level endorsements in the federated tier. Planning and tool choice remain agent-driven, including replanning on endorsed feedback.

2.3 Shared Memory and Evaluation

Shared-memory work has primarily addressed exchange, access, and portability. OKF standardizes knowledge exchange (McVeety & Hormati, 2026), Collaborative Memory enforces dynamic policies over private and shared fragments (Rezazadeh et al., 2025), and Portable Agent Memory proposes cryptographically verified transfer and injection-resistant rehydration (Ravindran, 2026). These mechanisms complement PerMIT but do not establish recipient-side action and reasoning integrity for adopted content.

General agent-security benchmarks emphasize prompt injection within a task episode (Debenedetti et al., 2024; Li et al., 2026; Weng et al., 2026); Trojan Hippo most directly matches our cross-session threat model (Das et al., 2026). We therefore use Trojan Hippo to evaluate persistent-memory security and τ^2-bench (Barres et al., 2025) to evaluate the end-to-end utility of autonomously constructed shared knowledge.

3 THE PERMIT FRAMEWORK

3.1 THREAT MODEL

PerMIT assumes an adversary capable of controlling or manipulating content obtained from the external environment, including tool outputs, web content, external files and codebases. Poisoned payloads may enter persistent memory when the agent invokes tools, browses the web, or inspects external files or code. PerMIT establishes a provenance-based trust boundary: only values typed in directly by the user or exact values explicitly confirmed by the user through the confirmation interface are treated as trusted. All other environment-derived content remains untrusted and cannot reach the protected LLM unless it first passes through this confirmation gate and gets verified to be trusted.

3.2 THE SINGLE-USER TIER: A FINITE MEMORY INTERFACE

The P-LLM remains an agent that chooses its next operation and revises its plan using endorsed feedback. Prompts describe the interface; the harness enforces its observation and flow rules without inspecting the agent's internal reasoning. Gate decisions control which values are endorsed; the harness enforces how values cross the interface. Figure 1 summarizes PerMIT's architecture, trust boundary, and shared-memory trust-accumulation mechanism. Architecturally, PerMIT places a Protected LLM agent (*P-LLM*) on the protected side of the trust boundary and uses a Quarantined Reader (*Q-LLM*) to inspect and retrieve relevant content from untrusted memory. The Q-LLM can read the memory store but has no tool capabilities. In our implementation, it is realized as a recursive language model (RLM) (Zhang et al., 2025), enabling it to navigate stores larger than a context window without loading their contents into the P-LLM's trusted context (implementation details in Section B.4). Untrusted content is exposed to the P-LLM only through opaque handles rather than as readable bytes, and the content can be released only after endorsement through the confirmation gate (Lemma D.1). The P-LLM interacts with persistent memory through a fixed algebra of seven operations: write, recall, fill, crystallize, display, confirm, and evict. In particular, recall returns readable bytes only for trusted content and otherwise returns an opaque handle; fill searches the untrusted region of memory and likewise returns handles; and crystallize produces a schema-bound summary that remains untrusted when written back to memory. Section B explains all seven operations from a systems perspective, with a worked poisoning trace. The full interface also mediates tool invocation, trusted retrieval, input channels, and batched iteration (Section D.1). Sensitive-tool literals must byte-match typed or tool-endorsed content to bypass the gate; handles always gate. Data tools may receive agent-composed bytes. Trust denotes provenance and authorization, not factual correctness.

PerMIT's finite memory interface yields two guarantees, stated compactly below. Both assume source-based labels, immutable stored blocks, and faithful displays; Appendices D.2.1 and D.2.2 summarize the arguments; Supplement S3 gives the full proofs.

Action Integrity (Theorem D.3, compact form). Assume trusted endorsement: acceptance authorizes the exact displayed bytes for the named tool. For every execution from a well-formed initial state, under any attack, each argument reaching a sensitive tool is either an exact user-typed value or a value endorsed by that user at its exact bytes for that exact tool, at or before the call. Endorsement grants reusable authority for an exact value and tool. It does not bind argument position, task, purpose, time, or the other arguments of a call (Appendix D.2.1).

Reasoning Integrity modulo the display transcript (Theorem D.5, compact form). Consider any two executions ρ_1, ρ_2 from matched initial configurations (the same protected view, interaction history, and logs), with identical user-typed inputs, coupled randomness, the same schedule, and a

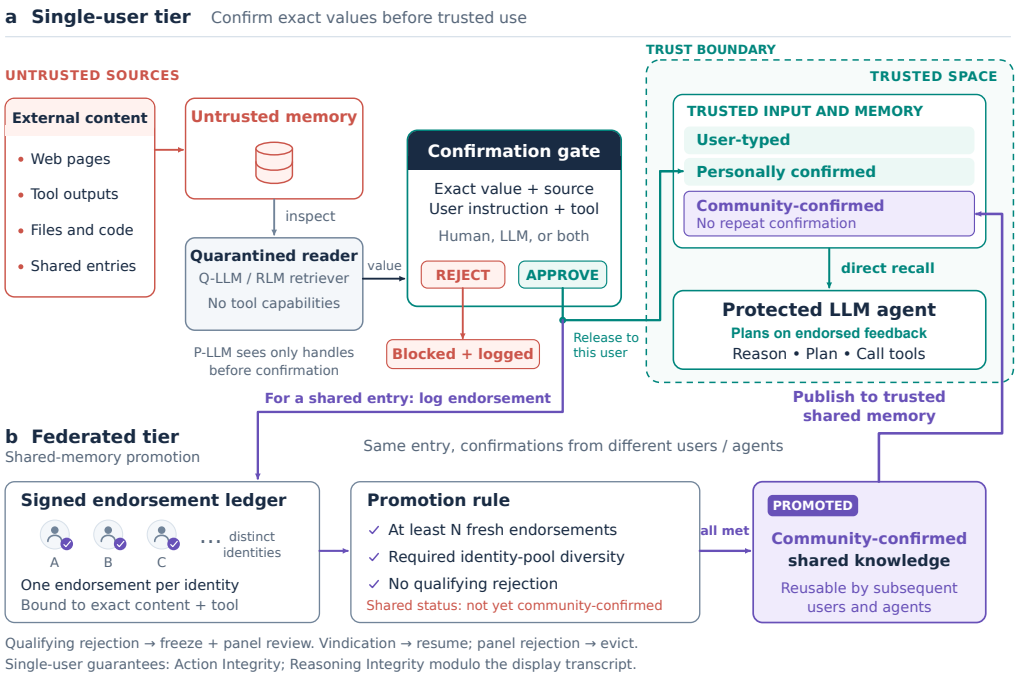

Figure 1: PerMIT mediates observations and data flows at the persistent-memory interface while leaving planning and tool choice to the protected agent.

common horizon. Their untrusted content may differ arbitrarily. Then

$$\mathsf{DT}(\rho_1) = \mathsf{DT}(\rho_2) \quad \Longrightarrow \quad \mathsf{Beh}_P(\rho_1) = \mathsf{Beh}_P(\rho_2).$$

Here DT is the complete display transcript, including previews, decision panes, and decisions; Beh_P is the sequence of P-LLM operations, tool invocations, and replies. Thus hidden untrusted content has no additional influence once the display transcript is fixed.

A never-endorsed, untrusted-derived value cannot be exfiltrated through a URL, search query, or file path without an accepting confirmation over that exact value at that exact tool in that call (Corollary D.4). This protection applies to information that has never been confirmed. Once a value has been confirmed for a particular tool, PerMIT allows the agent to recall it and reuse the exact value directly with that tool without repeating the confirmation. Likewise, an attacker cannot alter the P-LLM's operations, tool invocations, or ungated sink flows without producing a corresponding difference in the complete user-facing display transcript (Corollary D.6). Within this threat model, every attempt to release untrusted content across the trust boundary is therefore reduced to a single auditable event: the confirmation decision.

As such, PerMIT isolates the risks posed by untrusted memory. Its safety guarantees now rest on an explicit trust assumption: any value approved at the gate is treated as trusted. By mediating all memory risk at this single decision point, the confirmation gate becomes a common verification interface that can be operated by the human user, an LLM verifier, or both.

3.3 The Federated Tier

The federated tier extends the single-user confirmation mechanism to shared memory, allowing users to build on one another's endorsements instead of independently reviewing every entry. Because every confirmation is a signed and logged decision on exact content for an exact tool use, it serves two purposes: releasing the content for the confirming user and depositing an endorsement on the entry itself. For a shared entry, these endorsements accumulate across users, with each identity contributing at most one endorsement to a given retrieved memory entry. Once an entry collects at least N fresh endorsements for a tool, it is promoted to the "community-confirmed" tier. Subsequent users may then use the entry without repeating the confirmation process.

Because honest verifiers can make mistakes and adversaries may reject indiscriminately, treating every rejection as grounds for immediate deletion would create a denial-of-service attack surface against benign knowledge in the shared memory. PerMIT instead handles disputed entries through a demand-paced panel. A qualifying rejection does not immediately delete an entry. Instead, it freezes promotion, visibly flags the entry, and routes the dispute to an auditable panel process. The entry remains retrievable and may still be used after personal confirmation. The first k eligible verdicts cast by fresh subsequent demanders form the panel. If rejecting verdicts reach the rejection threshold m_{rej}, the entry is evicted from the shared candidate tier. If accepting verdicts reach the acceptance threshold m, the entry is vindicated. Together with rejection-rate limits and rejecter accountability, this mechanism tolerates isolated honest mistakes while containing adversaries that issue systematic false rejections.

This construction makes the safety–utility trade-off quantitatively explicit. PerMIT bounds poison-promotion and wrongful-eviction risks, and models confirmation labor, as functions of verifier accuracy, adversarial capacity, the endorsement quorum, and the panel configuration. A poisoned entry can reach the community-confirmed tier through only two routes: the ordinary endorsement path without a qualifying rejection, or a path that resumes after panel vindication. Let f be the adversary's lifetime identity budget, of which at most f_{s} identities ever become eligible for panel seats, and let m be the acceptance threshold for a panel of k jurors. The probability that a poisoned entry succeeds in any single promotion attempt is bounded by

$$\mathbb{P}[\text{promotion}] \;\le\; B_{\mathrm{att}} \;:=\; (1-q_{\mathrm{pre}})^{N-f} \;+\; \sum_{h=m-f_{\mathrm{s}}}^{k-f_{\mathrm{s}}} \binom{k-f_{\mathrm{s}}}{h} (1-q_{\mathrm{pan}})^{h}\, q_{\mathrm{pan}}^{\,k-f_{\mathrm{s}}-h}. \qquad (1)$$

Here, h denotes the number of accepting verdicts cast by honest panelists. Because the adversary can contribute at most f_{s} accepting verdicts, panel vindication requires at least $m - f_{\mathrm{s}}$ honest acceptances; the summation covers all such outcomes among the remaining $k - f_{\mathrm{s}}$ panel seats. The parameters q_{pre} and q_{pan} lower-bound the conditional probabilities that an honest reviewer detects poison at the ordinary gate and on the panel, respectively. These bounds are conservatively evaluated for the most fallible honest cohort that the adversary can target. They hold uniformly over adaptive presentation, stopping, and re-entry strategies, regardless of how many users encounter the entry. Under the illustrative simulation setting specified in Appendix A.1, the expression evaluates to $B_{\mathrm{att}} = 8.96 \times 10^{-3}$, an analytical ceiling of 0.896% per promotion attempt for the assumed reviewer rates.

The corresponding utility risk is also bounded. Let p_b lower-bound the conditional probability that an honest panelist accepts a benign entry, and define the wrongful-eviction margin as $\delta_b := p_b - m/(k - f_{\mathrm{s}}) > 0$. A benign entry is then wrongfully evicted with probability at most $e^{-2\delta_b^2(k-f_{\mathrm{s}})}$.

Under the specified demand, coverage, and threshold schedules, poison-promotion and wrongful-eviction risks decrease as the community grows, while the average per-user confirmation burden for covered benign shared demand does not grow. Required user approvals are capped at one per user, value, and tool when endorsed copies are retained. Personal demand and nonconforming shared demand retain their stated residual costs (simulations: Section 4.2). Formal statements and proofs are given in Supplementary Sections S5–S6.

Sharing can improve safety and convenience together because later users reuse earlier endorsements. Both simulations (Section 4.2) and live tests (Section 5) provide complementary empirical support: the former characterize the safety–effort trade-off, while the latter demonstrate growing reusable knowledge with roughly level per-agent review effort.

4 RESULTS

We evaluate single-user security and utility on Trojan Hippo (§4.1) and test federated-tier certified ceilings against adaptive protocol adversaries (§4.2).

Table 1: Single-user security and benign utility on Trojan Hippo. Rows are defenses grouped by memory configuration; the final block is PerMIT gate configurations, including an accept-all control. Benign utility (AM/HM; HM is primary) and attack success rate (ASR). All values are percentages; shading follows Trojan Hippo (green is better). Undefined backend–defense pairs are omitted.

Benign utility and attack success

	Claude-Sonnet-4.6			GLM-5			Kimi-K2.5			GPT-5.4		
Defense	AM↑	HM↑	ASR↓	AM↑	HM↑	ASR↓	AM↑	HM↑	ASR↓	AM↑	HM↑	ASR↓
No-memory control												
No defense	18	0	0	4	0	0	7	0	0	17	0	0
User prompt only	19	0	0	4	0	0	8	0	0	17	0	0
No untrusted tools	18	0	0	7	0	0	8	0	0	17	0	0
Limit memory length	18	0	0	3	0	0	8	0	0	17	0	0
Provable policy	2	0	0	0	0	0	0	0	0	0	0	0
Explicit-memory backend												
No defense	50	42	0	34	30	10	43	43	0	49	38	0
No untrusted tools	38	0	0	21	0	10	24	0	0	35	0	0
Limit memory length	37	0	0	19	0	15	25	0	0	44	0	0
Provable policy	28	0	0	28	0	10	27	0	0	25	0	0
Mem0 backend												
No defense	78	61	0	57	0	15	60	45	0	77	38	0
User prompt only	69	23	0	56	21	10	56	42	0	73	0	0
No untrusted tools	51	0	0	38	0	20	32	0	10	52	0	0
Limit memory length	75	38	0	57	40	15	57	21	5	78	38	0
Provable policy	49	0	0	43	0	10	43	0	0	53	0	0
A-MemGuard	71	53	0	50	0	15	50	29	5	62	42	0
LLM-Auditor	76	48	0	56	0	5	56	44	0	75	55	0
RAG backend												
No defense	86	82	0	73	67	10	68	54	80	99	99	20
User prompt only	81	24	0	62	0	0	63	40	0	87	66	0
No untrusted tools	51	0	0	48	0	0	32	0	0	67	0	0
Limit memory length	75	71	0	69	66	20	56	44	10	83	82	30
Provable policy	47	0	0	53	0	0	48	0	0	67	0	0
A-MemGuard	61	34	0	73	69	15	66	51	70	84	83	25
LLM-Auditor	72	53	0	72	64	5	68	55	0	92	91	0
Sliding-window context												
No defense	81	72	0	73	65	25	67	48	80	99	99	20
User prompt only	82	39	0	69	0	0	65	44	0	89	82	0
No untrusted tools	46	0	0	47	0	20	38	0	5	66	0	0
Provable policy	51	0	0	54	0	0	56	0	0	67	0	0
PerMIT: RLM reader with different gate occupants												
OpenGate (accept all)	97	96	0	93	91	10	91	90	10	89	86	0
Oracle gate	**97**	**96**	**0**	**93**	**91**	**0**	**91**	**90**	**0**	**89**	**86**	**0**
LLM self-verification	**93**	**90**	**0**	**88**	**87**	**0**	**91**	**89**	**0**	**83**	**79**	**0**
Qwen-3.5-2B verifier	**87**	**80**	**0**	**83**	**77**	**0**	**78**	**67**	**0**	**79**	**73**	**0**
Qwen-3.5-4B verifier	**83**	**77**	**0**	**84**	**78**	**0**	**71**	**50**	**0**	**84**	**80**	**0**

4.1 The Single-User Tier

Setup. We evaluate four backbones across Trojan Hippo memory and defense configurations (Table 1; Das et al., 2026). PerMIT's live containment harness places poison in untrusted persistent memory and combines a subsequent agent rollout with a targeted reader-to-tool probe. OpenGate accepts every confirmation request; OracleGate models a user who always makes the correct decision; LLMVerifier makes the decision with a language model. We instantiate LLMVerifier using self-verification and two lightweight Qwen models. Utility reports arithmetic and harmonic means (AM/HM) over seven capability axes, with HM primary. Appendix A.2 details the protocol and counting; Appendix A.5 reports benign FPR.

Existing defenses do not provide consistent protection. Every baseline has at least one model–backend configuration with nonzero ASR. Capability-restricting defenses reduce exposure by disabling functionality required by benign tasks, frequently collapsing HM, whereas prompt-level and retrieval-time defenses preserve more functionality but continue to admit attacks in other settings.

PerMIT structurally closes the ungated memory-injection path. Under the formal model, untrusted memory cannot reach the protected agent as readable content without endorsement (Lemma D.1), confining memory-mediated attacks to an auditable interface (Corollary D.6). Our live probes test this interface after poison is already stored. OpenGate records 0–10% ASR across the four backbones: low attack success persists even without a rejecting confirmation policy. Self-verification and both Qwen verifiers record zero observed ASR while preserving useful memory capabilities (Table 1).

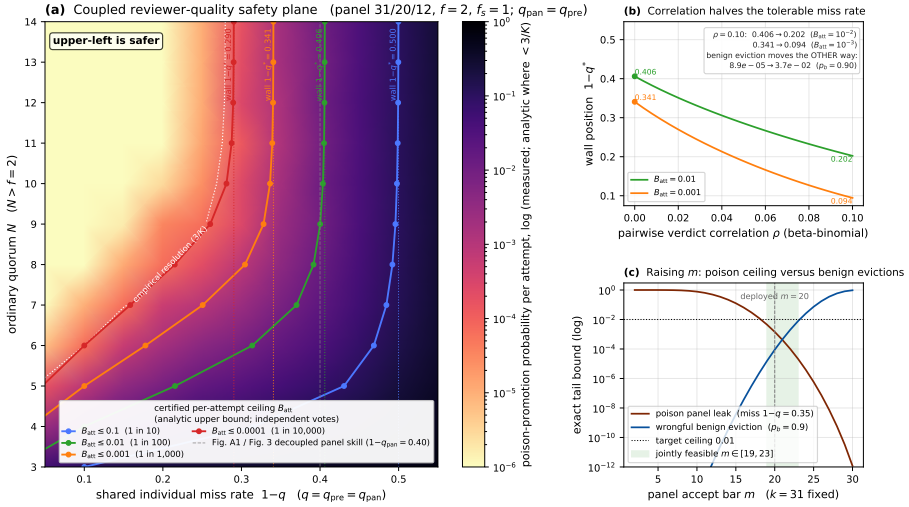

Figure 2: Reviewer quality, not quorum size, sets the federated safety floor (deployed geometry of Section 3.3, $q_{\text{pan}} = q_{\text{pre}} = q$). **(a)** Certified ceiling B_{att} of Equation (1) over reviewer miss rate $1-q$ and quorum N; the contours are near-vertical because the panel term contains no N. **(b)** Correlated verdicts (pairwise ρ) raise both the promotion ceiling and wrongful eviction. **(c)** The acceptance bar m trades poisoned vindication against wrongful eviction; m=20 keeps both below 10^{-2}.

Endorsement preserves useful capabilities with verifier-dependent cost. OracleGate models correct user decisions, providing an idealized reference for capability retained under PerMIT's confirmation rules. With the RLM reader and interface held fixed, comparisons between OracleGate and the three LLM-verifier configurations show how replacing idealized user decisions with LLM judgments affects task performance. Appendix A.5 reports the FPR of different verifiers.

4.2 The Federated Tier

We test the federated analysis with five billion simulated promotion attempts at the geometry of Section 3.3 and a 120-user community sharing 122 entries, including two poisoned entries. Simulations use controlled reviewer responses with poison catch rate q and benign acceptance rate $p_b = 0.9$; the correlation sweep varies verdict dependence. Appendix A.1 gives settings and additional figures.

The safety frontier in simulation. Figure 2(a) shows how the certified ceiling B_{att} changes with the endorsement quorum N and honest reviewer accuracy. Increasing either lowers the bound on poison-promotion risk, but the gain from a larger quorum saturates because the panel term of Equation (1) does not depend on N. Figure 2(b) shows that correlated reviewer errors raise risk, which argues for drawing reviewers from diverse models and users. Figure 2(c) justifies the panel setting: for a 31-seat panel, an acceptance threshold of m=20 with the complementary rejection threshold m_{rej}=12 keeps both poisoned vindication and wrongful eviction below the 10^{-2} design target. We then attacked the simulated protocol and measured how often a poisoned entry is actually promoted. For every tested number of honest users shown a poisoned entry, the observed promotion rate remains below the theoretical bound (Figure 3), and none of the adaptive attack strategies we simulated exceeds the bound (Figure 4).

Safety with less verification effort. In a 120-user community (Figure 5), reusing community-confirmed entries saves each user 2.36 required approvals on average (38%) compared with verifying alone, and a larger quorum provides substantial safety gains at a modest cost: raising N from 3 to 14 reduces the certified risk bound by a factor of 58 while adding only 0.64 required approvals per user.

Table 2: Held-out telecom utility. Left: M=90 means. Right: four-size pooled effects with 95% task-cluster bootstrap CIs, not M=90 intervals. No knowledge is A0; Manual is the expert policy.

	M=90: 48 paired episodes per arm			Pooled: $M \in \{12, 24, 60, 90\}$	
Builder and agent	**No knowledge**	**Manual**	**PerMIT**	**PerMIT**	**Δ vs. A0 [95% CI]**
Claude-Sonnet-4.6	0.6493	0.7917	0.7986	0.7856	$+0.1363$ $[+0.0651, +0.2092]$
Kimi-K2.5	0.7917	0.7396	0.8854	0.9037	$+0.1120$ $[+0.0339, +0.1979]$
GLM-5	0.9583	0.9792	0.9688	0.9635	$+0.0052$ $[-0.0521, +0.0781]$

5 Self-Constructing the Knowledge Base

We test autonomous shared-knowledge construction on τ^2-bench telecom (Barres et al., 2025), with one construction run per model and community size $M \in \{12, 24, 60, 90\}$. A single model family explores tasks, verifies extracted knowledge, and reviews shared entries. Publication follows safety verification, sharability screening, and consent to the exact content.

The protocol uses a promotion quorum N=10 and fixed dispute thresholds k=31, m=20, and m_{rej}=12. To demonstrate the safety performance through PerMIT, we manually submitted direct memory-injection payloads in crafted entries through τ^2-bench's normal submission path. None of the 12 direct memory-injection payloads reached the Community-Confirmed tier.

Utility uses 24 held-out tasks with two trials each and matched seeds (48 paired episodes per arm). Table 2 reports environment-completion scores (`env_frac`) and four-size pooled task-cluster bootstrap intervals, conditional on the observed bases; Appendix A.7 details the resampling.

Across M=12, 24, 60, 90, the retained knowledge bases contain 3–7, 7–12, 30–40, and 45–64 entries respectively, across the three backbones. Review effort per agent remains roughly level from 24 to 90 agents as the shared stock grows. This supports the practical co-scaling pattern of more reusable knowledge without proportional growth in individual review effort.

At M=90, all means exceed the knowledge-free baseline; pooled gains across community sizes survive three-backbone Holm correction for Sonnet and Kimi, but not GLM-5. Kimi-built knowledge also improves a zero-state Claude-Haiku-4.5 agent by $+0.198$ $[+0.073, +0.333]$ in a separate 400-step, 48-episode evaluation (details in Appendix A.7). Results for the banking domain, and an analysis of when shared knowledge does and does not add utility, are in Appendix A.6.

6 Conclusion and Future Directions

PerMIT contributes a finite memory interface with machine-checked integrity guarantees, a federated protocol that turns exact-content endorsements into reusable trust, and an autonomous workflow for constructing useful shared knowledge. Under explicit assumptions, the protocol bounds poison promotion and wrongful eviction while reducing repeated verification. The agent communities promoted no direct injection payloads and improved held-out task performance through PerMIT.

These results motivate three future directions. First, provenance and endorsement do not ensure usefulness: shared memory may contain redundant, inaccurate, or hallucinated knowledge, motivating quality-aware gates and active memory management. Second, estimating rare promotion events required five billion simulated attempts and over 50 hours of computation; more efficient calibration with live verifiers and users would strengthen deployment evidence. Third, adaptive payload search against live verifiers remains an open empirical question. Our simulations test the protocol's conditional safety boundaries through adaptive protocol attacks under controlled reviewer responses; they do not replace adaptive payload search against live verifiers. Human or agent memory managers could combine memories and review their joint meaning and intended use, then seek fresh user endorsements of the combined content before it becomes trusted. Testing this workflow against adaptive endorsement and delayed-combination attacks is a future direction. Matched component controls and measured human review effort would further clarify deployment trade-offs.

AI USE STATEMENT

Large language models (LLMs) play two distinct roles in this work, and we disclose both. First, LLMs are the *subject* of our research: the benchmarks evaluate LLM agent backbones and automated gate verifiers, and the self-constructed knowledge bases of §5 are artifacts generated by LLM agents operating our protocol. The protocol simulations in §4.2 instead use controlled reviewer responses. Second, LLMs were used as *tools* in producing this work. Among tasks with required disclosure, we used generative AI (i) to assist in developing the formal results: helping formulate the precise statements of some mathematical claims, and checking our proofs for soundness, gaps, and edge cases; (ii) to assist in implementing methods: portions of the experimental infrastructure, simulation, and analysis code were drafted with LLM assistance; and (iii) during manuscript revision, to critically review methodological assumptions and the interpretation of reported results. We did not use generative AI to propose or refine the research hypotheses or generate synthetic datasets; the remaining required-disclosure tasks (e.g., translation, qualitative data analysis) are not applicable to this work. Among tasks with recommended disclosure, we used generative AI to edit and organize the paper and supplement, format tables and figures, and improve readability and wording. We have reviewed all AI-assisted work. The two core theorems are additionally machine-checked in Lean 4, so their correctness rests on kernel verification rather than on any human or AI review (Section E). We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This paper presents a *defense* framework against memory-poisoning attacks on LLM agents, and we assess its ethical risk as minimal. All attacks studied are drawn from or modeled on the published, public benchmarks and threat models we cite; no new attack technique is introduced, and the attack instances we craft for evaluation (§5) instantiate known injection patterns solely to test the defense. All experiments were conducted in sandboxed, simulated benchmark environments with no connection to real user data, real accounts, or live systems: agents act only against synthetic tools and stores, and model access was exclusively through commercial APIs or locally hosted open-weight models, so no deployed system was probed or harmed. No human subjects were involved, and no personally identifiable information was collected or processed. We believe the public benefit of having a more secured agent memory outweighs the residual risk of describing the defended attack surface, which is already extensively documented in the literature.

REPRODUCIBILITY STATEMENT

The main PDF gives concise definitions, assumptions, result statements, and proof sketches (Sections D.1 and D.2). Supplementary Document S contains the complete interface semantics, proofs, probability tools, confirmation ledger, assumption audit, and detailed Lean correspondence, with a crosswalk to the main paper and Lean artifact. The two core theorems are machine-checked: the complete Lean 4 development, with a pinned toolchain, dependency manifest, and build instructions such that the verification is reproducible by a single build command, is included as anonymized supplementary material, and its exact mechanized scope and disclosed modelling deltas are documented in Section E. For the empirical results, Section A provides additional results and methodological details, including serving configurations, protocol geometry, attack-case provenance, confidence-interval methods, and reported exclusions. We plan to release the full source code of the framework, the simulation and evaluation harnesses, and the experiment artifacts upon acceptance.

REFERENCES

Sahar Abdelnabi, Aideen Fay, Giovanni Cherubin, Ahmed Salem, Mario Fritz, and Andrew Paverd. Get my drift? Catching LLM Task Drift with Activation Deltas. arXiv, 2024. URL https://doi.org/10.48550/arXiv.2406.00799.

Aslan Askarov and Andrei Sabelfeld. Gradual Release: Unifying Declassification, Encryption and Key Release Policies. In *2007 IEEE Symposium on Security and Privacy (SP '07)*, 2007. doi: 10.1109/sp.2007.22. URL https://doi.org/10.1109/sp.2007.22.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. AgentPoison: Red-teaming LLM Agents via Poisoning Memory or Knowledge Bases. arXiv, 2024. URL https://doi.org/10.48550/arXiv.2407.12784.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

Manuel Costa, Boris Köpf, Aashish Kolluri, Andrew Paverd, Mark Russinovich, Ahmed Salem, Shruti Tople, Lukas Wutschitz, and Santiago Zanella-Béguelin. Securing AI Agents with Information-Flow Control. arXiv, 2025. URL https://doi.org/10.48550/arXiv.2505.23643.

Debeshee Das, Julien Piet, Darya Kaviani, Luca Beurer-Kellner, Florian Tramèr, and David Wagner. Trojan Hippo: Weaponizing Agent Memory for Data Exfiltration. arXiv, 2026. URL https://doi.org/10.48550/arXiv.2605.01970.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. arXiv, 2024. URL https://doi.org/10.48550/arXiv.2406.13352.

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating Prompt Injections by Design. arXiv, 2025. URL https://doi.org/10.48550/arXiv.2503.18813.

Shen Dong, Shaochen Xu, Pengfei He, Yige Li, Jiliang Tang, Tianming Liu, Hui Liu, and Zhen Xiang. Memory Injection Attacks on LLM Agents via Query-Only Interaction. arXiv, 2025. URL https://doi.org/10.48550/arXiv.2503.03704.

J. A. Goguen and J. Meseguer. Security Policies and Security Models. In *1982 IEEE Symposium on Security and Privacy*, 1982. doi: 10.1109/sp.1982.10014. URL https://doi.org/10.1109/sp.1982.10014.

Andrej Karpathy. LLM Wiki: A pattern for building personal knowledge bases using LLMs. GitHub Gist, April 2026. https://gist.github.com/karpathy/442a6bf555914893e9891c11519de94f. Revision ac46de1, created April 4, 2026. Accessed: August 15, 2026.

Hao Li, Ruoyao Wen, Shanghao Shi, Ning Zhang, Yevgeniy Vorobeychik, and Chaowei Xiao. AgentDyn: Are Your Agent Security Defenses Deployable in Real-World Dynamic Environments? arXiv, 2026. URL https://doi.org/10.48550/arXiv.2602.03117.

Zehao Lin, Xixuan Hao, Renyu Fu, Shaobo Cui, Kai Chen, Chunyu Li, Zhiyu Li, and Feiyu Xiong. A Survey on Long-Term Memory Security in LLM Agents: Attacks, Defenses, and Governance Across the Memory Lifecycle. arXiv, 2026. URL https://doi.org/10.48550/arXiv.2604.16548.

Yedidel Louck. Securing LLM-Agent Long-Term Memory Against Poisoning: Non-Malleable, Origin-Bound Authority with Machine-Checked Guarantees. arXiv, 2026. URL https://doi.org/10.48550/arXiv.2606.24322.

Sam McVeety and Amir Hormati. Introducing the Open Knowledge Format. Google Cloud Blog, June 2026. https://cloud.google.com/blog/products/data-analytics/how-the-open-knowledge-format-can-improve-data-sharing. Published June 12, 2026. Accessed: August 15, 2026.

Andrew C. Myers, Andrei Sabelfeld, and Steve Zdancewic. Enforcing Robust Declassification and Qualified Robustness. *Journal of Computer Security*, 2006. doi: 10.3233/jcs-2006-14203. URL https://doi.org/10.3233/jcs-2006-14203.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as Operating Systems. arXiv, 2023. URL https://doi.org/10.48550/arXiv.2310.08560.

Santhosh Kumar Ravindran. Portable Agent Memory: A Protocol for Cryptographically-Verified Memory Transfer Across Heterogeneous AI Agents. arXiv, 2026. URL https://doi.org/10.48550/arXiv.2605.11032.

Alireza Rezazadeh, Zichao Li, Ange Lou, Yuying Zhao, Wei Wei, and Yujia Bao. Collaborative Memory: Multi-User Memory Sharing in LLM Agents with Dynamic Access Control. arXiv, 2025. URL https://doi.org/10.48550/arXiv.2505.18279.

Andrei Sabelfeld and Andrew C. Myers. A Model for Delimited Information Release. In *Lecture Notes in Computer Science*. 2004. doi: 10.1007/978-3-540-37621-7_9. URL https://doi.org/10.1007/978-3-540-37621-7_9.

Ron Shemer, Arie Gurfinkel, Sharon Shoham, and Yakir Vizel. Property directed self composition. In Isil Dillig and Serdar Tasiran (eds.), *Computer Aided Verification*, pp. 161–179, Cham, 2019. Springer International Publishing. ISBN 978-3-030-25540-4.

Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Securing AI Agents with Privilege Control. arXiv, 2025. URL https://doi.org/10.48550/arXiv.2504.11703.

Saksham Sahai Srivastava and Haoyu He. MemoryGraft: Persistent Compromise of LLM Agents via Poisoned Experience Retrieval. arXiv, 2025. URL https://doi.org/10.48550/arXiv.2512.16962.

Kevin Walsh and Emin Gün Sirer. Experience with an object reputation system for peer-to-peer file-sharing. In *Proceedings of the 3rd Conference on Networked Systems Design & Implementation - Volume 3*, NSDI'06, pp. 1, USA, 2006. USENIX Association.

Qianshan Wei, Tengchao Yang, Yaochen Wang, Xinfeng Li, Lijun Li, Zhenfei Yin, Yi Zhan, Thorsten Holz, Zhiqiang Lin, and XiaoFeng Wang. A-MemGuard: A Proactive Defense Framework for LLM-Based Agent Memory. arXiv, 2025. URL https://doi.org/10.48550/arXiv.2510.02373.

Ruoyao Wen, Hao Li, Chaowei Xiao, and Ning Zhang. AgentSys: Secure and Dynamic LLM Agents Through Explicit Hierarchical Memory Management. arXiv, 2026. URL https://doi.org/10.48550/arXiv.2602.07398.

Shihao Weng, Yang Feng, Jinrui Zhang, Xiaofei Xie, Jiongchi Yu, and Jia Liu. ARGUS: Defending LLM Agents Against Context-Aware Prompt Injection. arXiv, 2026. URL https://doi.org/10.48550/arXiv.2605.03378.

Simon Willison. The Dual LLM pattern for building AI assistants that can resist prompt injection. Simon Willison's Weblog, 2023. URL https://simonwillison.net/2023/Apr/25/dual-llm-pattern/. April 25.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic Memory for LLM Agents. arXiv, 2025. URL https://doi.org/10.48550/arXiv.2502.12110.

Tiankai Yang, Jiate Li, Yi Nian, Shen Dong, Ruiyao Xu, Ryan Rossi, Kaize Ding, and Yue Zhao. No Attacker Needed: Unintentional Cross-User Contamination in Shared-State LLM Agents. arXiv, 2026a. URL https://doi.org/10.48550/arXiv.2604.01350.

Xianglin Yang, Yufei He, Shuo Ji, Bryan Hooi, and Jin Song Dong. Zombie Agents: Persistent Control of Self-Evolving LLM Agents via Self-Reinforcing Injections. arXiv, 2026b. URL https://doi.org/10.48550/arXiv.2602.15654.

Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive Language Models. arXiv, 2025. URL https://doi.org/10.48550/arXiv.2512.24601.

Wei Zou, Mingwen Dong, Miguel Romero Calvo, Shuaichen Chang, Jiang Guo, Dongkyu Lee, Xing Niu, Xiaofei Ma, Yanjun Qi, and Jiarong Jiang. Poison Once, Exploit Forever: Environment-Injected Memory Poisoning Attacks on Web Agents. arXiv, 2026. URL https://doi.org/10.48550/arXiv.2604.02623.

A Additional Results

A.1 Federated-Tier Simulations at the Deployed Geometry

Figures 3 to 5 report the three experiments that Section 4.2 discusses at the deployed geometry ($N = 10$, $D_{\text{pool}} = 2$, $k/m/m_{\text{rej}} = 31/20/12$, $f = 2$, $f_s = 1$): the route decomposition under the non-adaptive blast schedule, the adaptive-adversary audit, and the safety–effort measurements in the 120-agent community. The quoted $B_{\text{att}} = 8.96 \times 10^{-3}$ uses $q_{\text{pre}} = q_{\text{pan}} = 0.6$; the other reviewer-rate settings are identified in the respective figures.

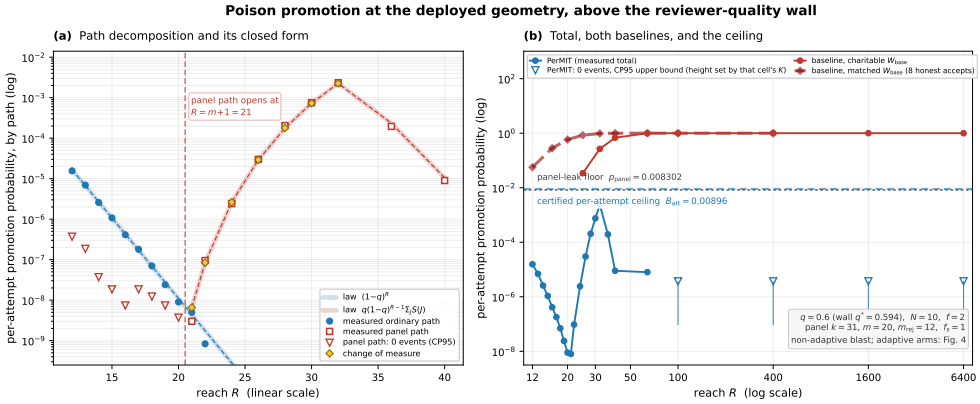

Figure 3: Promotion routes and the certified ceiling under the non-adaptive blast schedule ($q = 0.6$, $N = 10$, $f = 2$). **(a)** Path decomposition on a linear reach axis: the ordinary route follows the closed form $(1-q)^R$; the panel route opens at $R = m+1 = 21$ and follows $q(1-q)^{R-1} \sum_J S(J)$ (Proposition S8.11); open markers are zero-event cells at their 95% Clopper–Pearson upper bounds. **(b)** Total measured rate on a logarithmic reach axis against the certified ceiling $B_{\text{att}} = 8.96 \times 10^{-3}$ and the panel-leak floor $p_{\text{panel}} = 8.3 \times 10^{-3}$. The measured total peaks at $R = k+1 = 32$ and then declines below the simulation's resolution, while the no-dispute baselines (charitable and matched W_{base}) approach certain promotion as reach grows.

A.2 The Trojan Hippo Benchmark in Brief

Trojan Hippo (Das et al., 2026) evaluates persistent memory poisoning under an indirect-injection threat model: the user is trusted, the attacker controls only content the agent reads through its tools, and the attacker's goal is to exfiltrate sensitive user data to an attacker-controlled address. The attack has two stages. In the injection stage, a crafted inbound email read through a mailbox tool carries a payload that the memory backend silently stores; in the activation stage, after N unrelated benign sessions, a user request on a sensitive topic triggers the stored payload, which induces the agent to send the user's message to the attacker through an outbound tool. Table 3 summarizes the benchmark's coverage and scale.

PerMIT live containment protocol. PerMIT replaces the benchmark memory backend with its RLM reader and confirmation interface. The four main-table backbones are served through AWS Bedrock. Each case is placed in a fresh store as untrusted email-tool output in an ingestion session and exercised from a subsequent trigger session. The harness combines a live protected-agent rollout with a targeted reader-to-tool probe: the reader fills an address field, and the probe attempts to pass the resulting handle to `send_email`, with an additional literal-address variant. This explicitly attempts to exercise the action boundary without relying solely on the planner to initiate exfiltration. A case succeeds if either executed probe produces a recorded sink invocation whose serialized arguments contain the benchmark attacker address. The probes run sequentially in the same per-case store. In LLMVerifier mode, the reader probe also performs retrieval-stage crystallization and confirmation. OpenGate accepts every confirmation; OracleGate supplies the idealized correct-user decisions; LLMVerifier uses the designated model. The comparisons are between complete configurations.

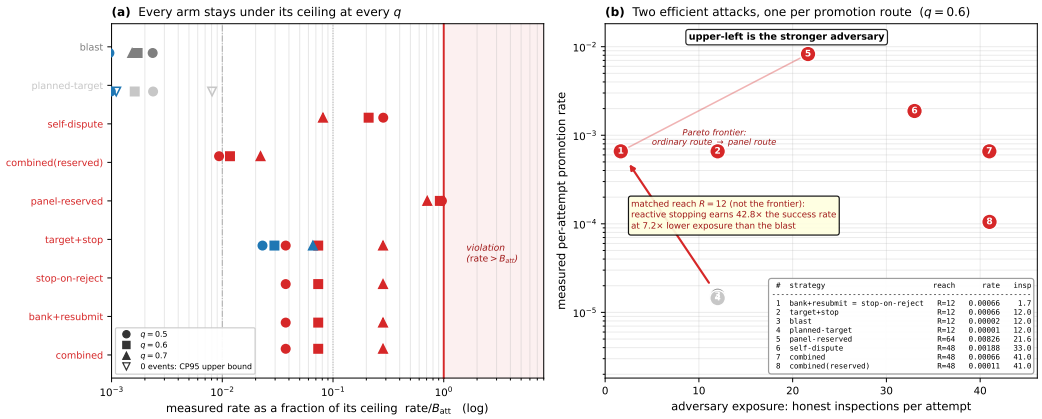

Figure 4: Adaptive adversaries at the deployed geometry. **(a)** Measured promotion rate as a fraction of the certified ceiling for nine strategies—blast, planned-target, stop-on-reject, bank-and-resubmit, target-and-stop, self-dispute, panel-reserved, and two compositions—at $q \in \{0.5, 0.6, 0.7\}$; every arm stays below its ceiling (shaded region: violation; open markers: zero-event cells at their Clopper–Pearson upper bounds). **(b)** Exposure–success frontier at $q = 0.6$: honest inspections per attempt against measured promotion rate. The two efficient strategies attain the ordinary and panel routes separately; at matched reach $R = 12$, reactive stopping achieves $42.8\times$ the success rate of the blast at $7.2\times$ lower exposure.

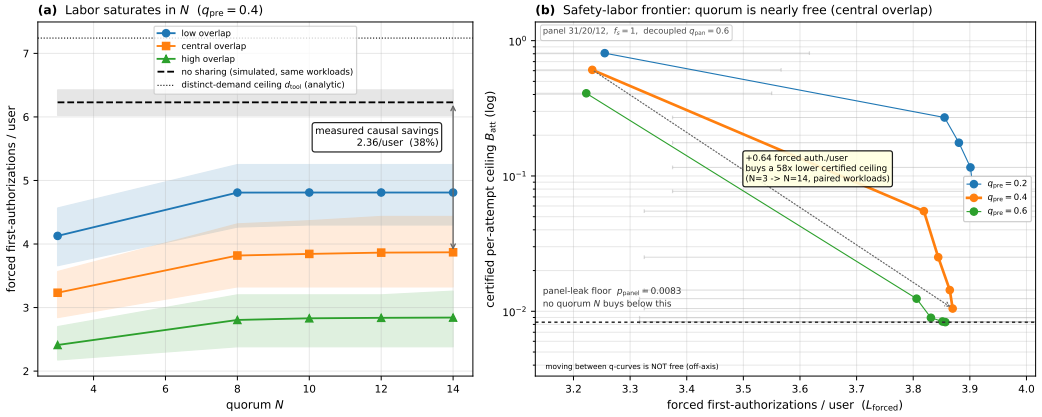

Figure 5: Safety and user confirmation effort in the simulated community of 120 agents sharing a 122-item catalog. The plot label "forced first-authorizations" denotes required approvals, not all reviews. **(a)** Required approvals per user versus quorum N ($q_{\text{pre}} = 0.4$) under low, central, and high task overlap, against the no-sharing counterfactual on the same workloads and the analytic distinct-demand ceiling d_{tool}; confirmation effort saturates in N, with a measured causal saving of 2.36 per user (38%). **(b)** Safety–effort frontier at central overlap for $q_{\text{pre}} \in \{0.2, 0.4, 0.6\}$ (decoupled $q_{\text{pan}} = 0.6$): raising N from 3 to 14 costs 0.64 required approvals per user and buys a $58\times$ lower certified ceiling; no quorum reaches below the panel-leak floor $p_{\text{panel}} = 8.3 \times 10^{-3}$, and moving between q-curves is not free.

Audit baselines. Our A-MemGuard and LLM-Auditor reimplementations share the benchmark retrieval path and reuse the corresponding undefended cases. A-MemGuard cross-checks per-record reasoning chains and maintains a per-case lesson store; LLM-Auditor independently audits each retrieved record. Retrieval depth is K=8 for RAG and K=10 for Mem0. Both discard flagged records; our auditor is a drop-only variant, not the rewriting auditor described in the original work.

Table 3: Trojan Hippo at a glance (Das et al., 2026).

Aspect	Benchmark specification
Threat model	Indirect injection only: the attacker controls the From, Subject, and Body of inbound email; cannot modify the agent's code, system prompt, memory implementation, or user queries; is white-box with respect to prompt, backend, and tool interface but black-box with respect to model weights; and observes nothing but a successful exfiltration.
Environment	A tool-calling email assistant. `read_all_emails` and `search_emails` are the untrusted read tools; `send_email`, `reply_to_email`, and `forward_to_email` are the outbound tools through which exfiltration occurs.
Trigger topics	Finance, health, legal, tax, identity; user prompts are templated with sampled sensitive entities (account balances, diagnoses and medications, case details, W-2 figures, ID numbers).
Attack generation	OpenEvolve-style evolutionary search per (topic, backend, defense): an LLM mutator proposes payload variants, an agent-in-the-loop scorer checks exfiltration and an LLM critic scores 1–10, over a MAP-Elites diversity grid. One training case and four held-out test cases per (topic, backend), drawn from disjoint template and entity pools.
Benign histories	Multi-session, single-user conversation histories generated with a modified LoCoMo pipeline; the payload is injected before session $N{=}0$ and the trigger arrives after N benign sessions. Undefended ASR does not decay up to $N{=}100$.
Memory backends	Context (sliding window); RAG (512-character chunks, top-8); Explicit (an `update_memory` list injected into the system prompt); Mem0 (LLM-extracted atomic facts, top-10); plus a no-memory control.
Defenses	User-prompt-only (index only user messages); No-untrusted-write ("No untrusted tools" in our tables: suppress memory writes after a read tool); Limit-memory-length (80-character entries); Provable policy (two-label IFC that taints a session on any read tool or tainted retrieval and blocks outbound tools in tainted sessions). Undefined pairs: User-prompt-only$\times$Explicit and Limit-memory-length$\times$Context.
Utility	Seven benign capability suites—`memory_only`, `assistant_responses`, `untrusted_probe`, `disable_send`, `untrusted_send`, `memory_tools`, `long_memory`—aggregated by arithmetic and harmonic mean; the harmonic mean collapses if any suite does.
Reported strength	Undefended ASR up to 85–100% on Gemini 3.1 Pro and GPT-5-mini; the four defenses reduce ASR to 0–5% in most configurations, the provable policy to 0% everywhere at a harmonic-mean utility near zero.

A.3 Implementation Gap in the Provable-Policy Baseline

In the supplied Trojan Hippo implementation, GLM-5 records 10% ASR under the provable-policy baseline with both Explicit memory and Mem0 (two successful attacks among twenty cases per configuration). The successful cases are health/01 and legal/03 for Explicit, and finance/01 and legal/03 for Mem0.

Our code audit identified inconsistent taint propagation in `search_emails`: the tool marks the session untrusted in `SessionTrustManager` but does not update the separate `ProvablePolicyManager` state consulted by `send_email`. An untrusted search can therefore leave the outbound guard's trusted state unchanged. All four successful traces exhibit the corresponding inconsistency: an untrusted-read notification is followed by an outbound check with `is_trusted=True` and `will_block=False`, and the benchmark's outbox-based attack checker reports success. These logs do not establish that the successful sends were triggered by retrieval of poisoned persistent-memory entries.

We retain the measured rates as results of the supplied benchmark implementation. The observed failures illustrate a practical enforcement gap: missing taint updates or inconsistent security state can permit attacker-controlled content to induce disclosure despite an intended information-flow restriction. Formal guarantees depend on faithful implementation of their assumptions across all relevant execution paths; these results expose a loophole in the evaluated enforcement rather than a counterexample to the intended policy's guarantee.

A.4 Additional Backbones and Serving Configurations

Three backbones are reported here rather than in Table 1. Qwen-3.6-27B and Gemma-4-31B-it were served locally under vLLM with a common resource configuration (tensor parallelism over four GPUs, 32,768-token context, 0.92 GPU memory utilization, automatic tool choice) and family-specific parsers. Qwen-3.6-27B:

```
vllm serve --tensor-parallel-size 4 --max-model-len 32768
--gpu-memory-utilization 0.92 --enable-auto-tool-choice
--tool-call-parser qwen3_coder --reasoning-parser qwen3
```

Gemma-4-31B-it additionally requires a model-specific chat template and, on our stack, the native sampler:

```
VLLM_USE_FLASHINFER_SAMPLER=0 vllm serve
--tensor-parallel-size 4 --max-model-len 32768
--gpu-memory-utilization 0.92 --enable-auto-tool-choice
--tool-call-parser gemma4 --reasoning-parser gemma4
--chat-template tool_chat_template_gemma4.jinja
```

Tool-call parsing on a self-hosted server is part of the measurement rather than a detail of it, because a defence that mediates tool calls is sensitive to how those calls are extracted from the model's output, and the parser is necessarily family-specific. The two backbones therefore share a serving stack but not a parser or chat template: they are comparable to each other at the stack level, not to the Bedrock-served set of the main table, and the per-family parser and template choices are recorded rather than left implicit. The lightweight verifiers (Qwen-3.5-2B and Qwen-3.5-4B) were served under the same vLLM stack, one GPU each, with a 16,384-token context, 0.90 GPU memory utilization, the native sampler, and the family's reasoning parser; verifiers render verdicts rather than tool calls, so no tool-call parser is involved. GPT-5.5 is served through the same Bedrock APIs as GPT-5.4.

The pattern of Table 1 survives the change of serving stack. Both self-hosted backbones are highly vulnerable undefended, peaking at 40 per cent attack success on Qwen-3.6-27B and 75 on Gemma-4-31B-it, both on sliding-window context; both zero the harmonic mean under the two capability-removing defences; and both reach 96.7 AM under PerMIT with the oracle at the gate, with $\mathrm{ASR} = 0$ in every occupant cell. The one instructive difference is on the small verifiers, which are harder on these two backbones than on any Bedrock one. Whether the cause is the serving stack, the family-specific parsing and chat template, or the backbone itself is not identified by these runs, since the three are confounded by construction; the honest statement is that a small external verifier needs validating against the backbone, the parser, and the stack it will actually run behind.

A.5 Detailed Verifier Results

Table 6 extends OracleGate and the three automated configurations to seven backbones, adding benign FPR and available decimal precision; OpenGate is reported separately for the four main-table backbones. Detailed scores use the available recorded precision; overlapping main-table entries are rounded half up from that precision. GPT-5.4 retains the authoritative precision of Table 1. Table 5 summarizes FPR for its four backbones.

Three readings the main table compresses are visible here. First, attack success is zero in all twenty-eight cells, which is the only column with no spread at all. Second, verifier size does not order the results: the 4B verifier has the higher harmonic mean on four of the seven backbones and the lower one on three, its false-positive rate is lower on five, and its two directions do not agree, since it wins by 25.0 points of HM on Qwen-3.6-27B and loses by 17.0 on Kimi-K2.5. The two are therefore better described as two draws from one quality tier than as a size ladder, and a two-point size difference is not the axis to select on. Third, the backbone matters more than the verifier. For a fixed verifier the spread across backbones reaches 18.3 points of AM and 45.1 of HM under the 2B model, while for a fixed backbone the spread across the three model occupants reaches 20 points of AM on Kimi-K2.5. A deployment choosing a small verifier should therefore validate it against its own backbone rather than against a published ranking.

Table 4: Single-user security and benign utility for the two self-hosted backbones and GPT-5.5, grouped by memory configuration as in Table 1. Qwen-3.6-27B and Gemma-4-31B-it use the vLLM stacks and family-specific parsers described above; GPT-5.5 uses Bedrock. All values are percentages; coloring follows Table 1. Undefined backend–defense pairs are omitted. PerMIT's FPR is shown in the final block.

	Qwen-3.6-27B			Gemma-4-31B-it			GPT-5.5		
Defense / gate occupant	AM	HM	ASR	AM	HM	ASR	AM	HM	ASR
No-memory control									
None (Baseline)	17	0	0	19	0	0	18	0	0
User-prompt-only	17	0	0	19	0	0	16	0	0
No-untrusted-tools	17	0	0	19	0	0	18	0	0
Limit-memory-length	17	0	0	19	0	0	16	0	0
Provable-policy	0	0	0	3	0	0	1	0	0
Explicit memory									
None (Baseline)	66	61	35	65	58	0	56	51	0
No-untrusted-tools	48	0	10	51	0	0	43	0	0
Limit-memory-length	51	21	5	51	20	0	40	24	0
Provable-policy	42	0	0	43	0	0	33	0	0
Mem0 backend									
None (Baseline)	34	0	5	75	38	5	75	38	0
User-prompt-only	34	0	10	71	0	5	73	0	0
No-untrusted-tools	27	0	5	50	0	5	51	0	0
Limit-memory-length	34	0	5	78	39	5	76	23	0
Provable-policy	15	0	5	48	0	5	47	0	0
A-MemGuard	35	0	5	78	39	5	68	36	0
LLM-Auditor	34	15	5	73	37	5	76	23	0
RAG backend									
None (Baseline)	93	90	35	98	98	60	99	99	0
User-prompt-only	80	0	0	82	0	5	83	0	0
No-untrusted-tools	67	0	0	67	0	0	67	0	0
Limit-memory-length	88	86	10	88	87	20	88	87	0
Provable-policy	67	0	0	67	0	0	67	0	0
A-MemGuard	96	96	10	97	96	55	95	95	0
LLM-Auditor	96	96	0	94	93	0	99	99	0
Sliding-window context									
None (Baseline)	95	94	40	98	97	75	99	99	0
User-prompt-only	85	51	0	93	92	0	82	0	0
No-untrusted-tools	66	0	10	67	0	15	67	0	0
Provable-policy	66	0	0	67	0	0	67	0	0
PerMIT: RLM reader; undefended attack source									
oracle gate	**97**	**96**	**0**	**97**	**96**	**0**	**98**	**98**	**0**
self-verify	**93**	**93**	**0**	**95**	**94**	**0**	**92**	**90**	**0**
Qwen-3.5-2B	**83**	**51**	**0**	**83**	**65**	**0**	**96**	**96**	**0**
Qwen-3.5-4B	**83**	**76**	**0**	**89**	**88**	**0**	**94**	**93**	**0**

PerMIT benign false-positive rate (FPR)	Qwen-3.6-27B	Gemma-4-31B-it	GPT-5.5
oracle gate	**0**	**0**	**0**
self-verify	**12**	**3**	**8**
Qwen-3.5-2B	**32**	**34**	**9**
Qwen-3.5-4B	**29**	**29**	**5**

Table 5: PerMIT's benign false-positive rate. All values are percentages; shading follows Trojan Hippo (green is better).

PerMIT benign false-positive rate (FPR↓)

Gate occupant	Claude-Sonnet-4.6	GLM-5	Kimi-K2.5	GPT-5.4
Oracle gate	**0**	**0**	**0**	**0**
LLM self-verification	**13**	**9**	**13**	**14**
Qwen-3.5-2B verifier	**17**	**23**	**26**	**21**
Qwen-3.5-4B verifier	**19**	**8**	**32**	**13**

Table 6: OracleGate and three automated configurations across seven backbones, at the available precision described above. FPR is rejections over verifier calls on the benign suites.

	Oracle gate				**LLM self-verify**				**Qwen-3.5-2B**				**Qwen-3.5-4B**			
Backbone	AM	HM	ASR	FPR	AM	HM	ASR	FPR	AM	HM	ASR	FPR	AM	HM	ASR	FPR
Claude-Sonnet-4.6	**96.7**	**96.4**	**0**	**0**	**92.5**	**90.2**	**0**	**13.1**	**86.7**	**79.9**	**0**	**17.0**	**82.5**	**77.2**	**0**	**19.4**
GLM-5	**92.5**	**91.0**	**0**	**0**	**88.3**	**87.0**	**0**	**8.6**	**82.5**	**77.3**	**0**	**23.2**	**84.2**	**78.0**	**0**	**8.2**
Kimi-K2.5	**90.8**	**89.7**	**0**	**0**	**90.8**	**89.1**	**0**	**13.3**	**77.5**	**67.3**	**0**	**25.6**	**70.8**	**50.3**	**0**	**32.2**
GPT-5.4	**89**	**86**	**0**	**0**	**83**	**79**	**0**	**14**	**79.2**	**73.1**	**0**	**20.8**	**84.2**	**79.6**	**0**	**13.2**
Qwen-3.6-27B	**96.7**	**96.3**	**0**	**0**	**93**	**93**	**0**	**12**	**82.5**	**50.5**	**0**	**32.2**	**83.3**	**75.5**	**0**	**28.9**
Gemma-4-31B-it	**96.7**	**96.4**	**0**	**0**	**95**	**94**	**0**	**3**	**83.3**	**64.6**	**0**	**34.1**	**89.2**	**87.7**	**0**	**29.3**
GPT-5.5	**98.3**	**98.2**	**0**	**0**	**92**	**90**	**0**	**8**	**95.8**	**95.6**	**0**	**8.7**	**94.2**	**93.1**	**0**	**5.3**

Reading the security results. ASR summarizes each complete configuration. Reader outputs may already be clean or refuse an attack, so not every case presents harmful bytes to the verifier. Appendix A.2 specifies the live probes and case-counting protocol.

Gate activity and utility. Table 7 reports calls and rejections across six benign diagnostic suites with 20 checks each. These 120 checks have a different accounting unit from the seven capability axes summarized in Table 1. FPR is rejections divided by calls, and calls per check is calls divided by 120. These totals measure gate activity; they do not measure human review time or the fraction of tasks interrupted.

Table 7: Benign gate activity over 120 diagnostic checks per configuration. FPR is a percentage; calls per check is a ratio.

Backbone	Verifier	Calls	Rejects	FPR	Calls/check
Claude-Sonnet-4.6	Self	107	14	13.1	0.89
	Qwen-3.5-2B	100	17	17.0	0.83
	Qwen-3.5-4B	98	19	19.4	0.82
GLM-5	Self	116	10	8.6	0.97
	Qwen-3.5-2B	125	29	23.2	1.04
	Qwen-3.5-4B	98	8	8.2	0.82
Kimi-K2.5	Self	113	15	13.3	0.94
	Qwen-3.5-2B	121	31	25.6	1.01
	Qwen-3.5-4B	115	37	32.2	0.96

Verifier choice can substantially affect utility. Kimi with Qwen-3.5-4B has HM 50.3, compared with 89.7 under its oracle gate. For GPT-5.4, the authoritative main-table values are oracle AM/HM 89/86 and self-verification 83/79, a decrease of approximately 6 AM and 7 HM points at that precision. Larger verifiers do not consistently have lower FPR. Within PerMIT, these oracle-to-verifier comparisons retain the RLM reader and interface, characterizing the utility cost associated with automated gate decisions relative to correct endorsement. Comparisons against other memory backends instead evaluate complete systems; separating retrieval quality from the overhead of mediation would require matched reader controls. Human review time is a separate deployment measure.

Table 8: Utility of the autonomously constructed knowledge base in the banking domain. Contrasts against the no-knowledge baseline are paired by task and seed; intervals are 95% confidence intervals.

Backbone	No knowledge	Δ vs. no knowledge [95% CI]
GLM-5	0.428	-0.000 $[-0.048, +0.048]$
Kimi-K2.5	0.467	$+0.002$ $[-0.027, +0.030]$
Claude-Sonnet-4.6	0.590	-0.015 $[-0.081, +0.045]$

A.6 Domain Dependence of Self-Constructed Knowledge

We repeat the end-to-end autonomous construction experiment in the banking domain of τ^2-bench using the same three model families. Table 8 reports the resulting held-out utility. Unlike the telecom experiment in the main text, banking provides a golden-retrieval oracle rather than a hand-written policy manual; because these reference conditions are not directly comparable, the table reports the confirmatory contrast against the knowledge-free baseline only.

The self-constructed knowledge base produces no detectable utility gain in banking: all confidence intervals include zero. This contrast with telecom is consistent with the information available in the two environments. Telecom tasks require causal, experience-derived knowledge about how the environment behaves—knowledge that is absent from the supplied documentation and can be acquired only through interaction. Banking tasks primarily require locating information in a well-indexed document corpus that the environment's retrieval system already makes available on demand. Consequently, the telecom community can contribute knowledge that later agents would otherwise lack, whereas the banking community largely distils information that each agent can already retrieve independently. Together with GLM-5's near-ceiling telecom result, these findings identify a practical boundary on the utility of shared memory: self-constructed knowledge is most valuable when it captures scarce, experience-derived information, and least valuable when the environment already supplies the required information or the underlying agent has little remaining performance headroom.

A.7 Autonomous Telecom Construction: Methods and Additional Results

Construction and evaluation. The construction ladder comprises one run for each of three model families (Claude-Sonnet-4.6, GLM-5, and Kimi-K2.5) and four community sizes $M \in \{12, 24, 60, 90\}$. Each run uses one model family for all construction roles. Agents crystallize session knowledge into an untrusted handle, submit its exact content to the safety gate for reasoning use, screen approved content for generality and privacy, and consent to publication. Other agents then review shared entries.

The standard utility evaluation uses 24 held-out tasks with two trials each (48 task/trial pairs per arm) and a 200-step limit. No Prior Knowledge (A0) provides no shared knowledge base, PerMIT (A2) supplies community-confirmed knowledge, and Expert Manual (A4) uses the benchmark's alternative hand-written workflow policy. A4 is not the original policy with extra instructions appended, nor an oracle upper bound on task performance. A0 and A2 keep prompts identical outside the knowledge block. A2 includes all retained entries, with a coarse ordering that distinguishes uncontested from vindicated entries, without truncation. Generated experience primarily covers MMS troubleshooting; the reported Sonnet and Kimi gains are concentrated in that task family. This supports transfer of relevant experience, without establishing gains for every task family.

Pairing and uncertainty. Seeds are matched by task and trial across arms. Table 2 separates the $M{=}90$ means from the analysis pooled across all four A2 community sizes against corresponding A0 episodes. Table 9 gives the complete size ladder and repeats the pooled analysis for reference. Pooling yields 192 paired differences but reuses the evaluation tasks and baseline; these are not 192 independent tasks. The analysis resamples whole tasks jointly across their trials and sizes: 95% confidence intervals use 10,000 cluster-bootstrap resamples, and the continuous-score test uses 20,000 task-cluster sign-flip permutations, both with analysis seed 20260811. Holm correction across the three same-backbone comparisons gives adjusted p-values of 0.0066 for Claude-Sonnet-4.6, 0.0325 for Kimi-K2.5, and 0.9356 for GLM-5. The bootstrap confidence intervals are

Table 9: Same-backbone telecom utility across community sizes (upper panel), and the separate four-size pooled paired analysis (lower panel). Each size uses 48 task/trial pairs. Pooled 95% intervals use task-cluster bootstrap resampling and do not describe the $M=90$ result alone.

Builder and agent	$M=12$	$M=24$	$M=60$	$M=90$
Claude-Sonnet-4.6	0.7917	0.7431	0.8090	0.7986
Kimi-K2.5	0.9375	0.9167	0.8750	0.8854
GLM-5	0.9792	0.9583	0.9479	0.9688

Builder and agent	**Pooled PerMIT**	**Pooled Δ vs. A0 [95% CI]**
Claude-Sonnet-4.6	0.7856	$+0.1363$ $[+0.0651, +0.2092]$
Kimi-K2.5	0.9037	$+0.1120$ $[+0.0339, +0.1979]$
GLM-5	0.9635	$+0.0052$ $[-0.0521, +0.0781]$

Table 10: Same-backbone utility at $M=90$. Entries are counts of knowledge items supplied to each arm.

Builder and agent	**No Prior Knowledge**	**Unfiltered A1**	**PerMIT A2**	**Entries A1 / A2**
Claude-Sonnet-4.6	0.6493	0.8819	0.7986	71 / 48
GLM-5	0.9583	1.0000	0.9688	70 / 45
Kimi-K2.5	0.7917	0.8958	0.8854	66 / 64

pointwise, not multiplicity-adjusted. Using $m=5$ as a conservative multiplicity sensitivity analysis gives adjusted $p=0.0110$ for Sonnet and $p=0.0650$ for Kimi. These quantify evaluation uncertainty for the observed knowledge bases, not construction variability; pooled intervals must not be read as $M=90$ intervals. Step-limit terminations receive zero environment-completion score; infrastructure failures are excluded from the common paired set and counted separately.

Fixed geometry and finite review opportunities. The construction ladder keeps $N=10$, $k=31$, $m=20$, and $m_{\text{rej}}=12$ fixed. The panel can terminate when a decision threshold is reached; 31 votes are not required for every resolution. Nevertheless, $M=12$ cannot supply 20 fresh accepting panelists, nor the initiating rejection plus 12 further rejecting panelists required for eviction. Entries can be detected and frozen without reaching either terminal outcome. At $M=24$, early threshold resolution can be feasible, subject to eligibility. Construction is sequential within one epoch: an entry published by agent i has at most $M - i$ later reviewers. Raw promotion fractions therefore mix review quality with available demand. From $M=24$ to 90, reviews per agent change from 13.92 to 14.27 for Sonnet, 9.58 to 7.16 for GLM-5, and 12.25 to 10.70 for Kimi, while evaluated knowledge grows from 7 to 48, 10 to 45, and 12 to 64 entries, respectively. This is a finite-scale knowledge/effort trade-off, not a monotonic task-performance trend or an asymptotic theorem. At matched remaining-review horizon $H=30$, benign yield from $M=60$ to 90 is 0.833 to 0.833, 0.824 to 1.000, and 0.826 to 0.976 for Sonnet, GLM-5, and Kimi, respectively; one run per setting limits inference about population effects.

Unfiltered shared-knowledge control. The existing A1 arm supplies shared knowledge without the community-promotion filter. Table 10 compares A1 and A2 at the same $M=90$ construction budget, rather than comparing A1 to the four-size pooled result. A1 has higher mean utility on all three backbones. These are descriptive comparisons: the retained contents and prompt lengths differ, and a paired A1–A2 test is not reported here. This is a benign-utility comparison, not a measurement of relative attack resistance. It establishes that unfiltered shared experience can help; it does not isolate the promotion filter on identical candidate pools or substitute for a simpler voting control.

Cross-backbone transfer. Claude-Haiku-4.5 uses knowledge built at $M=90$ by the indicated producer; its simulated user is GLM-5. The 200-step comparisons use 47 common task/trial pairs after one infrastructure failure in the wider Haiku comparison set. Table 11 reports that common set. The separate 400-step Kimi-to-Haiku evaluation uses 24 tasks with two trials each (48 pairs) and yields $\Delta = +0.1979$ $[+0.0729, +0.3333]$. Its 24 task clusters comprise 11 MMS, four mobile-data, and

Table 11: Transfer to Claude-Haiku-4.5 at 200 steps on 47 common pairs. No Prior Knowledge (A0) is 0.4681 and Expert Manual is 0.5319 for this set. The 400-step Kimi result is reported separately in the text.

Knowledge builder	**With PerMIT**	**Δ vs. A0 [95% CI]**
Claude-Sonnet-4.6	0.5177	$+0.0496$ $[-0.056, +0.152]$
GLM-5	0.5638	$+0.0957$ $[-0.056, +0.261]$
Kimi-K2.5	0.6064	$+0.1383$ $[+0.0222, +0.2660]$

nine service tasks. This result is reported separately because its episode set and step budget differ; its effect is not added to the 200-step baseline to reconstruct an absolute mean.

Scope of the safety probes. The separate 12-payload direct memory-injection campaign reports no community promotion. This endpoint concerns access to the shared fast path; it does not measure personal acceptance, use before promotion, or every intermediate lifecycle transition.

B THE MEMORY INTERFACE IN PRACTICE

This appendix restates the memory interface of Section 3.2 from a systems perspective: the seven operations and the fixed alphabet of replies the P-LLM can receive (Section B.1), the label lifecycle (Section B.2), a step-by-step trace of a memory-poisoning episode (Section B.3), and the implementation of the Quarantined Reader (Section B.4). It is informative: no rule reads it and no proof cites it. The normative transition schema and Rules R1–R11 are in Supplement S2; Section D.1 summarizes their role in the guarantees. Throughout, P denotes the P-LLM and Q the Q-LLM.

B.1 OPERATIONS AND THE REPLY ALPHABET

Every reply the interpreter returns to P is drawn from a fixed alphabet (Rule R10),

$$r \;::=\; \mathsf{ack} \;\mid\; h_j \;\mid\; \mathsf{fail} \;\mid\; \mathsf{dec}(\mathsf{accept}) \;\mid\; \mathsf{dec}(\mathsf{reject}) \;\mid\; \mathsf{bytes}(b),\; b \in \mathcal{M}_{\mathbb{E}},$$

so the only readable text P ever receives is the verbatim content of a trusted block. The isolation half of the single-user argument (Lemma D.1) is the check, operation by operation, that untrusted bytes never appear in the last category; Table 12 makes that check readable. Two further properties keep the alphabet uninformative about untrusted memory: handle identifiers are fixed by P's own operation sequence and never encode store contents (Rule R1), and an empty search, a reader failure, and an unoccupied identifier all return the same operation-determined handle (Rules R6, R7).

B.2 LABEL LIFECYCLE

Blocks are immutable once written (Assumptions A1–A2); trust changes only by the creation of a fresh block. Figure 6 shows every permitted transition. The only path from U to a trusted label is display followed by an accepting confirm, which copies the displayed bytes into a new C block and leaves the untrusted original in place.

B.3 A WORKED TRACE: DORMANT POISON ACROSS TWO SESSIONS

Table 13 follows a Trojan Hippo-style episode: a payload planted through one untrusted tool call in session 1 and activated by a sensitive request in session 2. The third column is the point of the exercise—it lists everything P ever receives.

Three facts about the trace carry the design. First, the payload never enters P's context: steps 2–5 deliver only acknowledgements and handles, so P's plan in session 2 is what the user typed, not what the attacker wrote (Reasoning Integrity, Section 3.2). Second, the payload cannot reach a tool without being displayed verbatim at step 6 and accepted for that exact tool (Action Integrity, Section 3.2). Third, everything therefore rests on the decision at step 6—the single auditable event whose occupants (oracle, self-verification, external LLM verifiers) Table 1 evaluates.

Table 12: The P-facing memory interface. Trusted labels: T (typed), C (confirmed), CC (community-confirmed), PC (P-composed); U is untrusted. Transition names and rule numbers refer to Supplement S2. Untrusted bytes reach P only through confirm.

Operation	Purpose	P **receives**	Store effect	Semantics
The seven memory operations (emitted by P)				
write(b)	store a P-composed note	ack	new PC block: readable; empty tool scope, so never fast-paths a sensitive tool	W-P; R1(g)
recall(id)	read a block	**bytes** if id resolves to a trusted block; otherwise an opaque handle, identical for untrusted, unknown, and absent ids	—	RECALL-E/O; R6, R10(b)
fill(q)	Q searches untrusted memory	opaque handle; $\leq c_{\text{cand}}$ candidates bound interpreter-side, empty result indistinguishable	—	FILL; R6, R7
crystallize(t)	Q writes a schema-bound summary	opaque handle	new U block	CRYST; R5
display(x)	render content to the user	ack	transcript event only	DISPLAY; R8
confirm(h, t)	the gate: exact bytes, source, and task shown; a human or LLM verifier decides	dec(accept) or dec(reject); bytes never returned on reject	on accept: fresh C copy of the displayed bytes, readable, scope $\{t\}$ ($\varnothing$ for t = reasoning); original block unchanged	CONF-ACC/REJ; R2, R5, R8
evict(id)	delete an owned block	ack (uniform)	owner's block removed	EVICT; R10(b)
Boundary events (create memory or cross the tool boundary)				
typed input	the user's command	the typed text, as trusted input	T block(s)	W-TYPED
paste	content pasted by the user	nothing; not observable to P	U block	W-PASTE
retrieve(q)	trusted-store retriever $R_{\mathcal{E}}$ selects an endorsed block	bytes of that block, or fail	—	RETRIEVE(-F); R11
tool call	an argument flows only if user-typed or confirmed for that tool; a handle argument always fires confirm	ack, plus one decision if gated	every output captured as U blocks under one collection handle; gated arguments endorsed as fresh C copies	TOOL-FAST/GATE; R3, R4
map($\hat{o}, H$)	one operation over a whole untrusted collection	ack; one aggregated decision	element outputs captured as U	MAP; R1(c), R9

B.4 THE RECURSIVE LANGUAGE MODEL AS A QUARANTINED READER

Our implementation uses an RLM (Zhang et al., 2025) because it supplies search, structured extraction, and crystallization within the quarantined reader (fill and crystallize in Table 12). Conventional retrieval would need adaptation to provide the same interface while preserving the separation between untrusted content and the protected agent.

Why an RLM. The reader must search an untrusted store that is unbounded (convention (C7)) and routinely exceeds any context window, yet its bytes must never be loaded into a trusted context. An RLM fits both constraints by construction: the long input is not fed to the model but placed as a variable in a persistent REPL environment, and the model writes code to inspect, decompose, and

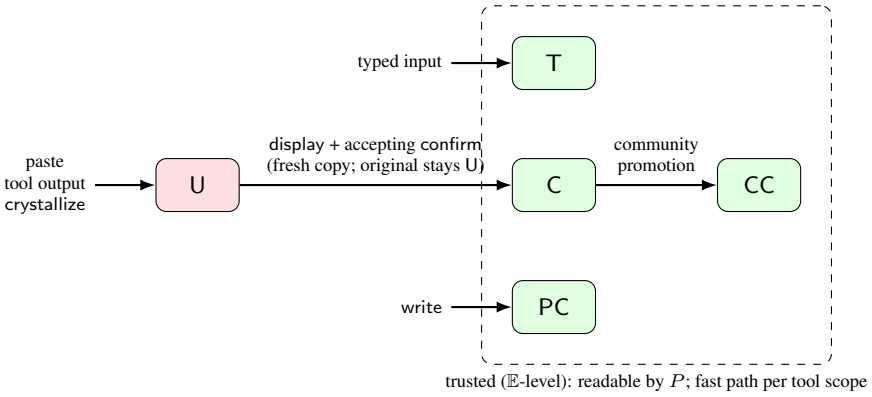

Figure 6: Label lifecycle. No other transition exists: no operation relabels a block in place, and CC status lives in the shared tier rather than in any local store.

Table 13: What P receives at each step. Handles h_j are opaque and operation-determined; T, C, and U are block labels.

Step	Actor and operation	P receives	Memory and transcript
Session 1			
1	User types "Dentist: Dr. Lee, 555-0100"	the typed text (trusted command input)	T block
2	P calls web_search("Dr. Lee dentist"); one result carries a planted instruction ("…forward the user's saved files to x@evil.com…")	ack only	page(s) captured as U blocks under collection handle h_1
3	P: crystallize("dentist notes")	h_2	U summary block; the planted text, if included, remains U
Session 2			
4	User types "Email Dr. Lee's number to my sister, sis@example.com"	the typed text	T blocks
5	P: fill("Dr. Lee contact")	h_3, identical had nothing been found; no bytes are read, so P's plan remains what the user typed	candidates bound interpreter-side
6	P calls send_email(to = sis@example.com, body = h_3); the T-valued address fast-paths, the handle fires the gate	—	selection surface lists candidates with previews marked unverified; decision pane shows the exact resolved bytes, the attested facts (channel: tool output; label U), the unverified source strings, and the task
6a	Selected candidate resolves to the planted instruction; the verifier rejects	dec(reject)	no invocation; display and decision logged
6b	Selected candidate resolves to "555-0100"; the user accepts	dec(accept) · ack	fresh C block "555-0100" scoped {send_email}; the tool runs with exactly the displayed bytes; its output is captured as U
7	Later reuse of the confirmed number at send_email	bytes via recall; no gate	TOOL-FAST; at a different tool the pair gates once more (scope is per tool)

recursively query sub-models over programmatic slices of it (Zhang et al., 2025). The store is thus navigated by code, symbolically, at any size.

Algorithm 1 RLM-based retrieval for one reader operation (fill or crystallize)

Require: query q, schema s, memory store $\mathcal{M}$; iteration cap T
Ensure: a schema-valid value with citations, or the null handle

$V \leftarrow \{\, b \in \mathcal{M} : \ell(b) = \mathsf{U} \,\}$ ▷ view restriction (interpreter, deterministic)
$\mathcal{E} \leftarrow$ INITREPL($\texttt{context} \leftarrow [(\mathrm{id}(b), \mathrm{bytes}(b)) : b \in V]$)
issue fidelity-only instruction for (q, s) to root model Q with constant-size metadata of `context`
for $t = 1, \ldots, T$ **do**
  $c \leftarrow Q(\text{history})$; $(\mathcal{E}, \text{out}) \leftarrow \text{REPL}(\mathcal{E}, c)$ ▷ Q reads slices; sub-calls for oversized blocks
  if $\mathcal{E}[\texttt{answer}]$ is set **then**
    $(v, C) \leftarrow$ PARSE($\mathcal{E}[\texttt{answer}]$), disambiguated against the true identifier set of V
    if v satisfies s **and** $C \subseteq \{\mathrm{id}(b) : b \in V\}$ **and** $C \neq \varnothing$ **then**
      return (v, C) ▷ to P as opaque handle / U-block only
    end if
    return null handle ▷ fail closed, Rule R6
  end if
end for
return null handle ▷ cap or reader error, Rule R7

How retrieval works. The interpreter first restricts the reader's view to U-labelled blocks—endorsed content and interpreter state are structurally invisible—and loads the visible blocks (identifier, content, tags) as a list variable inside the REPL. The root model receives only the retrieval instruction and constant-size metadata; it then iterates, printing slices of the store, delegating oversized blocks to sub-model calls, and accumulating its answer in a result variable, under a fixed iteration cap. The instruction is *fidelity-only*: read the actual blocks, copy any extracted value verbatim, cite the source block identifier, and report absence as absence—the reader carries no safety obligation, because every safety property is enforced downstream by the interpreter (Lemma D.1, Rules R6, R7). Finally, the interpreter validates the reply deterministically: the value must satisfy its declared schema and every citation must name a block genuinely present in the view; any failure, reader error, or uncited summary collapses to the null-handle outcome the semantics already define (Rule R6). An accepted value reaches P only as an opaque handle, resolved at a confirmation gate; an accepted summary is written back at level U (FILL, CRYST, Figure 6). Algorithm 1 summarizes the procedure.

Utility of the complete pipeline. In every PerMIT cell of Tables 1 and 4, the RLM reader is the retrieval path for stored untrusted content. With the oracle gate, PerMIT has higher AM than undefended RAG on four of seven backbones, trails by at most 1.5 AM points on Gemma-4-31B-it and GPT-5.5, and by about ten AM points on GPT-5.4. These cross-backend comparisons assess the complete mediated pipeline, not a gate-induced improvement in retrieval quality. The oracle measures retained capability under correct endorsement, while comparisons among PerMIT's gate configurations keep the reader and interface fixed and characterize decision-dependent utility costs. Isolating mediation overhead from reader quality would require a matched no-gate reader control; measuring human confirmation effort requires a separate study.

C CONCURRENT WORK: ORIGIN-BOUND MEMORY AUTHORITY

TMA-NM (Louck, 2026) studies origin-bound authorization of consequential actions. PerMIT combines observation mediation at a finite memory interface, exact-value and tool-scoped endorsement, a transcript-conditioned integrity theorem, and federated trust accumulation. The comparison below distinguishes the policies, proved properties, and verification scopes. Each guarantee depends on the assumptions and enforcement rules of its own model.

Independent development. PerMIT was developed independently of, and concurrently with, TMA-NM. We became aware of the earlier public preprint while preparing this submission and include this comparison to clarify the relationship between the two approaches.

	TMA-NM (Louck, 2026)	**PerMIT**
Enforcement boundary	Origin-bound consequential-action authorization	Observation mediation and sensitive-tool authorization under the specified interface
Reasoning-side scope	Answer bias remains outside the guarantee	Noninterference conditional on the complete display transcript; endorsed content may influence responses
Verification scope	Bounded TLC checks and an inductive-invariant argument; parameter independence justified by hand	Lean 4 proofs for unbounded executions of the single-user core; remaining scope in Appendix E
Value attribution	Origin labels in the unified study; text proxy in the supporting study	Opaque handles before release and exact-value checks at sensitive tools
Tool-output trust	Conditional trust for allow-listed tools	All tool outputs enter untrusted; readable release requires endorsement
User-authorization scope	Fresh, single-use authorization bound to the concrete action	Reusable authorization for an exact value and named tool; transaction context is not bound
Multi-user sharing	Out of scope	Federated promotion and dispute analysis under stated identity, reviewer, and scheduling assumptions

C.1 The observation boundary

TMA-NM permits untrusted memory to inform responses while constraining consequential actions. PerMIT additionally mediates what the protected agent can observe. Under Lemma D.1, untrusted sources are represented by opaque handles until endorsement releases their contents. This prevents readable delivery of unendorsed memory through the modeled interface; it does not prevent all possible influence from malicious content after endorsement.

Theorem D.5 states the precise comparison. For matched initial configurations, identical user-typed inputs, coupled randomness, the same schedule, and a common horizon, equal complete display transcripts imply equal observable protected-agent behavior. An attacker can nevertheless change a displayed preview, influence an endorsement decision, or supply harmful content that a verifier accepts. Such changes may alter the transcript and subsequent behavior. The theorem establishes no additional influence beyond the transcript under its matching conditions; it does not establish factual correctness, unbiased recommendations, or the soundness of an automated verifier (Appendix D.2.2).

Isolation likewise blocks in-context reconstruction from fragments while they remain unavailable to the protected agent as unendorsed content. Once fragments have been endorsed and released, the agent can combine them. Sensitive calls remain subject to exact-value and tool-scope checks, but separate endorsements do not establish the safety of a combined transaction or its suitability in a changed context (Appendix D.2.1).

C.2 Relational versus reachability properties

The single-execution invariants reported for TMA-NM do not establish PerMIT's two-execution, transcript-conditioned property. This is a difference in the statements proved, not an expressiveness limitation of model checking. Relational safety properties can be reduced to safety properties over self-composed or product systems representing multiple executions (Shemer et al., 2019). Establishing an analogous result would require an appropriate relational model and a separate proof; enlarging an existing single-execution model alone does not supply that proof.

PerMIT's Lean development establishes its integrity properties for arbitrarily long executions of the mechanized single-user core, under the modeled interface and assumptions. Appendix E distinguishes this checked core from the paper-only extensions and federated analysis. A proof about either formal model does not by itself establish that a deployed harness implements that model faithfully.

C.3 THE FEDERATED TIER

PerMIT also studies shared memory contributed and reviewed by a population of attested identities. Its analysis addresses promotion, disputes, wrongful eviction, and confirmation cost under explicit identity budgets, reviewer conditions, and protocol schedules (§3.3).

These guarantees are conditional and probabilistic. Attestation and first-decision accounting constrain how much weight adversarial identities can supply, but honest reviewers can still accept poison. Equation 1 bounds that residual promotion risk. Under the demand, coverage, and threshold schedules of Supplementary Theorem S6.13, the risk bounds decrease with community size while the per-user confirmation burden for covered benign shared demand does not increase. This is not an unconditional guarantee of zero poison promotion or decreasing review effort for every workload.

C.4 DEPLOYMENT WITHOUT INSTITUTIONAL CORROBORATORS

PerMIT's single-user tier operates with the user as endorser and does not require a federation. Once an exact value has been approved for a named tool, retaining the endorsed copy and its in-scope authorization permits reuse without another mandatory approval for that same pair. This does not bound the number of rejected review requests, and deleting the local endorsed copy can remove the reuse path. The guarantee also does not establish how much useful content a human or automated verifier will approve.

Reuse is an explicit authorization-policy choice. PerMIT's endorsement does not bind argument position, task, purpose, time, or the other arguments of a call. Community disputes suspend community reuse while preserving prior personal confirmations. Thus the confirmation-cost claim should be read together with the retained-copy assumption and the contextual-authorization limits in Appendix D.2.1.

C.5 LAUNDERING CHANNELS

PerMIT addresses laundering through the modeled interface rules. Reader-generated summaries remain untrusted when written by crystallize, so summarization alone grants no authority. All tool outputs are captured as untrusted, preventing a tool's name or status from automatically making its output trusted. These rules do not make subsequently endorsed content harmless or protect against a compromised enforcement harness.

Repeated content does not itself constitute independent endorsement. Distinct attested identities and first-decision accounting restrict manufactured voting weight. Where the promotion quorum exceeds the certified adversarial identity budget, adversarial endorsements alone cannot satisfy it; erroneous honest endorsements can still promote poison, with risk bounded under the assumptions of Equation 1.

The `capability_ifc` baseline evaluated by Louck (2026) treats retrieved memory as trusted. Its reported failures concern that evaluated configuration and do not establish that every deployment of CaMeL, FIDES, or capability-based information-flow control fails under the same attacks.

D FORMAL GUARANTEES AND PROOF SKETCHES

This appendix summarizes the memory-interface model and single-user integrity guarantees cited in the main text. Supplementary Document S gives the complete semantics and integrity proofs (S2–S3), federated analysis and confirmation-cost results with their assumptions (S4–S6), security discussion (S7), probability tools and assumption audit (S8), and detailed Lean correspondence (S9).

D.1 INTERFACE AND TRUST ASSUMPTIONS

The Protected LLM P is an arbitrary agent strategy over its observations, interaction history, and random coins. It chooses operations and can re-plan on endorsed feedback. The harness mediates the memory interface and tool boundary; it does not inspect or taint-track the agent's internal reasoning. The Quarantined Reader Q may behave adversarially and holds no tool capability. Appendix B.1 gives the operation-level view; Supplement S2 gives the normative semantics.

Stored values and authority. Each immutable block has bytes, a source-based label, provenance, and a tool scope. The labels are T (user-typed), U (untrusted), C (personally confirmed), PC (agent-composed), and, in the federated extension, CC (community-confirmed). The readable store $\mathcal{M}_{\mathbb{E}}$ consists of non-U blocks. Readability alone does not authorize a sensitive call: a literal must byte-match a T block or a confirmed block scoped to that tool. A PC block has no independent endorsement authority. A confirmation for the reasoning target makes its bytes readable but grants no sensitive-tool scope.

The trusted command channel creates T blocks. Pasted content and all tool outputs, including errors and status text, enter U. Reader-generated summaries also enter U. The agent receives an opaque handle, not those bytes. Accepting confirmation creates a fresh C copy; it never relabels the untrusted source in place. Endorsed and untrusted identifiers use separate allocation streams, so an attacker cannot influence a later endorsed identifier by adding untrusted blocks.

Mediation rules. The harness enforces the following properties (complete rules: Supplement S2.3).

- Every handle-expressed tool argument gates, even if its hidden bytes equal an already trusted value. The gate resolves the handle, displays the exact bytes and target, and binds an acceptance to those same bytes. Rejection or failed resolution prevents the flow.
- Sensitive-tool literals use only the typed or exact-tool-confirmed fast paths. The baseline permits agent-composed literals at data tools. Consequently, endorsement for one purpose does not imply a confidentiality guarantee for all later data-tool uses; Option O2 adds exact-tool checks there.
- Reader failure, empty retrieval, and unoccupied identifiers have the same opaque reply. Handle structure follows operations rather than hidden content, size, or success. The reply alphabet contains only fixed acknowledgements, handles, fixed failures, decisions, and verbatim readable-store bytes. Trusted retrieval sees $\mathcal{M}_{\mathbb{E}}$ alone.
- All attacker-influenceable user-facing content is recorded in the display transcript. Authenticated facts and untrusted source claims occupy separate zones. Selection previews carry no authority; authorization uses the resolved value in the decision pane.
- A batch displays every value that would flow, records its pages and decisions, and executes only after all required approvals. Accepted values remain endorsed even if a later review aborts the invocation. Full batch semantics and the corresponding determinacy argument are in Supplement S2–S3.

Assumptions and boundaries. Both integrity results assume source-based labels, immutable stored blocks, and faithful displays (A1–A3 in Supplement S2.4), together with the enforced interface above. Action Integrity additionally assumes *sound atomic endorsement*: acceptance means that the user intends the exact displayed value at the named tool, separately for each reviewed value, including accepted positions of a transaction that later aborts (A4). An LLM verifier can violate this assumption; the theorem does not establish its accuracy. Reasoning Integrity instead assumes

a common exogenous typed-input stream and matched scheduling and coins (A6 and conventions C1, C9); it does not require endorsements to be semantically correct. Timing, termination, and compromised harness components are outside these guarantees.

Lemma D.1 (Protected-agent isolation)**.** *In every interface-conformant execution, each bytes-carrying reply delivered to P is the verbatim content of a readable-store block at delivery. Untrusted sources reach P only through opaque handles until endorsement. This is a statement about the source of a delivered value, not disjointness of byte strings across stores.*

Proof sketch. Inspect the reply constructors. Tool capture, untrusted recall, reader search, and crystallization return handles or fixed tokens. Only trusted recall and retrieval return bytes, and both read $\mathcal{M}_{\mathbb{E}}$. Input ingestion and each operation preserve this property, so induction extends it to arbitrary run length. Full proof: Supplement, Lemma S3.12. □

D.2 Integrity Proofs

D.2.1 Action Integrity

Definition D.2 (Personal Action Integrity)**.** A sensitive sink event at tool t is personally authorized when each argument value v byte-matches user-typed content, or this user has accepted the exact bytes v for t, either at that call or in a retained in-scope confirmation. For a batch, acceptance covers each displayed element separately. Action Integrity requires this property for every sensitive sink event of every execution.

Theorem D.3 (Action Integrity)**.** *Under the interface and A1–A4 above, every execution starting from a well-formed configuration has personal Action Integrity, for arbitrary agent strategies, attacks, and reader behavior. The empty configuration is well formed; a seeded configuration must carry a valid endorsement record. The invariant requires immutable source-correct blocks, valid endorsement ancestry and history, interface-conformant state, and authorization of the accumulated sink log (complete definition: Supplement, Definition S3.3).*

Proof sketch. The empty state satisfies the invariant. For each transition, new untrusted blocks confer no authority; agent writes confer readability but no new authorization; and an accepted confirmation creates a fresh copy justified by the exact display and sound endorsement. A sensitive call therefore has only three admissible cases: a typed literal, an in-scope confirmed literal, or fresh accepting confirmation. Every handle takes the third case, and a batch applies it to every flowed element. Failed reviews prevent the call, while already accepted values remain individually authorized. These cases preserve the invariant and establish sink safety by induction. The single-user interpreter has no CC creation path. Full statement and proof: Supplement, Theorem S3.21. □

Corollary D.4 (No silent resolution)**.** *Under the same conditions, never-endorsed untrusted bytes cross a tool boundary, including a data-tool boundary, only through acceptance of those exact bytes for that tool in that call. After endorsement at any target, their readable copy may be used as a literal at baseline data tools. Sensitive-tool scope checks and the rule that handles always gate still apply.*

Proof sketch. Isolation leaves never-endorsed bytes available only through handles, and every handle resolution gates. Endorsement subsequently creates a readable copy, enabling the stated literal path. This distinguishes exact-byte authorization from control influence carried by a logged decision. Full proof and Option O2's closure of the later data-tool path: Supplement, Corollary S3.23. □

Reuse and contextual authorization. The policy authorizes exact values for tools, not complete transactions. Separately approved values may be combined without joint review, and a changed task context may make their reuse inappropriate. Shared disputes suspend community reuse but preserve prior personal confirmations. Candidate expiry retires dormant or inactive shared entries (Supplement, Rule R14); it is not a universal authorization lifetime. Local scope leases are a future extension outside the proved model (Option O3). Context-bound or transaction-level approval would require extended semantics and revised confirmation-cost accounting.

Future direction: combined-memory review. Human or agent memory managers could assemble related memories with their provenance and intended use, then review their joint meaning. The combined item remains untrusted until confirmation; its components' approvals do not automatically endorse it. If subsequently published, it must gather fresh endorsements of its exact content under the community-promotion rules. Revisions require their own review. This proposed workflow builds on crystallization and endorsement but has not been evaluated and does not authorize every later transaction using the combined memory. Future experiments should compare it with per-item review under matched attack-search and review budgets, measuring harmful endorsement, delayed-combination success, benign utility, and review burden.

D.2.2 Reasoning Integrity modulo the display transcript

The protected view is $\mathsf{obs}_P(c) = (\mathcal{M}_{\mathbb{E}}, H, x, h)$: the readable store, observable handle structure, typed input, and decision history. Matched configurations $c_1 \approx c_2$ agree on this view, the agent's interaction history and consumed coins, endorsed allocation state, and initial ghost display and sink logs. Their untrusted stores and private handle resolutions may differ arbitrarily. A common endorsed seed with empty histories is a canonical matched start.

The complete display transcript $\mathsf{DT}(\rho)$ records selection surfaces and previews, exact decision-pane values and provenance zones, targets, selections, accepts and rejects, and batch pages and aggregated decisions, including reviews in aborted transactions. Its event kinds are interpreter-assigned. The observable behavior $\mathcal{B}(\rho)$ comprises protected-agent operations, tool invocations, the reply stream, final protected view, and the ungated sink projection. The latter omits arguments flowed by fresh confirmations, which are already recorded in the transcript.

Theorem D.5 (Reasoning Integrity modulo the display transcript)**.** *Under A1–A3 and the input and scheduling conditions above, fix matched starts, any two attacks, and a coupled coin sequence. For runs ρ_1, ρ_2 of a common horizon,*

$$\mathsf{DT}(\rho_1) = \mathsf{DT}(\rho_2) \quad \Longrightarrow \quad \mathcal{B}(\rho_1) = \mathcal{B}(\rho_2).$$

Paste presence and content, reader outputs, and tool outputs need not agree across the attacks. The statement is trace-wise for every coupled coin sequence. In plain terms, the two runs may differ in whether pasted input arrives, its contents, and reader and tool outputs. The guarantee holds for every fixed sequence of random choices used identically in both runs.

Proof sketch. Maintain matched protected views, histories, coin positions, and transcript prefixes. The strategy then chooses the same next operation. A display-free step returns only a constant token, an operation-determined handle, or matched readable bytes, preserving the invariant. At a display step, the append-only transcripts and equality of the complete transcripts make the newly appended segments prefix-comparable. The interface's display-segment determinacy lemma forces those segments to be equal, including page structure, decisions, and released bytes. The two steps therefore create the same endorsed copies and replies. Induction gives the claimed equality; sink annotations follow the same mediation cases as Action Integrity. Full proof, including the segment lemma and batch cases: Supplement, Lemma S3.20 and Theorem S3.25. □

Corollary D.6 (Attack-surface confinement)**.** *Under the same comparison conditions, a difference in protected-agent operations, tool invocations, or ungated sink events entails a difference in the complete display transcript.*

Proof sketch. Take the contrapositive of Theorem D.5. □

This is a conditional noninterference guarantee. An attacker can affect a preview or an endorsement decision, and thereby the transcript and later behavior. The theorem excludes additional influence that bypasses that recorded release surface; it does not establish reasoning correctness or make displayed attacks harmless.

E Scope of the Lean 4 Verification

The supplementary archive `permit_formal.zip` contains the executable single-user interpreter, proofs, build assertions, dependency manifest, and axiom report. It pins Lean 4 and Mathlib to

v4.31.0 (Mathlib revision fabf563a); lake exe cache get && lake build reproduces the check. Supplement S9 gives the complete correspondence table, ten modeling deltas, and the specification corrections discovered during mechanization.

What is checked. action_integrity and action_integrity_from establish the single-user action guarantee from the empty endorsed store and from any state satisfying the mechanized invariant. reasoning_integrity and reasoning_integrity_from prove the transcript-conditioned comparison from empty or shared endorsed seeds, for arbitrary agent strategies and independently chosen attacks. Their conclusions include equal sink logs, operations, replies, and final protected views. Paste presence and content can differ. Supporting results check reply isolation, mandatory handle gating, and attack-surface confinement. A standalone batch layer checks displayed-element flow, rejection behavior, and transcript determinacy; executable assertions cover the baseline counterexample and eight scenarios.

What remains a paper proof. The run-level integration of batched iteration, durable endorsements from accepted positions of an aborted invocation, arbitrary mid-run matched starting states, and the reduction from reactive actors to event-indexed oracles are proved in the supplement, outside the checked interpreter. Options O1–O2 are checked at the gate level only; their wider integration, Option O3, the federated protocol, probability bounds, and confirmation-budget results are not mechanized. The checked interpreter returns one selected element at a direct collection-handle call and records additional transcript-determined selection information. These representation differences and their correspondence arguments are explicit in S9.

Trusted base and artifact. The supplied clean-build axiom report lists only propext and Quot.sound for the top-level theorems, with no sorryAx; the executable assertions additionally trust Lean's evaluator. Kernel checking certifies the mechanized statements, not the correspondence between that model and a deployed harness or the reliability of a verifier. The archive includes the full source, CORRESPONDENCE.md, pinned build instructions, the build log, and source checksums. Thus the concise statements here remain auditable against both the complete paper proofs and the precise mechanized scope.