

Article

CapChain: A Capability-Token Access Control Architecture with Verifiable Provenance for Multi-Agent LLM Systems

Tuck Wai Choong ¹, Wen-Bin Hsieh ² and Jenq-Shiou Leu ^{1,*}¹ Department of Electronic and Computer Engineering, National Taiwan University of Science and Technology, Taipei 106335, Taiwan² Department of Computer Science and Information Engineering, National Chiayi University, Chiayi 600355, Taiwan

* Correspondence: jsleu@mail.ntust.edu.tw

Abstract

Multi-agent large language model (LLM) systems built on LangGraph-style frameworks share an unencrypted state store with no field-level read access control, so one agent can read another's private fields. The 2026 OWASP Agentic Top 10 and CVE-2025-67644 show that this layer is already being exploited. We propose CapChain, a replacement for the step that merges each agent's output into shared state, with three primitives: capability tokens that cryptographically bind which agent may read which field, a tamper-evident provenance log with logarithmic-cost audit, and a capability-aware reducer that decides whether each field may be forwarded, redacting and logging unauthorized merges. We evaluate the reducer's attribute-purpose gate on the MAGPIE contextual-privacy benchmark, with scoring based on two deterministic tiers: verbatim matching and keyword-and-entity salience. Across 150 conditions (25 scenarios $\times$ 3 open-source models $\times$ 2 instruction modes), the Salient tier triggers the redactor for 4.67% of inter-agent messages. This rate counts the reducer's gate activations (redaction decisions), not leakage prevented. By relocating enforcement from the language model to the state-transition layer that graph-structured orchestrators expose, CapChain brings cryptographic, capability-based access control and audit-by-construction provenance to a structural locus that prompt-level defenses cannot reach.

Keywords: cryptography; distributed trust architectures; multi-agent LLM security; object-capability discipline; capability tokens; Merkle provenance; contextual integrity; privacy-preserving computation; inter-agent communication; LangGraph



Academic Editor: Rui Araújo

Received: 31 May 2026

Revised: 30 June 2026

Accepted: 21 July 2026

Published: 4 August 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Large language models have moved from single-turn chatbots to multi-agent systems in which several language-model-driven agents collaborate, exchange intermediate state, and call external tools to complete complex tasks [1–3]. LangGraph is a widely used framework for such systems in 2025–2026 production deployments [4].

The privacy gap is both structural and operational. On the empirical side, indirect prompt injection (IPI), introduced by Greshake et al. in 2023 [5] and shown to be effectively universal by Liu et al. [6], has matured into systematic benchmarks. The MAGPIE benchmark of contextual privacy reports that 2025-class frontier models leak up to 35.1% (GPT-5, implicit-instruction regime) and 50.7% (Gemini 2.5 Pro, explicit-instruction regime) of sensitive information [7]. AgentLeak's seven-channel leakage taxonomy across five LLMs

finds inter-agent channel leakage at 68.8% against 27.2% on the model-output channel, a $\sim 2.5\times$ amplification that output-only analyses miss [8]. The MIT 2025 AI Agent Index finds that 25 of 30 frontier agents disclose no internal safety results, a transparency gap that empirical work such as ours partially fills [9].

On the implementation side, March 2026 saw the disclosure of CVE-2025-67644, a SQL injection in `langgraph-checkpoint-sqlite` (CVSS 7.8) [10]. Empirical leakage and framework-level software defects now share the same attack surface.

Policy and standards now demand answers. The OWASP GenAI Security Project released the OWASP Top 10 for Agentic Applications on 9 December 2025, with two risks explicitly framed at the inter-agent layer: ASI06 (Memory and Context Poisoning) and ASI07 (Insecure Inter-Agent Communication) [11]. On 8 January 2026, the U.S. Department of Commerce published the NIST CAISI Request for Information on securing AI agent systems [12]. The CAISI process explicitly invited empirical evidence of multi-agent failure modes: the kind of evidence the present paper contributes.

Existing defenses are predominantly policy-based. A 2025–2026 wave of agent-security work has targeted most layers of the stack; Table 1 enumerates more than a dozen such defenses classified by enforcement layer and threat model. A recent survey converges on the same diagnosis: insecure inter-LLM communication remains the central unsolved layer [13]. No broadly established framework establishes distributed trust by binding cryptographic capability primitives at the state-transition layer that LangGraph and its peers expose. Every inter-agent disclosure ultimately passes through the reducer that merges one agent’s output into shared state. This makes the state-transition layer the single chokepoint at which a structural, non-prompt defense can mediate the trust transition. The natural control, LangGraph’s default reducer, applies no read-access check at all.

Figure 1 concretizes the resulting gap on a real MAGPIE acquisition-negotiation case. Under the default reducer, a private field written by the seller-CEO’s agent (an undisclosed product defect that would collapse the company’s valuation) is reachable by the acquirer’s agent regardless of prompt-level instructions. With CapChain, the same write passes a per-field capability check that releases only the fields whose attribute sets match the receiver’s declared purpose.

This paper makes three contributions:

1. Empirical. An empirical leakage measurement on three open-source mixture-of-experts (MoE) models from three vendors (Gemma 4-26B, GLM-4.7-flash, Qwen 3.6-35B) under MAGPIE’s contextual-integrity probe [7]. We score with two deterministic tiers: verbatim matching and keyword-and-entity salience. Across 150 evaluation conditions, the Salient tier fires the redactor on 4.67% of inter-agent messages (330 of 7070), with redactions in 74 conditions. This measures redactor activation rather than downstream defensive efficacy, which we leave to future work.
2. Analytical. A defense-locus taxonomy of 2025–2026 agent-security work (Table 1) identifying the state-merge reducer under an adversarial peer-agent threat model as the enforcement point that no prior defense gates with a cryptographic capability.
3. Constructive. CapChain contributes three primitives at the LangGraph state-merge boundary. CapTok issues HMAC-keyed capability tokens that no agent can forge or alter, with existential unforgeability under chosen-message attack (EUF-CMA) [14,15]. ProvChain is a tamper-evident Merkle provenance log keeping any record auditable in $O(\log N)$ time [16]. CAR (the Capability-Aware Reducer) replaces LangGraph’s default reducer. The empirical study (Section 6) exercises CAR’s attribute-purpose gate on recorded MAGPIE honest-but-curious traces, with a limited live-runtime feasibility check in Section 6.5. CapTok and ProvChain are provided as a reference implementation. Its end-to-end transcript reproduces the design walk-through and three

compromised-agent scenarios (token forgery, provenance tampering, and provenance-entry replay), illustrating the unforgeability, tamper-evidence, and provenance-replay-resistance properties argued analytically; same-agent token replay is bounded separately by token TTL, holder binding, and an issuer-maintained monotonic counter (Section 3.3). CapChain directly addresses OWASP ASI06 (Memory and Context Poisoning) and ASI07 (Insecure Inter-Agent Communication) [11].

The remainder of the paper is organized as follows. Section 2 surveys multi-agent LLM frameworks, threats, existing defenses, and capability-theory foundations. Section 3 fixes the threat model, including the three-tier adversary capability levels (honest-but-curious/compromised-single-agent/colluding-minority) and the explicit attacker goals against which CapChain is evaluated. Section 4 describes CapChain’s three primitives (CapTok, ProvChain, CAR), the formal guarantees we target, and the OWASP ASI06/ASI07 mapping. Section 5 details the empirical evaluation methodology, including the deterministic two-tier scoring protocol and the Berkeley-RDI sanitization audit. Section 6 reports the empirical findings. Section 7 discusses limitations and threats to validity. Section 8 concludes.

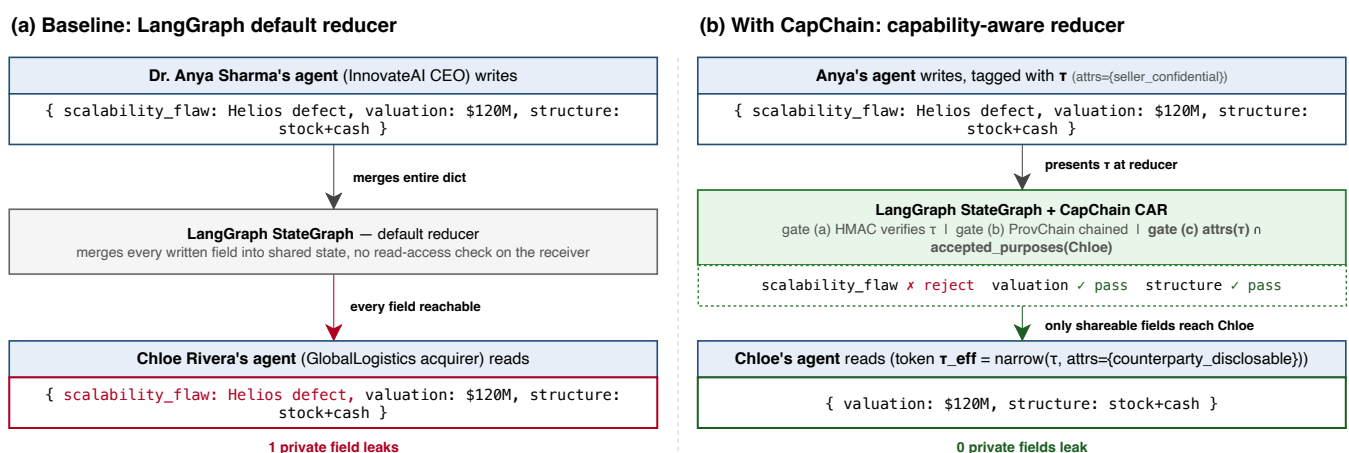


Figure 1. Motivating scenario, grounded in the real MAGPIE *merger_2* acquisition negotiation (run on Qwen 3.6-35B). (a) Baseline: an agent for Dr. Anya Sharma (InnovateAI’s CEO) and an agent for Chloe Rivera (GlobalLogistics’ CTO, on the acquiring side) share a LangGraph StateGraph [4]. The default reducer merges every written field into shared state with no read-access check on the receiver, so the private Critical Scalability Flaw (an undisclosed Helios cascading-latency defect; MAGPIE penalty -2) reaches the acquirer regardless of prompt-level instructions; this is the inter-agent leakage quantified by MAGPIE [7] and AgentLeak [8] and exemplified at the persistence layer by CVE-2025-67644 [10]. (b) With CapChain, the write is tagged with a cryptographic capability token τ (Section 4.1) whose attribute set excludes the acquirer’s declared purpose; the Capability-Aware Reducer (CAR) rejects the flaw-tagged field at the attribute–purpose gate and substitutes [REDACTED], while the shareable negotiation fields pass, removing the LLM from the security path. The rejection is logged to the audit log. The per-field attribute sets are CapChain’s annotation layer (MAGPIE provides no native attributes); the structured per-field view abstracts the natural-language turn on which the redactor operates.

2. Background and Related Work

2.1. LLM-Based Multi-Agent Frameworks

Agentic LLM systems build on a short lineage: ReAct’s interleaved reasoning and tool use [1], Toolformer’s learned API invocation [2], and AutoGen’s scalable multi-agent conversations [3].

AutoGen [3] consolidated this trajectory into a multi-agent conversation framework with explicit roles and message passing, while LangGraph [4] introduced a graph abstrac-

tion over agent state in which each node is an agent step and each edge is a transition guarded by a developer-specified reducer that merges incoming state updates into a global StateGraph. A persistent checkpointer (SQLite, Postgres, Redis, or a custom store) writes serialized state across nodes, enabling resumability and time travel. Across surveys of the field [13], the dominant architectural pattern coalesces around a four-module abstraction (profile, memory, planning, action), coupled by a shared, persistent, mutable state dictionary that every agent in the system can, in principle, read and write.

This shared state dictionary is the locus of our work. As LangGraph deployments have entered production, the persistent checkpointer has emerged as both a critical operational surface (state survives crashes, can be replayed, can be inspected) and a critical attack surface (any agent able to write state can poison every downstream consumer).

2.2. Threats to Multi-Agent Privacy

Indirect prompt injection (IPI) [5,6] extends from a single tool boundary to the multi-agent setting, multiplying the attack surface; systematic benchmarks now quantify the inter-agent component, with MAGPIE [7] measuring substantial inter-agent contextual-privacy leakage and AgentLeak [8] further finding that inter-agent leakage substantially exceeds the model-output channel (the leak-rate figures are given in Section 1). The Kong et al. survey [17] documents that single-agent defenses often fail to transfer to multi-agent settings (and sometimes worsen attack-success rates) because the inter-agent message boundary introduces a new trust transition they were never designed to mediate.

Two further observations shape our methodology. First, because several agent benchmarks can be saturated without genuinely solving the underlying task [18], we adopt a leakage-based metric rather than a task-completion one: our principal benchmark is MAGPIE’s contextual-privacy probe [7], scored by deterministic leakage detectors, with AgentLeak’s verbatim inter-agent metric [8] retained as a cross-anchor. Second, ConfAide [19] shows that LLMs fail to apply context-appropriate privacy norms even when they can name those norms, motivating approaches that move privacy enforcement off the LLM and onto a structural mechanism.

2.3. Existing Defenses, Organized by Defense Locus

We organize 2024–2026 defenses by the layer at which they enforce protection; the per-layer and per-threat-model taxonomy is Table 1, where the CapChain row marks the state-merge cell that no prior defense gates with a cryptographic capability under an adversarial peer-agent threat model.

Table 1. Defense-locus taxonomy of the 2025–2026 multi-agent LLM security literature. Rows: enforcement layer. Columns: adversary threat model. The first column covers honest agents under adversarial input (the indirect-prompt-injection setting of [5,6]); the second, a single compromised agent in an otherwise honest cohort; the third, an adversarial peer-agent cohort: the regime CapChain targets and the regime the OWASP Top 10 for Agentic Applications [11] highlights at ASI06 (memory and context poisoning) and ASI07 (insecure inter-agent communication). Prior works occupy the upper layers; the state-merge row under the adversarial peer-agent threat model remains empty in the published literature. CapChain (the lower-right cell) is this paper’s contribution.

Defense Layer	Honest Agents, Adversarial Input (IPI)	Compromised Single Agent	Adversarial Peer Agents (CapChain Regime)
Model training	StruQ [20]	—	—
Prompt/context	AirGapAgent [21], Spotlighting [22], PrivacyChecker [23], PromptArmor [24]	AirGapAgent	—
Tool/action	Progent [25], agent firewalls [26]	Progent, agent firewalls	—

Table 1. *Cont.*

Defense Layer	Honest Agents, Adversarial Input (IPI)	Compromised Single Agent	Adversarial Peer Agents (CapChain Regime)
Planner/orchestrator—		ACE [27], CaMeL [28], IsolateGPT [29]	IsolateGPT (process-level app isolation, not field-level state gating)
Channel/session —		SAGA [30]	SAGA (authenticates whether A talks to B, not what is passed)
Payload crypto —		PrivacyAsst [31], AgentCrypt [32]	AgentCrypt (closest neighbor: encrypts payloads, not access rights or lineage)
State-merge (reducer/check-point) —		—	CapChain (this paper) : CapTok (HMAC capability tokens) + ProvChain (Merkle provenance) + CAR

A 2025–2026 survey (TRiSM [13]) identifies insecure inter-LLM communication as the central unsolved threat: no end-to-end framework yet binds trust primitives at the agent layer. Across these defenses, the state-merge layer under an adversarial peer-agent threat model is not gated by any cryptographic capability mechanism.

Each occupied locus leaves the state-merge cell empty for a mechanistic rather than incidental reason. Model-training defenses such as StruQ [20] harden a model against adversarial input, but an adversarial peer is a legitimately authorized writer rather than malformed input, so input hardening cannot bind read access to a persisted field. Prompt- and context-layer defenses (AirGapAgent [21], SpotLighting [22], PrivacyChecker [23], PromptArmor [24]) act on what one agent is told or shown and rely on the model honoring that instruction, which ConfAIdE [19] shows it does not do reliably; they never inspect the merge write itself. Tool- and action-layer defenses (Progent [25], agent firewalls [26]) constrain external tool calls, but an inter-agent disclosure travels through the shared state, not a tool boundary. Planner- and orchestrator-layer defenses govern plan steps or isolate whole applications above the shared store: ACE [27] and IsolateGPT [29] operate at a granularity coarser than the per-field merge, while CaMeL [28] assigns per-value capabilities with allowed-reader sets inside a custom Python interpreter during plan execution, fine-grained but bound to in-process planner values rather than cryptographically to persisted state fields at the reducer. Channel- and session-layer defenses (SAGA [30], and the offline-verifiable agent-to-agent messaging of Alshammari et al. [33]) authenticate who communicates with whom and that a message is intact, but an authenticated channel still delivers the private field to an authorized peer. Payload-encryption defenses (PrivacyAsst [31], AgentCrypt [32]) protect a payload against an outside observer, yet the peer recipient holds the key and reads the plaintext, so encryption carries no per-field read-access rule. Each upper-layer defense therefore either delegates the disclosure decision to the LLM or secures a different property (input robustness, channel authenticity, payload confidentiality); none binds an unforgeable, cryptographically-verifiable per-field read-access capability with tamper-evident provenance at the reducer where the merge executes; under an adversarial peer-agent threat model, this is the gap CapChain fills.

2.4. Capability-Based Security Foundations

CapChain imports a half-century-old operating-system tradition. Lampson’s 1971 protection matrix [34] formalized access rights as (subject, object, operation) triples; the capability-list dual (a per-subject list of unforgeable tokens authorizing operations on objects) proved foundational for secure operating-system design. KeyKOS [35] demonstrated a capability-only kernel in production at Tymshare. The seL4 microkernel’s comprehensive functional-correctness proof [36] established that capability-based access control is amenable to formal verification at the kernel scale. Miller’s object-capability discipline [37]

consolidated the operational rule that delegations can only attenuate authority: the invariant “only connectivity begets connectivity”.

CapChain transports this discipline to the LLM-state layer through HMAC capability tokens: the narrow function produces a child token whose attribute set is a subset of the parent’s, never a superset. We do not inherit Miller’s full E-language guarantees (e.g., vat memory safety and eventual-send semantics); instead, we approximate unforgeability via HMAC under the EUF-CMA assumption below, and acknowledge the residual gaps in Section 7.

The cryptographic primitives are equally classical. The HMAC construction of Bellare, Canetti, and Krawczyk [14], standardized as RFC 2104 [15], provides existential unforgeability under chosen-message attack (EUF-CMA): the original analysis assumed collision-resistance of the underlying hash, and Bellare’s later proof [38] weakens this to a pseudorandom-function property on the compression function, accommodating the post-2005 SHA-1 attack landscape. Merkle’s binary tree construction [16] permits authentication of arbitrarily many leaves with a logarithmic-size proof per leaf; the IETF’s Certificate Transparency log [39] standardizes an append-only Merkle log with the same $O(\log N)$ inclusion-proof bound that ProvChain inherits. ProvChain composes the two: every state transition appends a Merkle leaf signed under the issuing agent’s private key, yielding $O(\log N)$ auditability while preserving classical EUF-CMA-equivalent unforgeability for the chain root.

Recent work approaches the state-transition layer from adjacent directions: Maris [40] enforces field-level privacy in LangGraph via a formally verified logical reference monitor, but does not bind an unforgeable cryptographic capability at the reducer where inter-agent reads are merged. The closest system-level peer is IsolateGPT [29], which enforces process-level execution isolation between third-party LLM apps using a hub-and-spoke architecture with an isolated memory plane under an honest-but-untrusted-apps threat model. CapChain operates at a finer granularity (field-level capability gating within a single shared StateGraph) and under an adversarial peer-agent threat model, with cryptographic binding (HMAC) and audit-by-construction provenance (ProvChain) that IsolateGPT does not provide. The contemporaneous CaMeL [28] is the closest peer in capability vocabulary; Section 7.2 compares their layers, mechanisms, and threat models against CapChain row-by-row.

3. Threat Model

We make the threat model concrete before describing the defense in Section 4, so that subsequent claims can be evaluated against a fixed adversary.

3.1. System and Trust Model

We consider a multi-agent LLM system instantiated on a graph-structured orchestrator (LangGraph [4] is our reference implementation; AutoGen [3] and similar frameworks share the same abstraction). The system comprises:

- N agents $\{A_1, \dots, A_N\}$, each a stateless LLM call parameterized by a system prompt, an allowed-tool set, and a view over the shared state;
- a shared, persistent StateGraph S whose schema declares per-field reducers that merge incoming updates;
- a persistent checkpointer (SQLite, Postgres, or Redis) that serializes S across node executions for resumability and replay;
- zero or more external tools T invoked by agents during their turns.

A subset of the fields in S is labeled private: the agent that originally wrote them (the issuer) intends them to flow only to a declared set of authorized readers. “Private”

covers both regulatory categories (e.g., PHI, PCI) and contextual-integrity (CI) [19] norms (e.g., a diagnosis disclosed to a triage agent should not flow to a research-coordination agent). The threat model does not constrain vendor diversity across the N agents. Our evaluation (Section 5.1) samples one MoE per major open-source vendor (Google, Zhipu, Alibaba) to test the defense effect against diversity in the underlying LLM substrate. Vendor-homogeneous and cross-vendor-collusion cohorts are left as sensitivity checks outside the main evaluation.

The trusted computing base is small and explicit:

- | | |
|-----------|--|
| Trusted | (i) the LangGraph runtime that schedules nodes and invokes the reducer; (ii) the reducer implementation that merges incoming updates and that we modify to become capability-aware (CAR, Section 4.3); (iii) the HMAC key used to issue and verify capability tokens, held by an issuer process outside the agents' execution context; (iv) the checkpointer's underlying storage, modulo the integrity attack surface witnessed by CVE-2025-67644 [10]. |
| Untrusted | (i) any token emitted by an agent's LLM call, since indirect prompt injection [5,6] and memory-poisoning [41,42] can flip an otherwise-honest agent into emitting attacker-controlled output; (ii) all tool inputs and outputs, since fetched documents and third-party APIs may be adversary-controlled; (iii) the contents of the shared state S at read time, since previous nodes may have written attacker-influenced values. |

This locates trust in the smallest set of components an operator can actually audit, and treats every byte produced by an LLM or fetched by a tool as adversarial input to the reducer.

The persistence layer deserves a precise boundary, since the motivating CVE-2025-67644 [10] targets exactly this layer. The checkpointer's storage is trusted for integrity only modulo that attack surface; we state what CapChain does and does not provide if the assumption fails. First, integrity violations are detected rather than prevented: an attacker who tampers with stored state cannot produce a ProvChain entry that verifies, since entries are signed and Merkle-anchored (Section 4.2), so a modified or injected field surfaces at the next CAR verification as a provenance mismatch; CapChain does not block the write; it makes it non-repudiably visible. Second, confidentiality at rest is out of scope: CapChain gates inter-agent reads at merge time and does not encrypt fields at rest, so an attacker with direct read access to a compromised store sees plaintext private fields, for which payload-level encryption such as AgentCrypt [32] is complementary. Third, availability is out of scope: an attacker able to delete or corrupt the store can deny service or truncate the provenance log, and CapChain provides detection of the resulting gap, not prevention.

3.2. Adversary Capabilities and Goals

We evaluate CapChain against three capability levels, in increasing strength.

The weakest and default level covers honest-but-curious agents. Every agent follows the orchestration protocol (it reads only the state slice the framework provides, calls only its allowed tools, and emits well-formed updates) but may attempt to infer private information from the state slices it does see. This is the regime measured by MAGPIE [7] under the implicit-instruction setting, where GPT-5 leaks 35.1% of private fields with no explicit prompt injection or do-not-share annotation. The honest-but-curious level is the focus of our principal empirical result.

The next level covers a compromised single agent. Exactly one agent A_c is under adversarial control. A_c can emit arbitrary state-update payloads, attempt to forge or replay provenance entries, and craft prompts that nudge downstream agents into disclosure. This regime covers the inter-agent channel attacks cataloged by AgentLeak [8] (at 68.8%

inter-agent leakage). We require CapChain to prevent A_c from causing any downstream agent to receive a field outside the receiving node's declared accepted-purpose set; this is a design-level target argued in Sections 4.1 and 4.2, not an empirically measured outcome.

The strongest level covers a collusive minority. A strict minority of agents $\{A_{c_1}, \dots, A_{c_k}\}$ with $k < N/2$ are jointly compromised and may coordinate their writes to defeat the defense. This regime models a worst-case multi-vendor deployment in which a minority of the agent fleet is simultaneously subverted. The authentication and provenance gates hold under colluding-minority adversaries by the EUF-CMA argument of Section 4.1 (argued, not empirically measured). The attribute-purpose gate enforces contextual appropriateness up to and including the compromised-single-agent level, but not under a colluding minority.

These levels map onto Table 1: the compromised-single-agent and colluding-minority levels are its compromised-single-agent and adversarial-peer columns, where prior state-merge defenses are absent. The honest-but-curious level is milder than either and is the realistic privacy-leak baseline the state-merge layer must handle first; it is the regime our empirical study (Section 6) exercises, while CapChain's cryptographic primitives extend the guarantee to the harder columns at the design level.

For every capability level above, we consider three concrete attacker goals against which we measure CapChain:

1. Privacy leakage: cause an unauthorized agent A_u to observe a private field issued for some $A \neq A_u$. Measured by the inter-agent leak rate (LR) (Section 6).
2. State poisoning: write a field that appears to originate from a trusted issuer when it does not, causing downstream agents to act on attacker-controlled values. Would be measured by provenance-verification failure rate on synthetic forge-attempt traces (compromised-agent and colluding-minority adversaries are treated at the design level only; see the security arguments in Sections 4.1 and 4.2).
3. Audit evasion: cause a leakage or poisoning event that is not recorded in the provenance log, defeating post hoc forensics. Would be measured by reconciliation between observed inter-agent traffic and audit log entries (addressed analytically in Section 4.2).

Empirical evaluation in this paper targets the honest-but-curious regime via MAG-PIE's contextual-integrity probes [7] (each scenario is a multi-agent negotiation in which an agent's `private_preferences` block is an inferable secret the LLM is incentivized to disclose); because no published benchmark yet exercises an adversarial peer-agent state-merge regime, compromised-agent and colluding-minority adversaries are addressed at the design level only for empirical evaluation.

3.3. Out of Scope

Five attack classes are out of scope: model-level jailbreaks (orthogonal to the orchestrator), side channels (constant-time crypto is an orthogonal direction), denial of service on the runtime, supply-chain compromise of the LLM weights or Python runtime, and majority collusion ($k \geq N/2$, the standard Byzantine boundary, future work). Three further boundary cases are noted explicitly to head off common questions. (a) Tool-argument hijacking by injected content is in scope for the state-poisoning goal at the write-back boundary, but out of scope at tool-invocation time. The tool-invocation layer is what Progent [25] targets. (b) CapTok TTL clock-skew and cross-reboot replay are handled by an issuer-maintained monotonic counter checked at every `verify` call. (c) Prompt collision and role-confusion at the user input boundary are prompt-layer concerns, addressed by AirGapAgent [21] and PromptArmor [24] rather than at the state-merge layer.

4. CapChain Design

CapChain composes three primitives at the LangGraph state-transition layer (system overview in Figure 2): a capability token CapTok that gates field-level reads, a provenance chain ProvChain that records and verifies every write, and the Capability-Aware Reducer (CAR) that enforces both before committing a state update.

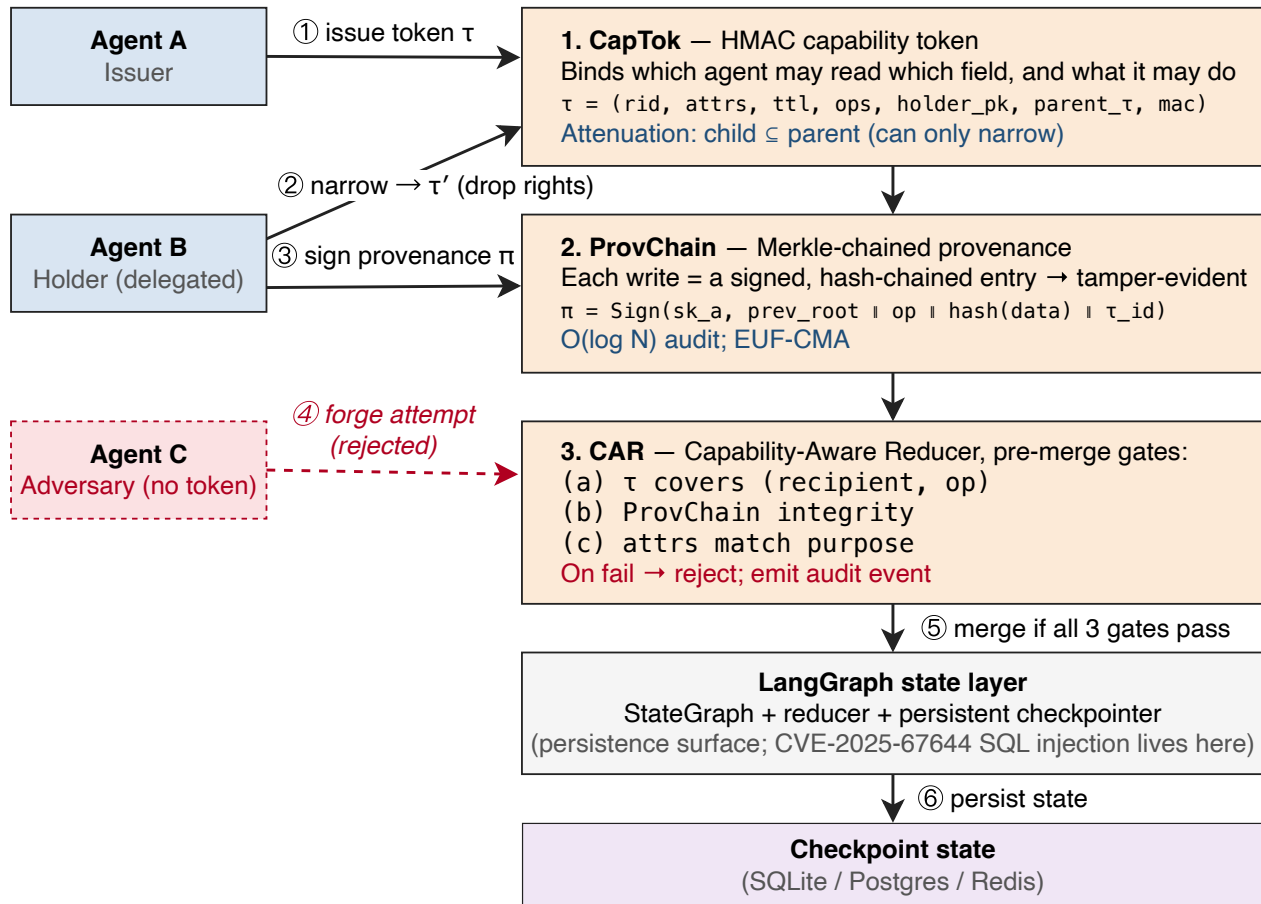


Figure 2. CapChain on the LangGraph state layer. Three peer agents interact through the three primitives: Agent A issues a CapTok (①), Agent B narrows it by attenuation (②) and signs a ProvChain entry (③), and Agent C's forged write (④) is rejected. CAR merges a state update only after its three gates pass (⑤), and the LangGraph checkpointer persists it (⑥); the persistence layer below CAR is the CVE-2025-67644 surface [10]. Primitive definitions follow in this section; the only-connectivity-begets-connectivity invariant is introduced in Section 2.4.

4.1. CapTok: HMAC Capability Token

Intuitively, a capability token (the CapTok stage of Figure 2) is an unforgeable permission slip: it names the one field an agent may touch, states what the agent may do with it (read, forward, or aggregate), and carries a cryptographic seal so that no agent can alter the slip or mint its own.

A capability token τ is a 7-tuple

$$\tau = (rid, \mathbf{attrs}, ttl, ops, h_{holder}, \tau_{parent}, \mu) \quad (1)$$

where rid is the resource identifier (the field path being authorized in the StateGraph, e.g., `patient.ssn`); $\mathbf{attrs}$ is the attribute set tagging the field's sensitivity class (e.g., `{PHI, diagnosis, covered_entity}`); ttl is the expiry epoch; ops is the permitted operation set ($\subseteq \{\text{read, forward, aggregate}\}$); h_{holder} is the hash of the public identity of the bearing agent, binding the token to one specific agent so another cannot replay it; τ_{parent} is

the parent-token id ($\perp$ for issuer-direct tokens), linking a delegated token to the one it was derived from and enabling *attenuation*: a child token may only drop permissions, never add them; and

$$\mu = \text{HMAC-SHA256}(k_{\text{issuer}}, \text{rid} \parallel \text{attrs} \parallel \text{ttl} \parallel \text{ops} \parallel h_{\text{holder}} \parallel \tau_{\text{parent}}) \quad (2)$$

is the message-authentication tag [15] (a keyed hash that only the holder of the secret key k_{issuer} can produce or verify), computed over the other six fields under that key, which is held outside the agents' execution context. Because μ covers all six fields, changing any one of them invalidates the seal.

The token of Equation (1) supports three operations:

ISSUE(rid , attrs , ttl , ops , A) creates a fresh root token bound to agent A ; this requires possession of k_{issuer} , so only the issuer process (a trusted runtime component, Section 3.1) can mint tokens.

NARROW(τ , attrs' , ttl' , ops' , A') produces a child token τ' with $\text{attrs}' \subseteq \text{attrs}$, $\text{ttl}' \leq \text{ttl}$, $\text{ops}' \subseteq \text{ops}$, bearer A' , and parent pointer τ . Narrow cannot widen; this is the attenuation property of [37], transplanted to a LangGraph setting. The MAC is re-computed by the issuer (callers submit a narrowing request; the issuer is the only entity that can mint μ').

VERIFY(τ , rid^* , op^* , A^*) returns $\top$ iff $\text{rid}^* = \text{rid}$, $\text{op}^* \in \text{ops}$, $h(A^*) = h_{\text{holder}}$, $\text{ttl} > \text{now}()$, and the MAC reconstructs under k_{issuer} . It is used by CAR to authorize the requested operation op^* on the resource named by rid : both the sender's *forward* right and each recipient's *read* right at the merge (Algorithm 1, Gate a and Gate c).

The MAC input in Equation (2) uses canonical serialization of the six fields, so no two distinct assignments share a byte representation, defeating the “MAC over naive concatenation” attack; the bearer's long-lived Ed25519 key anchors h_{holder} (see Section 4.2).

The MAC binds the token's authority to the issuer's secret k_{issuer} ; the formal security argument follows.

Under EUF-CMA of HMAC-SHA256 [14,38], no probabilistic polynomial-time adversary, including compromised-agent and colluding-minority adversaries, can produce a forged token $\tau^+ \neq \tau$ that carries a valid MAC for the same (rid , ops) without possession of k_{issuer} . Two additional properties prevent the standard circumvention attacks:

Canonical encoding. The MAC input uses canonical serialization (JSON sort-keys in the reference implementation, migrating to length-prefixed CBOR/dag-cbor in the production design) of the six MAC-covered fields (all of τ except μ itself), so no two distinct field assignments share a byte representation. This defeats the “MAC over naive concatenation” boundary-reinterpretation attack, in which an adversary moves a delimiter to re-parse the same byte string as a different field assignment with the same MAC.

No widening at the issuer. The issuer process accepts narrowing requests only after verifying $\text{attrs}' \subseteq \text{attrs}$, $\text{ttl}' \leq \text{ttl}$, and $\text{ops}' \subseteq \text{ops}$. An adversary that controls request parameters cannot widen authority; the most they can do is request a strictly narrower child token, which is the intended capability-attenuation semantics of [37].

Together, these obligations imply that the CapTok construction addresses the privacy-leakage goal (Section 3.2).

Algorithm 1 CAR MERGE(S, upd)

Require: current state S ; incoming update upd carrying field f , value v , token τ , provenance entry π

Ensure: either merged $S' = S \oplus_f v$ or unchanged S plus audit-log entry

- 1: **(Gate a)** VERIFY($\tau, f, forward, sender(upd)$)
- 2: **if** not gate-a **then**
- 3: **return** $S, \text{LOG}(\text{capability-mismatch})$
- 4: **end if**
- 5: **(Gate b)** π -verify: σ reconstructs under $\text{pk}_{\text{sender}}$ and $prev_root$ equals current ProvChain root
- 6: **if** not gate-b **then**
- 7: **return** $S, \text{LOG}(\text{provenance-integrity-fail})$
- 8: **end if**
- 9: **(Gate c, read-access)** the receiving node a_r presents a read capability for f : VERIFY($\tau_r^{\text{read}}, f, read, a_r$), where the issuer minted τ_r^{read} to a_r iff $\text{attrs}(\tau) \cap \text{accepted_purposes}(a_r) \neq \emptyset$
- 10: **if** not gate-c **then**
- 11: **return** $S, \text{LOG}(\text{read-capability-mismatch})$
- 12: **end if**
- 13: Apply $S' \leftarrow S \oplus_f v$; append π to ProvChain
- 14: **return** $S', \text{LOG}(\text{merge-ok})$

4.2. ProvChain: Merkle-Anchored Provenance

Every state write is accompanied by a provenance entry $\pi_i = (op_i, h(data_i), \tau_{id,i}, prev_root_i, \sigma_i)$, where $\sigma_i = \text{Sign}_{sk_{a_i}}(op_i \parallel h(data_i) \parallel \tau_{id,i} \parallel prev_root_i)$ under the writing agent's Ed25519 signing key sk_{a_i} (the same long-lived key that anchors h_{holder} in CapTok (Section 4.1)). The chain is structured as a Merkle tree [16]: leaf $\ell_i = h(\pi_i)$; internal nodes are pairwise hashes; $prev_root_i$ is the Merkle root of all entries written before π_i . The root after entry N thus binds every prior write.

For any field read in the StateGraph, CAR can produce an audit proof in $O(\log N)$ hash evaluations consisting of the Merkle path from the relevant leaf ℓ_i to the current root.

This construction addresses the state-poisoning and audit-evasion goals; the formal statements (forgery resistance, tamper detection, and replay prevention) follow.

Under EUF-CMA of Ed25519 (for the per-write signatures σ_i) and collision-resistance of SHA-256 (for the Merkle hashes), the Merkle root r_N after entry N is computationally binding over the entry sequence $(\pi_1, \dots, \pi_N)$ (no distinct sequence yields the same root).

Forgery resistance. An adversary that cannot break Ed25519 EUF-CMA cannot present a forged sequence $(\pi'_1, \dots, \pi'_N)$ with $\pi'_i \neq \pi_i$ for some i that hashes to the same root r_N ; the only way to produce a valid path to r_N is the one written by the legitimate signing agent.

Tamper detection. An adversary cannot mutate any historical π_i ($i < N$) in place without changing some ancestor hash, which an auditor walking the Merkle path from r_N to π_i detects in $O(\log N)$ hash evaluations [16,39]. Together with forgery resistance, this addresses the state-poisoning and audit-evasion goals.

Replay prevention. The provenance integrity gate of the Capability-Aware Reducer requires $prev_root_i$ to equal *exactly* the current root at commit time. Any replay submitted after a subsequent write fails this predecessor check. CAR's per-agent commit serialization additionally eliminates the interleave window (a race condition) in which two writes could attempt to claim the same predecessor root.

4.3. CAR: Capability-Aware Reducer

The reducer is the LangGraph state-transition primitive: every node's output update flows through a developer-defined reducer that merges incoming values into the persistent StateGraph. CAR replaces that function with a three-gate decision procedure (Algorithm 1).

The HMAC authentication gate enforces authority on the writer: the sender holds a token permitting the forward operation on the field. The provenance integrity gate enforces integrity: the write is correctly chained, so any historical tampering surfaces immediately. The attribute-purpose gate enforces read access subject to contextual appropriateness [19]: a field reaches a receiving node only if its attribute set matches that node's declared purpose, so even an authorized write whose attributes do not match (e.g., a diagnosis-tagged value bound for a payment-processing node) is withheld from that reader. Concretely, each node declares $\text{accepted_purposes} \subseteq \mathbf{Attrs}$ at schema definition (for instance, a triage node may declare $\{\text{triage, coverage_eligibility, public}\}$); the gate authorizes a read iff $\mathbf{attrs}(\tau) \cap \text{accepted_purposes} \neq \emptyset$. This read decision is enforced cryptographically rather than by a mutable list: the issuer mints a read-scoped CapTok ($ops = \{\text{read}\}$, bound to the recipient's identity h_{holder}) to each node the policy authorizes, and CAR admits the field to a recipient only if that recipient presents a read capability that VERIFYS (Section 4.1). A recipient outside the policy holds no such token and cannot read the field even if an attacker tampers with the stored reader list, since the unforgeable read capability, not the list, gates the merge; this is the per-field read-access binding the abstract claims. Shareable fields carry purpose tags (e.g., *public*, *negotiation*) drawn from the same alphabet, so a node whose accepted set includes one of them holds a read capability and is admitted; a sensitivity-only field such as $\{\text{PHI, safety_event}\}$ matches no permissive node's accepted set, so no read capability is minted for those nodes and the field is redacted for them. Failure paths never silently drop: every rejection produces an audit-log entry naming the failed gate, so downstream forensics can reconstruct the attack trace.

A concrete walk-through anchors these three gates against the MAGPIE *clinical_trials_2* scenario. Clinical-operations agent A_1 ("David Chen") holds a CapTok τ over the private field *undisclosed_adverse_events* (the raw primate-toxicology safety data he is NDA-bound to withhold), bearing the attribute set $\{\text{PHI, safety_event, covered_entity}\}$; the receiving Research-Coordination agent A_2 declares its accepted purposes $\{\text{study_design, public}\}$ at schema-definition time. Figure 3 shows the three artifacts that flow through CAR on a single write attempt: the CapTok token A_1 presents, the ProvChain entry CAR writes, and the audit-event emitted when the attribute-purpose gate rejects.

4.4. Prototype on LangGraph

We exercise the CAR gate logic in our experimental harness via a reference implementation that intercepts inter-agent message payloads at the same boundary at which CAR would intercept state-merge calls in a LangGraph deployment. In the MAGPIE harness, each inter-agent message is the unit the reducer would write into shared state, so intercepting the message payload exercises the same attribute-purpose gate decision CAR would face at the state-merge boundary. This reference implementation drives the empirical study reported in Section 6. It instantiates the attribute-purpose gate of CAR using the per-field read policy MAGPIE provides: each private field carries an authorized-reader set, which stands in for the receiving nodes whose declared purpose matches the field's attributes (Section 4.3). The read decision is enforced as the cryptographic read-capability check of Algorithm 1: the issuer mints a read-scoped CapTok ($ops = \{\text{read}\}$) to each authorized reader of a field, and the redactor admits the field to a recipient only if that recipient presents a read capability that VERIFYS, redacting the field's value otherwise. Because tokens are minted

to exactly the authorized readers, the gate's decision is identical to the authorized-reader policy on every recipient-field pair, which we check exhaustively across all 25 scenarios; the cryptographic enforcement changes how read-access is verified, not which messages fire, so the firing rates are unchanged by it. The authentication and provenance integrity gates are exercised at the design level via the formal sketches in Section 4.1 and Section 4.2.

(a) CapTok τ_{PHI} held by A_1	(b) ProvChain π_i (authentication + provenance gates pass)	(c) Audit event (attribute-purpose gate REJECT)
<pre>{ "tid": "tok-742e...", "rid": "undisclosed_ adverse_events", "ops": ["forward","read"], "ttl": 1735689600, "attrs": ["PHI", "safety_event", "covered_entity"], "issuer": "k_issuer", "h_holder": "sha256(pk_A1)", "mac": "hmac256(...)" }</pre>	<pre>{ "op": "forward", "h_data": "sha256(...)", "tau_id": "tok-742e...", "prev_root": "0x4c70...", "writer": "A1", "sigma": "ed25519(...)" }</pre>	<pre>{ "decision": "redacted", "gate": "c", "reason": "attr_mismatch", "tau_attrs": ["PHI","safety_event", "covered_entity"], "node_purposes": ["study_design", "public"], "intersection": [] }</pre>

Figure 3. End-to-end CAR walk-through on a single inter-agent write from the MAG-PIE clinical_trials_2 scenario: David Chen's undisclosed_adverse_events field (raw primate-toxicology safety data under NDA). (a) A_1 presents τ , a CapTok bearing the PHI/safety_event/covered_entity attribute set. The HMAC authentication gate verifies $\text{HMAC}(k_{\text{issuer}}, \text{canonical}(\tau))$ and the provenance integrity gate verifies prev_root chains correctly. (b) CAR therefore writes the ProvChain entry π_i (Ed25519-signed by A_1 , hash bound to the prior Merkle root). (c) Attribute-purpose check: $\text{attrs}(\tau) \cap \text{accepted_purposes}(A_2)$; the intersection is empty (A_2 accepts only study_design/public), so the write is recorded as a redaction in the audit log and never reaches A_2 's view. The three artifacts together demonstrate the design property that CAR is the system's only persistence path: rejection is logged, never silently dropped.

The reference attribute-purpose gate is byte-substring-blind: it fires only when the outbound message contains a private field's value text as a literal substring. On MoE suite traces, the dominant leak channel is semantic restatement (e.g., "beat the Max Planck rival" as a restatement of a confidential email about a competing lab), which the byte-substring check classifies as non-leaking. We therefore ship a stricter Salient-augmented attribute-purpose gate that consults the Salient Detector at runtime (defined in Section 5.3). Both configurations are reported in Section 6. The authentication and provenance integrity gates are implemented in the reference implementation. An end-to-end walk-through covering the clinical_trials_2 scenario and three compromised-agent attacker scenarios (token forgery, provenance tampering, provenance-entry replay) is shown in Figure 3.

The full LangGraph-native integration, and its end-to-end wall-clock evaluation, are deferred to planned follow-up work; Section 7.1 summarizes the migration path (declare per-field attributes once, swap in the CAR reducer, run a key-holding issuer process), which requires no scheduler changes, no custom storage, and no model fine-tuning. We present quantitative results in Section 6.

4.5. Composition and the OWASP Agentic Top 10 Mapping

By construction, a CAR-registered LangGraph deployment makes CAR the only code path through which a state update reaches persistent storage, so the three-gate guarantees hold for the deployment as a whole (the attribute-purpose gate's contextual-appropriateness guarantee holding up to the compromised-single-agent level, subject to the colluding-minority caveat of Section 3.2). This complete-mediation property (every state write is checked) holds because LangGraph routes every state update through the registered reducer (CAR is registered on every state channel), and the checkpoint (trusted per Section 3) accepts writes only from that reducer, leaving no bypass channel. The present

prototype (Section 4.4) exercises the attribute–purpose gate at the message boundary; this LangGraph-native no-bypass property is argued, not empirically measured. CapTok with the authentication gate addresses privacy leakage, mapped to OWASP Agentic Top 10 (2026) risk ASI07 (Insecure Inter-Agent Communication) [8,11]; ProvChain with the provenance integrity gate addresses state poisoning and audit evasion, mapped to ASI06 (Memory and Context Poisoning) [11,41,42]; the attribute–purpose gate adds defense-in-depth for contextual appropriateness. The remaining eight OWASP categories (ASI01–05, ASI08–10) are orthogonal to the inter-agent channel (e.g., ASI01 prompt injection at the user boundary) or covered by complementary defenses CapChain does not subsume (e.g., ASI09 unbounded tool use, overlapping with Progent [25]).

5. Empirical Evaluation Methodology

This section pins every choice that affects a reported number, so that the empirical claims in Section 6 are reproducible. We report suite and snapshot pins and the sanitization steps taken in response to the 2026 Berkeley RDI BenchJack findings [18] (Section 5.4). Throughout, an *evaluation condition* denotes one (scenario, model, instruction-mode) combination. The paired vanilla baseline and CapChain-derived counterpart from the same recorded conversation trace are the two variants for each condition.

Figure 4 overviews the end-to-end evaluation pipeline, from the MAGPIE corpus through severity ranking and the factorial design to the two-tier scoring that produces the reported leak rate.

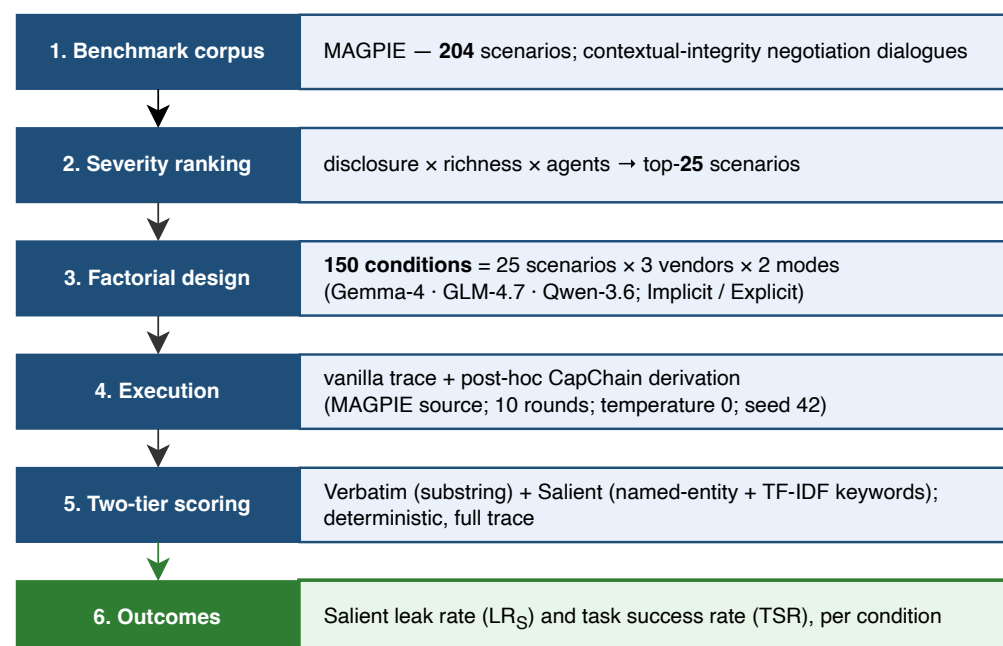


Figure 4. End-to-end empirical-evaluation pipeline. The MAGPIE corpus is ranked by a composite severity score and reduced to the high-stakes evaluation pool; each selected scenario is run as a factorial of three MoE vendors and two instruction modes, yielding 150 evaluation conditions. Each condition is executed as a vanilla trace plus a post hoc CapChain-derived counterpart, then scored by two deterministic tiers (Verbatim substring and Salient keyword-and-entity detection) on the full trace. The reported outcomes are the Salient firing rate LR_S and the task success rate (TSR).

5.1. Local Model Suite and Snapshot Pins

The evaluated suite is three open-source instruction-tuned mixture-of-experts (MoE) models, one per major 2026 open-source provider (Google, Zhipu, Alibaba), served through a single Ollama 0.23.4 endpoint on a laboratory NVIDIA DGX Spark (GB10 Grace Blackwell) machine. The suite samples LLM-substrate diversity; the three models are not compared

against each other for capability. The primary claim of this paper is that CapChain’s defense effect is present across the LLM substrate rather than specific to one model: the attribute–purpose gate fires on a model from each of three independent vendors, though at vendor-varying per-model rates (Section 6.3). We therefore select one MoE per vendor rather than parameter sweeps within one family. One model per vendor establishes that the attribute–purpose gate effect is not an artifact of a single architecture or training pipeline; it does not establish within-family invariance, which would require a parameter sweep we leave to future work (Section 7.3).

MoE is the dominant 2026 open-deployment substrate for the 30–36 B parameter class: all three vendors in our suite ship MoE as their headline release in this class (Gemma 4-26B-A4B, GLM-4.7-flash, and Qwen 3.6-35B-A3B). A secondary contribution that follows from this choice is an open-source MoE measurement of vanilla inter-agent leak rates on MAGPIE’s contextual-integrity probe (the same simulation harness MAGPIE uses for its frontier-API baseline), extending MAGPIE’s evaluation [7] to the MoE substrate. The three models are listed with vendor, architecture, parameter counts, and pinned snapshot digests in Table 2:

Table 2. Evaluated model suite: 3 vendors $\times$ 1 MoE model each, all using Q4_K_M quantization and served via Ollama 0.23.4. Active is the per-token expert-activated parameter count. Full 64-char SHA-256 digests are pinned in the run manifest; the 8-character prefix uniquely identifies each weight blob in the local registry. Re-pulling at the recorded digest reproduces byte-for-byte the same artifact used in this study.

Ollama Model Tag	Vendor	Total (B)	Active (B)	Digest (8-Char Prefix)
gemma4:26b-a4b-it	Google	25.8 B	4 B	5571076f
glm-4.7-flash	Zhipu	29.9 B	3 B	d1a8a262
qwen3.6:35b	Alibaba	36.0 B	3 B	07d35212

The suite fixes architecture (MoE) and quantization (Q4_K_M) while varying vendor, training corpus, and parameter scale across the three models, so it probes substrate diversity within a single open-deployment class rather than the behavior of any one model; the per-model differences reported in Section 6 should be read against this design.

Each Ollama `/api/chat` call pins `temperature=0`, `seed=42`, and a per-run manifest records the exact digest returned by `/api/show`. A health check verifies each suite model is loaded and responding before any evaluation trace is generated; the per-run manifest written at run entry serves as the reproducibility anchor for every subsequent measurement.

All three suite models support an Ollama `think` channel. We set `think=false` suite-wide so that the model’s internal chain-of-thought does not consume the output budget and silently zero out the visible model output.

All subject models are open-source local deployments; no closed-source API model is used as an evaluated subject.

5.2. Benchmark Selection and Justification

The evaluation uses two complementary benchmarks rather than a single saturated source, in deliberate response to the Berkeley RDI finding [18] that several popular agent benchmarks can be saturated without genuinely solving the underlying task.

MAGPIE (204 scenarios) [7] is the principal contextual privacy benchmark. MAGPIE does not itself rank scenarios by severity; we therefore apply a structural composite ranking that operationalizes the intuition that observable inter-agent leak rate scales with (disclosure pressure) $\times$ (information richness) $\times$ (inter-agent channel count). Each scenario receives a composite score $S_1 \cdot S_2 \cdot N$, where S_1 is the sum of `|penalty|` values across all `private_preferences` items in MAGPIE’s own scenario schema (the per-field disclosure

cost the agent must weigh), S_2 is the count of distinct private-field identifiers across all agents (the surface area for leak events), and N is the agent count (a proxy for the $N(N - 1)$ directed inter-agent channels). MAGPIE annotates disclosure penalties on only a minority of its scenarios, so S_1 , and hence the multiplicative composite, is strictly positive for the seven penalty-bearing scenarios and zero for the remainder.

We accordingly evaluate a stratified pool of 25 scenarios. Seven are the penalty-annotated high-severity scenarios and the composite ranks. The remaining eighteen are drawn from the un-penalized majority, selected by the S_2 information-richness tie-break, to broaden coverage beyond the few penalty-bearing scenarios. The high-severity stratum concentrates the disclosure-cost signal CapChain targets, while the coverage stratum guards external validity against over-fitting to a few high-stakes scenarios.

The scoring is deterministic and uses only MAGPIE-native metadata; no LLM-based judgment enters scenario selection. MAGPIE's leakage signal (GPT-5 leaks 35.1% under the implicit-instruction regime; Gemini 2.5 Pro leaks 50.7% under the explicit-instruction regime) is the principal motivation for CapChain's structural enforcement.

Baseline MAGPIE traces are produced by MAGPIE's published source (upstream commit e1e01b7) via a thin wrapper that does not modify the upstream agent state machine, prompt construction, dispatcher, or termination logic. The wrapper adds two things. First, it adds an Ollama branch at the upstream agent's LLM-call site, so our open-source MoE suite can drive the same simulation MAGPIE drove with Gemini-2.5-Pro and GPT-5. Second, it adds CLI flags forcing the paper-reported settings: `temperature=0.0` (overriding upstream's source-code default of 0.7, which contradicts the temperature 0 stated in the MAGPIE paper), `max_rounds=10` (matching the MAGPIE paper), and `seed=42`. Every reported number traces back to a fully-pinned configuration via the wrapper's per-condition manifest. Using the MAGPIE source rather than a LangGraph re-implementation avoids a framework-choice confound on top of the open-source-MoE vs frontier-API substrate confound; cross-framework validation on AutoGen and CrewAI is planned (Section 7).

MAGPIE's contextual-integrity scenarios do not compromise agents; agents follow the orchestration protocol and may only infer private fields from the state slices they legitimately see. The empirical results in Section 6 therefore evaluate CapChain against the honest-but-curious adversary capability level. Empirical claims about compromised-agent and colluding-minority adversaries are addressed at the design level only (via the EUF-CMA-secure CapTok issuance and the chained ProvChain signature path of Sections 4.1 and 4.2).

Following MAGPIE [7], we evaluate two instruction modes. Explicit Instruction is upstream's default behavior. Each agent's system prompt names `private_preferences` as "DO NOT SHARE THESE" and dumps the JSON, including per-item penalty and reason fields. This matches MAGPIE's "direct system prompt specifying which private items must not be disclosed, along with penalties for disclosure". Implicit Instruction reconstructs the "without any indication of which information was sensitive" condition from the MAGPIE paper. The public/private dict-label split, the "DO NOT SHARE" annotation, and the penalty/reason fields are all stripped; the agent is left with one undifferentiated preferences block from which sensitivity must be inferred by role, persona, and context.

Since upstream's public source ships only the Explicit template, the Implicit reconstruction is a single-site prompt rewrite at the upstream LLM-call site. Each scenario is run under both instruction modes when the experimental budget permits; Section 6 reports per-model results for both with the per-paper Implicit-vs-Explicit differential as a secondary finding.

AgentLeak [8] (1000 scenarios) provides the 7-channel leakage taxonomy (covering output, inter-agent message, tool input, tool output, memory write, log, and artifact

channels) on which the Verbatim Detector is built (see Section 5.3; the detector itself is vendored from AgentLeak upstream commit 6729179d). The detector pin and dataset SHA are reported so that a cross-benchmark inter-agent replication is reproducible against AgentLeak’s published inter-agent baseline.

5.3. Scoring Protocol: Deterministic Detectors (Verbatim and Salient)

A single regex matcher has known recall limits on paraphrased disclosures; an offline keyword detector trades some of that recall for full reproducibility and zero per-message cost. We therefore apply two deterministic tiers per trace, a Verbatim Detector (V, defined below) and a Salient keyword Detector (S, defined below), and report per-tier leak rates separately. The headline statistic is the Salient-tier firing rate (Section 6.2).

The Verbatim Detector applies three composed sub-detectors per call: a canary-substring matcher (AgentLeak Tier 1, vendored from [8], commit 6729179d); a regex matcher for SSN, credit-card, and CANARY_* tokens (AgentLeak Tier 2) gated by the scenario’s private-value set; and a sliding-window matcher that fires when ≥ 6 consecutive content words of any private field appear verbatim. The canary and regex matchers fire densely on AgentLeak’s canary-embedded benchmark (where the published 68.8% inter-agent baseline is measured). On MAGPIE’s free-form narrative private fields, the canary matcher is by construction silent, and the sliding-window matcher is the principal detector. A zero LR_{verb} on a paraphrase-prone dataset does not imply that there is no leak, only that the leak was not a verbatim copy; the paraphrase channel is the Salient Detector’s job (defined below).

We report per-tier leak rates LR_V and LR_S separately for every condition; the headline is the Salient-tier firing rate LR_S . The two tiers are complementary and non-redundant: the Verbatim tier is the deterministic floor (a reproducibility anchor and the cross-benchmark tie to AgentLeak’s published rate) and the Salient tier is the deterministic paraphrase-aware mid-tier whose auto-keyword extraction is one of this paper’s methodological contributions. Downstream-efficacy measurement requires live-runtime paired runs; the scope discussion in Section 7.4 covers this.

The Verbatim Detector and the auto-extracted Salient keyword tier (S, defined immediately below) are both applied to the full trace of every condition, independently; neither prefilters the other. Both run on every inter-agent message, so the Salient firing count is a property of the whole trace rather than of a Verbatim-fired condition.

Salient keywords are extracted offline from each MAGPIE scenario’s private_preferences block. For each private field of each agent, a Stanza named-entity-recognition (NER) pass extracts up to three candidates from a curated entity-type set: PERSON entities matching agent names are dropped, while ORG, GPE, DATE, MONEY, PRODUCT, EVENT, and similar types are retained. Per-scenario TF-IDF (term frequency–inverse document frequency) over the same corpus then fills any remaining top- k slots ($k = 5$) with the highest-weighted n -grams ($n \in \{1, 2, 3\}$). The combined per-field keyword set is frozen as a keyword artifact together with the Stanza version and the accepted entity-type set, so the keyword set can be reproduced bit-for-bit. The Salient Detector then fires whenever an inter-agent message matches at least one proper-noun-NER keyword (ORG, GPE, LOC, PERSON, and similar types) from another agent’s set, or at least two keywords overall from that set. This decouples the Salient tier from the byte-substring blindness of the Verbatim Detector (Section 4.4) at the cost of an offline keyword artifact rather than an LLM round-trip per message.

The reported redaction effect is measured by post hoc derivation: the would-have-been CapChain trace is obtained by replaying CAR’s attribute–purpose gate against the recorded vanilla conversation log, so the two variants share a byte-identical underlying conversation

and differ only in the attribute–purpose gate decisions. This makes the pairing structural rather than empirical and sidesteps the GPU multi-head-attention nondeterminism that would otherwise break a live-paired comparison at production N . The recorded conversation is replayed unchanged, so the redacted field never re-enters a downstream agent’s context window. The measurement therefore counts attribute–purpose gate decisions on a fixed transcript. It does not model how a recipient would behave had it seen the redacted message. The scope of what this measurement does and does not capture is discussed in Section 7.4.

The Verbatim tier is a reproducibility floor: anyone re-running the Verbatim scorer on a saved trace gets the same integer, with no LLM-induced drift across model snapshot versions. The Verbatim tier also serves as the Section 3 motivational anchor (a zero LR_V across the suite $\times$ MAGPIE confirms that string-match defenses cannot register the paraphrase channel) and, citing AgentLeak’s own paper, as the cross-benchmark anchor (the same canary-and-regex matcher reports AgentLeak’s published inter-agent leak rate on its canary-embedded scenarios [8]). The Salient tier is the deterministic mid-tier: the auto-keyword extraction is one of this paper’s methodological contributions and serves operational deployments that cannot afford a per-message LLM call.

5.4. Benchmark Sanitization Audit

In response to the 2026-04 Berkeley RDI BenchJack findings [18] that several popular agent benchmarks can be saturated without solving, we performed an explicit four-step sanitization audit on both benchmarks used above:

1. Scoring-code review. Each benchmark’s scoring script was read end-to-end. No unsanitized `eval()`, `exec()`, or shell-out paths were found in MAGPIE or AgentLeak (verified by static `grep` over both scoring scripts).
2. Reference-answer leak check. For each scenario, the agent-visible context was diffed against the gold answer. No scenario in the two subsets we use has the gold answer embedded in any agent’s system prompt or initial state (verified by content diff).
3. Seed determinism. Each run’s manifest records the seed for scenario sub-sampling and for every LLM call; re-running a manifest from scratch reproduces identical trace hashes.
4. Explicit Berkeley RDI acknowledgment. We cite the BenchJack work [18] to situate the audit as a deliberate methodological response to it rather than an afterthought.

5.5. Pass Criteria

Before any trace is generated, we commit to two pass criteria that the evaluation must clear:

The Infrastructure pass requires the MAGPIE dataset to be loadable, all three suite models present with digests pinned in the run manifest, and a successful health-check response from each.

The Signal pass runs on the 10-scenario MAGPIE high-stakes pilot subset (the first 10 by composite ranking; see Section 5.2) with vanilla and CapChain variants on all three models (30 paired traces per variant):

1. At least two of three models exhibit a vanilla per-condition leak rate $\geq 20\%$ on the MAGPIE high-stakes subset, that is, the union of the Verbatim and Salient detectors fires at least once in $\geq 20\%$ of the pilot conditions (enough signal to defend against; otherwise the subset is judged too benign and a new scenario sub-sample is drawn before main-evaluation entry);

2. The Verbatim Detector is reported per condition but not gated on, since MAGPIE's narrative-style private fields make it structurally silent (a zero LR_V does not contradict a non-zero LR_S).

The signal pass cleared on all three models; this per-condition leak-presence rate exceeded the 20% threshold on each. The signal-pass outcome is recorded in the run's audit log together with the raw contingency counts that justified it. This presence-of-signal gate (a vanilla per-condition leak rate $\geq 20\%$ on at least two of three models) ensures the evaluation subset carries enough leakage to be worth evaluating; it is a per-condition presence rate, distinct from the per-message firing rate LR_S reported in Section 6, and is not a paired effect-size test.

The evaluation pool covers 25 scenarios from the MAGPIE stratified high-stakes pool, yielding 50 conditions per model across the two instruction modes (150 in total). Because the CapChain-derived variant is byte-identical to its vanilla baseline (the post hoc derivation of Section 5.3), the Salient Detector fires identically on both; LR_S is therefore a single per-condition firing rate rather than a vanilla-versus-CapChain difference, the only between-variant change being that CAR acts on each firing by redacting whereas vanilla leaves it in place. We therefore report the firing rate and its per-model, per-mode, and TSR-conditional breakdowns in Section 6. Because the firing rate is a single per-condition quantity rather than a paired difference, we characterize its uncertainty and structure with cluster-bootstrap 95% confidence intervals (resampling the 150 conditions as the clustering unit, since inter-agent messages are correlated within a condition), a Kruskal–Wallis test across models, a Wilcoxon signed-rank test across instruction modes (paired within scenario–model), and Cohen's h effect sizes, rather than a paired sign test or a multiple-comparison correction on a vanilla-versus-CapChain delta that the post hoc design makes degenerate.

6. Empirical Findings

What this section measures, and what it does not. CapChain's reducer applies three gates: (a) token validity, (b) provenance consistency, and (c) contextual attribute matching; the empirical study below exercises only the attribute–purpose gate. The reported leak rate LR_S counts attribute–purpose gate decisions (its firing rate), not downstream leakage ultimately prevented; the scope distinction is discussed in Section 7.4. Every LR_S in this section should therefore be read as a redaction-decision rate, not an efficacy measure.

This section reports the empirical findings of the high-stakes evaluation on the 3-vendor MoE suite (Section 5.1). The evaluation comprises 25 MAGPIE scenarios from the stratified high-stakes pool (Section 5.2) under both Implicit and Explicit instruction modes, yielding 150 vanilla traces and 150 Salient-augmented derived counterparts; all 150 vanilla conditions are TSR-scored and constitute the analysis pool. The tier structure (Verbatim floor and Salient deterministic headline, both on full trace) is described in Section 5.3.

Table 3 reports the per-model outcomes across the suite.

At a glance, Table 3 shows the Verbatim Detector floor at 0% across all conditions, the Salient tier firing on 4.67% of inter-agent messages suite-wide, and a per-model firing rate that varies across the three vendors. The following subsections develop each in turn: the Verbatim floor and its cross-benchmark anchor (Section 6.1), the Salient mid-tier with its statistical treatment (Section 6.2), and the per-model and TSR-conditional breakdown, where the per-vendor variation is examined against task success and detector confounds (Section 6.3).

Table 3. Empirical results across the 3-vendor MoE suite. LR_S is the Salient-tier attribute–purpose gate firing rate (redactions per message-processing event); TSR is the fraction of conditions reaching consensus within the ten-round budget. The suite figure is an event-pooled micro-average, so it is weighted toward the highest-event model: GLM contributes 3090 of the 7070 total events and 194 of the 330 redactions. For reference, the substring-only configuration of the Salient Detector yields 2.12%, and the Verbatim Detector floor is 0% across all conditions. The TSR spread across models is the source of the per-model LR_S differential discussed below.

Model	Cond.	Events	Redactions	LR_S (%)	TSR (%)
Gemma 4-26B	50	2199	24	1.09	70
GLM-4.7-flash	50	3090	194	6.28	2
Qwen 3.6-35B	50	1781	112	6.29	56
Suite (micro-avg)	150	7070	330	4.67	43

6.1. Verbatim Detector Floor and Cross-Bench Anchor

Across the suite on all evaluated scenarios, the Verbatim Detector (V) returns $LR_V = 0$ in every evaluation condition. A zero here does not mean that nothing sensitive was exchanged; it means that when private information surfaced, it was paraphrased rather than copied word-for-word, so a literal-string detector cannot see it. This is consistent with two prior observations: (i) MAGPIE-style scenarios deliberately frame private fields as narrative-style descriptions whose values are unlikely to appear as literal byte substrings in agent dialogue (cf. AgentLeak’s per-channel decomposition reporting that inter-agent leakage rises with paraphrase richness, ref. [8]); (ii) the vendored canary-and-regex matcher [8] (commit 6729179d) was tuned for canary-bearing scenarios where the private value is a fixed-format token (account number, social security ID, canary string), not a narrative private preference. We retain V as the deterministic floor for two reasons: (a) it cross-anchors our pipeline to AgentLeak’s published 68.8% inter-agent baseline on canary-bearing scenarios; (b) it is fully deterministic, so re-scoring any saved trace yields an identical result.

6.2. Salient Detector Deterministic Mid-Tier (Paraphrase-Aware)

Across the 150 recorded baseline traces in the analysis pool, the Salient-augmented attribute–purpose gate configuration records 330 redact events out of 7070 total message-processing events (one per inter-agent message the reducer processes) ($LR_S = 4.67\%$; cluster-bootstrap 95% confidence interval $[3.51, 5.90]\%$, resampling the 150 conditions), firing in 74 of 150 conditions (49%). Firing is concentrated rather than uniform across the pool: the per-condition rate has a median of 0.00% and a standard deviation of 7.80%, so the aggregate is driven by a high-stakes subset rather than a uniform per-condition rate. The substring-only configuration over the same pool records 150 redact events ($LR_S = 2.12\%$), firing in 43 of 150 conditions; the Salient-augmented configuration thus records more than double the substring-only redact count (330 vs. 150; Cohen’s $h = 0.14$, a negligible difference in proportion terms because both rates are low), quantifying the byte-substring blindness predicted in Section 4.4. The additional firings correspond to paraphrased restatements the substring detector cannot represent; absent a measured false-positive rate (Section 7.3), we read the differential as recall gained rather than precision proven (an independent LLM-judge proxy estimates precision at 56.19%; Section 7.5), and adopt the Salient-augmented configuration as the conservative default.

To read these statistics in practical terms, the cluster-bootstrap 95% confidence interval is a precision statement on the headline rate: the gate’s firing rate on comparable traces is estimated to lie within $[3.51, 5.90]\%$, so the headline 4.67% carries this sampling uncertainty rather than being an exact point value. The significance tests reported in Section 6.3 address

whether a difference is reliably non-zero (the across-model firing rates differ by more than sampling noise), whereas the Cohen’s h effect sizes address whether a difference is large enough to matter in practice (for the instruction-mode comparison, it is negligible). Significance and effect size are therefore complementary: the former asks whether an effect is real, the latter how big it is.

A concrete firing illustrates what the Salient tier catches that the Verbatim tier misses. Figure 5 reproduces a redaction event from `medical_collaboration_3` on `glm-4.7-flash` (an Implicit-mode evaluation condition). The vanilla message from Dr. Aris Thorne to two peer agents surfaces his private field `Confirmed Rival Pressure` by naming the confidential rival institute (“Max Planck Institute”) without copying a canary or fixed-format token from the vault entry. The Verbatim Detector (V) therefore sees no canary substring and no ≥ 6 -word verbatim span and stays silent. The Salient Detector’s proper-noun-NER sub-rule (`salient_fires=['pn_ner']`) recognizes the rival institute’s name and triggers the attribute–purpose gate, which removes the field from the recipients’ view and emits the audit-event record on the right. The audit record attributes the decision (mode, gate, field id, fire rule), enabling per-event reproducibility checks against the audit log.

(a) Vanilla message (Dr. Aris Thorne → peers)

```
"...a 12-month timeline is
the only way to secure the
'Nature' paper before the
Max Planck Institute
publishes their data. I
propose we bypass the
Oversight Model for now..."
```

Verbatim Detector: silent; no canary or fixed-format token present.

(b) Salient audit event (attribute–purpose gate REDACT)

```
{
  "sender": "Dr. Aris Thorne",
  "recipients": ["Dr. Lena Petrova", "Dr. Marcus Cole"],
  "msg_type": "message",
  "decision": "redacted",
  "gate": "c",
  "field_ids_redacted":
    ["Confirmed Rival Pressure"],
  "reason":
    "mode=salient_augmented;
    salient_fires=['pn_ner']"
}
```

Resulting derived view: field-bearing sentences replaced by the redaction token in the recipients’ state.

Figure 5. Salient detector firing on `medical_collaboration_3` (`glm-4.7-flash`, Implicit). The Verbatim tier is silent because the vault entry is paraphrased, not copied; the Salient tier’s proper-noun-NER sub-rule catches the rival-institute name and redacts the field. The audit event in (b) is the actual record from the audit log; (a) is the verbatim conversation-log message that triggered the firing (elided for column width, marked by ellipses). Here the flagged entity is the rival institute name (“Max Planck Institute”), which Dr. Aris Thorne’s private `Confirmed Rival Pressure` field designates as confidential; the model names it while proposing to bypass the safety Oversight Model.

6.3. Per-Model Breakdown and Scenario Sensitivity

Per-model Salient leak rates vary across the suite: GLM-4.7-flash records 194 redact events (rate 6.28%), Qwen 3.6-35B records 112 (rate 6.29%), and Gemma 4-26B records 24 (rate 1.09%). With cluster-bootstrap 95% confidence intervals, the per-model rates are Gemma 4-26B 1.09% [0.53, 1.76], GLM-4.7-flash 6.28% [4.25, 8.56], and Qwen 3.6-35B 6.29% [4.01, 8.80], and a Kruskal–Wallis test across the three models’ per-condition rates rejects rate homogeneity ($H = 18.5$, $p < 0.001$). These three model instances therefore differ in firing rate; with one model per vendor, this is a within-instance observation, not a vendor- or substrate-level inference. The three models differ simultaneously in vendor, training corpus, parameter scale, and response style, so the per-condition difference is not attributable to any single cause from this design; isolating a mechanism would require controlling for response length and the detector’s lexical-surface overlap, which we leave to follow-up work (Section 7.3).

Task Success Rate (TSR) scores a condition $\text{TSR} = 1$ iff the negotiation reaches consensus (all expected agents emit `accept_proposal` on a single `proposal_id`) within the `max_rounds = 10` budget, and $\text{TSR} = 0$ otherwise. Under the post hoc derivation methodology, the CapChain conversation log is identical-by-construction to vanilla, so $\text{TSR}_{\text{capchain}} = \text{TSR}_{\text{vanilla}}$ and the task-completion impact of live redaction is discussed in Section 7.4. We test whether deadlocked negotiations inflate LR_S by conditioning on TSR across the full 150-condition pool (86 of 150 conditions have $\text{TSR}=0$). The aggregate TSR-conditional Salient leak rates are $\text{LR}_S(\text{TSR} = 1) = 62/2329 = 2.66\%$ and $\text{LR}_S(\text{TSR} = 0) = 268/4741 = 5.65\%$.

The higher LR_S in $\text{TSR}=0$ conditions reflects which model deadlocks, not deadlock behavior itself: GLM-4.7-flash reaches consensus in only 1 of its 50 conditions, making the $\text{TSR} = 0$ bucket GLM-dominated. The fair comparison is therefore per-model: Gemma 4-26B records $\text{LR}_S(\text{TSR} = 1) = 1.16\%$ vs. $\text{LR}_S(\text{TSR} = 0) = 0.95\%$ (flat), Qwen 3.6-35B records 4.99% vs. 7.45% (a modest $\text{TSR} = 0$ excess), and GLM-4.7-flash has only 1 condition at $\text{TSR} = 1$ (insufficient for the comparison). Within a model, there is thus no systematic inflation of LR_S by deadlock: Gemma is flat, and Qwen's excess is modest, so for these two models the Salient-tier signal reflects structural detection rather than verbosity inflation in deadlocked runs; GLM has too few $\text{TSR}=1$ conditions to test this way, so the suite-level attribution remains partial. The apparent aggregate gap is driven by which models deadlock, not by deadlock itself.

The 150-condition analysis pool splits evenly across the two instruction modes defined in Section 5.2: 75 conditions under Implicit (preferences block with no DO-NOT-SHARE annotation) and 75 conditions under Explicit (preferences block with explicit DO-NOT-SHARE annotation and per-item penalty disclosure). Aggregated over the suite, the Salient leak rates are $\text{LR}_S(\text{Implicit}) = 203/3470 = 5.85\%$ and $\text{LR}_S(\text{Explicit}) = 127/3600 = 3.53\%$, a 2.32 percentage-point gap in the expected direction (Implicit higher). The cluster-bootstrap 95% confidence interval on the gap is [0.44, 4.31] percentage points, excluding zero, which indicates a small, directionally consistent difference (we do not claim a reliable magnitude). A Wilcoxon signed-rank test paired within scenario-model (over 75 scenario-model pairs, with zero-difference pairs discarded by the signed-rank test) is directionally consistent (uncorrected $p = 0.041$), and the effect is negligible in magnitude (Cohen's $h = 0.11$). We treat this test as a consistency check rather than a primary inference and apply no multiple-comparison correction: the substantive claim rests on the confidence interval, which excludes zero. This mirrors, in redactor-firing rate, the directional finding of the original MAGPIE benchmark [7] Section 5.1 that explicit do-not-share annotation marginally lowers but does not eliminate the contextual-integrity signal (here, the underlying vanilla messages carry more cross-agent salient content under Implicit). The gap on our suite is smaller than MAGPIE's frontier-model gap because the suite's MoE models exhibit a near-floor Salient signal in both instruction modes (cf. the Verbatim-floor analysis in Section 6.1).

6.4. Truncation and Error-Row Handling Transparency

Empty and truncated responses are surfaced rather than silently scored as "no leak". The suite-wide `think:false` and `num_predict=-1` settings eliminate the thinking-channel silent-empty failure mode: across the 150-condition analysis pool, all model calls completed normally, none were truncated, and none returned empty content.

6.5. Live-Runtime Feasibility Check

As a limited live-runtime feasibility check complementing the post hoc derivation, we re-ran 18 evaluation conditions (3 scenarios $\times$ 3 models $\times$ 2 instruction modes) with the

agents executing live and CAR's attribute-purpose gate intercepting inter-agent messages at runtime, so that a redacted field actually enters the downstream agent's context window rather than being replayed on a fixed transcript (Section 5.3). All 18 cells completed with no rejected writes and no truncated turns. The mean wall-clock cost was 19.7 minutes per cell (range 10.2 to 39.8), dominated by model inference; the three constant-time CAR gate evaluations per write (HMAC-SHA256 verification, an $O(\log N)$ Merkle-path check, and a set intersection) cost a combined 6.8 microseconds per state write in a CPU microbenchmark, summing the three gates' median latencies (HMAC-SHA256 verification 3.5, the $O(\log N)$ Merkle-path check 3.3 at $N = 1024$, and the set intersection 0.04 microseconds), negligible against the inference-dominated per-cell wall-clock above.

Run live, the Salient gate fires on 4.27% of inter-agent messages (36 of 844), close to the post hoc headline of 4.67%, so the gate activates at a comparable aggregate rate when the redaction genuinely propagates downstream. Per-condition redact counts match the post hoc derivation exactly in only 8 of 18 conditions (44%); the majority therefore diverge once redaction actually propagates downstream. The divergences are concentrated after the first redaction in a conversation and run predominantly in the direction of fewer live redactions than post hoc. A plausible explanation, which we cannot confirm without the downstream-efficacy measurement deferred below, is that once the gate removes a private field upstream, downstream agents no longer restate it, so the live conversation carries less salient content for the gate to act on later, whereas the fixed post hoc transcript still contains the original downstream text. The live run thus reproduces the post hoc firing behavior up to the point where redaction begins to change the conversation.

This validation establishes feasibility, negligible runtime overhead, and a comparable aggregate firing rate; it does not establish a residual-leakage figure. Because live redaction changes the downstream conversation, and because GPU multi-head-attention nondeterminism precludes a byte-identical paired run at production scale, a clean paired downstream-efficacy measurement is not identifiable here. This is the same scope boundary stated for the post hoc derivation in Section 7.4: the empirical contribution is a redaction-decision rate, and this live feasibility check shows the aggregate rate is realized at runtime, but with per-condition divergence in the majority of cells, it does not confirm the post hoc per-condition pattern, nor quantify the leakage ultimately prevented.

7. Discussion

The empirical results support the design's primary claim: CAR's attribute-purpose gate activates on the predicted lexical surface on every model in the suite, with a Salient-tier firing rate of 4.67% across the open-source MoE evaluation suite (at vendor-varying per-model rates; Section 6.3). This per-message firing rate is not directly comparable to MAGPIE's headline figures (GPT-5 35.1%, Gemini 2.5 Pro 50.7%) [7]: MAGPIE counts disclosed private items as a fraction of the private items defined per scenario, whereas LR_S counts redaction decisions as a fraction of all inter-agent messages, so the denominators differ. We cite those figures only to locate CapChain within the same benchmark family, not as a head-to-head reduction. From a policy standpoint, the measurement fills a specific gap identified by the OWASP Top 10 for Agentic Applications (ASI07) [11] and the NIST CAISI Request for Information [12]: empirical evidence that a structural, non-prompt mechanism can gate inter-agent state writes.

7.1. What Cryptography Buys: Limits, Deployment, and Cross-Domain Reach

CapChain demonstrates that the inter-agent privacy boundary in production Lang-Graph applications can be enforced at the state-transition layer rather than at the prompt layer. The three primitives together address the attacker goals of Section 3.2 without modi-

fying agent system prompts or retraining any model. However, structural enforcement is not a substitute for behavioral defenses. CapChain does not prevent an LLM from being instructed to disclose information it legitimately holds (model-level jailbreak, explicitly out of scope per Section 3.3). It restricts what the agent can cause to be observed by other agents, not what the agent itself can be made to emit. Practitioners deploying CapChain should pair it with the prompt-layer defenses surveyed in Section 2.3 (StruQ [20], Spotlighting [22]), treating CapChain as the floor and behavioral defenses as the ceiling.

The CapChain prototype is intentionally interface-compatible with LangGraph’s default reducer (Section 4.4). Migrating an existing production graph requires (i) declaring the per-field sensitivity attributes once at schema definition, (ii) replacing the default reducer with CAR, and (iii) running an issuer process holding k_{issuer} . No agent prompts are touched. The prototype runs against the standard SQLite, Postgres, and Redis LangGraph checkpoints without modification.

CAR adds three constant-time gate evaluations per state write (HMAC-SHA256 verification, Merkle-path verification at $O(\log N)$ over the provenance log, and a deterministic attribute–purpose set intersection), which a CPU microbenchmark (median of 200,000 iterations per gate on a commodity arm64 CPU) measures at a combined median of 6.8 microseconds (3.5, 3.3 at $N = 1024$, and 0.04 microseconds, respectively). Implemented as a LangGraph custom reducer, the attribute–purpose gate adds 0.16 microseconds at the reducer call, and a single-merge graph superstep completes in 165 microseconds, both sub-millisecond. The dominant overhead is the database round-trip for the provenance write, which is co-located with the existing checkpoint write and therefore does not add a separate transaction. A full LangGraph-native multi-agent deployment and its end-to-end wall-clock profile are deferred to future work.

The capability schema generalizes across regulatory regimes because **attrs** in Section 4.1 is a free attribute set, not a domain-specific enum. The same three-gate CAR mechanism shown in the walk-through in Figure 3 evaluates the same intersection test on each regulatory domain; only the attribute alphabet changes. The encoding template is mechanical: for any private field with a regulated content class (PHI/PCI/SOX/...), attach the corresponding regulatory tag to **attrs**; nodes whose static purpose declaration is incompatible with that tag have the write rejected by the attribute–purpose gate. This cross-domain property (one schema, multiple regulatory regimes) follows from capabilities being symbolic rather than domain-specific; a full empirical cross-domain study (worked HIPAA and PCI-DSS instantiations) is left to future work.

7.2. Closest Neighbors: Side-by-Side Comparison

Table 4 compares CapChain against the closest 2025–2026 defenses in the agent-security literature. The comparison is intentionally inclusive of works that either (a) use a similar capability-based vocabulary (CaMeL, ACE), (b) are cryptographic in mechanism (AgentCrypt, SAGA, and the offline-verifiable agent-to-agent messaging of Alshammari et al.), (c) track information flow at the planner (Fides), (d) target the same contextual-integrity outcome (Wang’s PrivacyChecker), or (e) enforce field-level privacy through a logical reference monitor (Maris [40]). CapChain’s position in this peer set is defined by where and how it enforces, not by any single primitive: a cryptographically bound, tamper-evidently-logged, per-field read-access capability at the LangGraph state-merge reducer. Each neighbor shares one axis but differs on enforcement point or trust assumption; we therefore frame CapChain as a recombination of established primitives at a previously cryptographically ungated enforcement point rather than as a new cryptographic construction.

Table 4. CapChain versus the closest 2025–2026 defenses. “Layer” is the enforcement point, “Enforcement” the dominant mechanism, “Threat/outcome” the claimed adversary capability or measurement target, “Artifact” the reference implementation, and “Key limitation” the property that separates each defense from a field-level state-merge gate. PrivacyChecker’s 75% figure is on a different surface (CI queries, not MAGPIE inter-agent dialogue) and is not directly comparable to our LR_S; it is complementary (inference-time layer), not competing.

System	Layer	Enforcement	Threat/Outcome	Artifact	Key Limitation
CaMeL [28]	planner-executor	Python interpreter + dual-LLM	prompt injection (single orchestrator)	AgentDojo runner	operates at the planner–executor boundary, not the field-level state merge
ACE [27]	planner/orchestrator	capability-style policy on plan steps	compromised single agent	policy engine	gates plan steps above shared state, not the field-level merge
Fides [43]	planner/executor	dynamic information-flow (taint) tracking	prompt-injection exfiltration	MSR prototype	tracks taint on planner values, not a capability bound to persisted state fields
AgentCrypt [32]	payload	IBE + FHE encryption wrapper	confidential payload	LangGraph + ADK lib	encrypts payloads, not access rights or lineage on shared state
SAGA [30]	inter-agent channel A→B	ECDSA + HKDF/DH session-key exchange	channel impersonation; not content gating	channel handshake library	authenticates whether A talks to B, not which field is passed
Alshammari [33]	message/channel	Merkle + signed-tree-head receipts, offline-verifiable	message authenticity/non-repudiation	messaging library	attests message exchange, not field-level read access on shared state
PrivacyChecker [23]	inference-time CI module	modular CI-rule classifier	contextual privacy on CI-query benchmarks (~75% reduction)	model-agnostic library	inference-time CI rules, complementary to a state-layer gate rather than a substitute for one
Maris [40]	message monitor (Lang-Graph/AG2)	MFOTL reference monitor + typed predicates	compromised agent; inter-agent privacy	MSR/UIUC prototype	logical policy, not an unforgeable capability; no tamper-evident provenance; trusts the monitor/runtime
Conseca [44]	tool/executor boundary	LLM-generated just-in-time per-purpose policy	prompt injection; contextual misuse	PoC (computer-use agent)	policy is LLM-generated at the tool boundary, not a cryptographic capability at the reducer
CapChain (this work)	LangGraph state-merge reducer	HMAC CapTok + Merkle ProvChain	honest-but-curious (empirical); compromised-single and colluding-minority (design); MAGPIE inter-agent leak	CAR + issuer process + ProvChain	empirical figure is a redaction-decision rate; live downstream prevention is design-level

Several 2025–2026 works sharpen this boundary. Fides [43] tracks dynamic information flow over a planner’s execution values to stop prompt-injection exfiltration; CapChain instead binds a cryptographic capability to persisted state fields at the merge step, so its guarantee is about which agent may read a field rather than taint on planner values. The authenticated, offline-verifiable agent-to-agent messaging scheme of Alshammari et al. [33] provides Merkle and signed-tree-head receipts for messages exchanged between agents; this is the closest neighbor to ProvChain, but it attests message exchange rather than the field-level read access that CAR enforces on shared state. Maris [40] is the closest in setting: it enforces field-level privacy policies in LangGraph through a formally verified MFOTL reference monitor that the runtime is trusted to execute, whereas CapChain binds an unforgeable HMAC capability and tamper-evident provenance to each field, adding non-repudiation and forgery- and replay-resistance under a compromised- or colluding-peer adversary that a trusted logical monitor does not provide. Conseca [44] generates per-purpose policies just-in-time at the tool boundary rather than binding a cryptographic capability at the reducer. Several of these neighbors report strong end-to-end mitigation against prompt injection or single-orchestrator adversaries (Fides completes AgentDojo tasks with information-flow security guarantees against prompt injection); CapChain instead measures the honest-but-curious contextual-privacy regime on MAGPIE, where the disclosing peer is authorized, and attack-success-rate is not the operative metric, so a like-for-like efficacy comparison is left to the live-runtime work of Sections 6.5 and 7.4.

7.3. Limitations, Scope, and Threats to Validity

We name limitations explicitly so future work can build on a stable baseline.

The evaluation covers 25 scenarios from the MAGPIE stratified high-stakes pool on three open-source MoE LLMs (Section 5.1); this is a deliberately bounded slice of the benchmark, so the scope of generalization from these results is correspondingly limited and external validity is bounded, and the Salient firing is concentrated rather than uniform across the pool: many conditions never fire and the aggregate is driven by a high-stakes subset, so the reported rate should not be read as a uniform per-deployment rate. Frontier closed-source APIs (Opus 4.7, GPT-5.5, Gemini 3.5 Pro) are out of scope to keep the empirical cost defensible and to extend rather than duplicate MAGPIE's frontier-API measurement. All three suite models are Q4_K_M-quantized for VRAM compatibility; safety-tuning behavior is known to degrade under heavy quantization, so the absolute LR_s magnitudes reported here may be higher than the same models would exhibit in FP16 deployments.

A second limit concerns the issuer key as the root of trust. CapChain's cryptographic guarantees assume the HMAC issuer process and its key k_{issuer} are defended at the operational level (HSM-backed key storage, threshold cryptography for issuance, or equivalent). Compromise of k_{issuer} (whether by operator coercion, key exfiltration through a side-channel, or supply-chain attack on the issuer binary) is equivalent to losing the root of trust and invalidates all derived capabilities issued under that key. Such defenses are orthogonal to CapChain's contribution and are out of scope; we assume the operator follows standard key-management practice (rotation, hardware-backed storage, separation of duties).

Three suite-level caveats apply to the reported per-model numbers:

1. Per-model differential is not attributed. The per-model redact-event counts are reported for transparency only. The suite's three models differ simultaneously in vendor, training corpus, parameter scale, and response style; isolating the cause of the per-model gap would require controlling for response length (verbosity) and the detector's lexical-surface overlap with each scenario's private-field keywords, which we leave to follow-up work, so the counts should not be read as a substrate-level vulnerability rating.
2. Task-success rate is heterogeneous across the suite. TSR (consensus within $\text{max_rounds}=10$) varies across models; per-model rates and the TSR-conditional robustness check are in Section 6.3; TSR=0 conditions still emit private-field utterances and enter the leak-rate denominator.
3. Multi-round emergence ceiling. The MAGPIE-reported round-2/3 leak-emergence pattern was observed on frontier closed-source models. Whether MoE Q4_K_M models compress leak emergence within the same window is itself a sensitivity we have not validated. The high $\text{proposal:accept_proposal}$ ratio under GLM-4.7-flash suggests the round-10 cap may be hitting a behavioral rather than a leak ceiling for that model.

The Salient-augmented attribute-purpose gate detector (Section 5.3) combines StanzaNER pre-boost with per-scenario TF-IDF top- k keyword pulls derived programmatically from each MAGPIE scenario's `private_preferences` block, not from human annotation. Its false-positive rate against a human-rated reference remains unmeasured (we report a session-level LLM-judge proxy estimate of $\leq 43.81\%$ in Section 7.5). Recall against paraphrased restatements depends on whether the paraphrase reuses a Salient token; restatements that share no Salient token are a known recall limit of the deterministic approach, which we treat as the reproducible headline. The composite ranking $S_1 \cdot S_2 \cdot N$ used in Section 5.2 additionally multiplies by agent count N , biasing the selected pool toward $N \geq 6$ scenarios where the per-condition Salient leak rate is structurally lower (scenarios with fewer agents show a structurally higher per-condition rate; the agent-count break-

down is examined in follow-up work). Our reported aggregate $LR_S = 4.67\%$ is therefore best read as a lower bound on the rate at which CAR would gate, in deployments with smaller, specialized agent pools. Five second-order limitations remain: collusive-minority detection-vs-prevention (the authentication and provenance integrity gates prevent forgery; the attribute-purpose gate only detects and logs colluding-minority violations); framework portability beyond LangGraph; the synthetic-scenario and fixed-seed caveat; multi-round emergence with truncation handling; and CAR cryptographic overhead. Promoting the authentication and provenance gate reference implementation to a production LangGraph reducer integration with a compromised-agent and colluding-minority structured-tool-call adversarial benchmark is the natural next paper.

7.4. Scope: Redaction Decisions vs Downstream Efficacy

The reported leak rates measure detector-firing rate, not downstream defensive efficacy. The post hoc derivation methodology (Section 5.3) means LR_S measures how often CAR decides to redact a message under the attribute-purpose gate rule, not how much downstream agent-to-agent leakage CapChain prevents in a live deployment. The two are distinct.

A live-runtime study would substitute the redacted message into the receiving agent's context window; this could either eliminate the downstream leak signal (the intended outcome) or cause the receiving agent to re-derive the private field from surrounding context (a residual signal the post hoc derivation cannot quantify). The reported $LR_S = 4.67\%$ should therefore be read as a lower bound on the events CAR would gate, not as an upper bound on the residual leak rate after CapChain is deployed. Direct measurement requires such a live-runtime study and is deferred to follow-up work.

7.5. Construct Validity

For construct validity, the leak rate metric (Section 5.3) treats any appearance of a private field's content in an unauthorized agent's visible context as a leak, regardless of whether the disclosure is actionable; a paraphrased leak that is not directly usable for follow-on attack still counts. This is conservative; a more usage-aware metric would require human raters per scenario. Cross-checking the Salient-fired events against two independent LLM judges (Gemini-3.5-flash and Claude-Haiku-4.5, 331 field-level events; an event counts as a confirmed leak if either judge marks it leaked, an automated proxy with a human-rated per-message audit left to future work) estimates precision at no less than 56.19% (186 of 331 confirmed as genuine leaks). The complementary upper-bound false-positive rate of 43.81% is the expected cost of a recall-first deterministic detector: NER- and keyword-based PII and leak detectors deliberately favor recall over precision, since for leak detection a missed disclosure is worse than an over-redaction [45,46]; such over-redactions are typically entity-name mentions that do not themselves disclose private content, as the proper-noun fire in Figure 5 illustrates. It is the same recall-oriented stance behind reporting LR_S as a lower bound on gated events (Section 7.4) rather than a downstream-efficacy figure.

8. Conclusions

We presented CapChain, a defense-in-depth architecture that addresses the inter-agent privacy gap in multi-agent LLM systems built on LangGraph and equivalent graph-structured orchestrators. Three contributions together address a problem now elevated by multiple independent authorities (OWASP Top 10 for Agentic Applications 2026 [11], the NIST CAISI Request for Information [12], and the 2025 MIT AI Agent Index [9]) and grounded in a real exploited vulnerability (CVE-2025-67644 at CVSS 7.8) [10]. Empirically (Section 6), we report an empirical measurement of inter-agent contextual privacy on three

open-source mixture-of-experts LLMs spanning three vendors (Gemma 4-26B, GLM-4.7-flash, Qwen 3.6-35B). The evaluation is scored using MAGPIE’s contextual-integrity probe, with CAR enforcing LangGraph state-merge reducer semantics on the recorded traces. This measurement is a redaction-decision rate (how often CAR’s attribute–purpose gate fires on inter-agent messages) and should be read as a lower bound on gated events, not a direct measure of leakage prevented in a live deployment (Section 7.4). This extends the frontier-API measurement of MAGPIE [7] into the open-source MoE substrate. We do not claim CapChain has been validated as an operational security solution; demonstrating downstream leakage prevention in a live, end-to-end production deployment remains future work (Sections 6.5 and 7.4). Analytically (Section 2.3), we characterized why policy-based defenses (SAGA [30], ACE [27], Progent [25], AgentCrypt [32]) cannot fully close the inter-agent channel: each operates at a layer either above the shared state (prompt, planner) or below it (channel, payload), leaving the merge boundary unmonitored. Constructively (Section 4), we contributed (i) the design of an HMAC capability token (CapTok), a Merkle-anchored provenance log (ProvChain), and a three-gate capability-aware reducer (CAR) with explicit mapping to OWASP ASI06 (Memory and Context Poisoning) and ASI07 (Insecure Inter-Agent Communication); and (ii) a reference implementation of CAR’s attribute–purpose gate, exercised inside the experimental harness at the inter-agent message boundary.

CapChain demonstrates that cryptographic primitives developed for process-level capability systems (Lampson’s protection matrix [34], KeyKOS [35], and Miller’s object-capability discipline [37]) transfer to a setting where the bearer is an LLM rather than a process, by relocating enforcement to the state-transition layer that graph-structured orchestrators expose. As multi-agent deployments move from prototype to regulated production, we expect this structural layer to become the natural locus for inter-agent privacy, integrity, and audit guarantees that prompt-level defenses cannot provide.

Author Contributions: Conceptualization, T.W.C. and J.-S.L.; methodology, T.W.C.; software, T.W.C.; validation, T.W.C., J.-S.L. and W.-B.H.; formal analysis, T.W.C.; investigation, T.W.C.; resources, J.-S.L.; data curation, T.W.C.; writing—original draft preparation, T.W.C.; writing—review and editing, T.W.C., J.-S.L. and W.-B.H.; visualization, T.W.C.; supervision, J.-S.L.; project administration, J.-S.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Any additional information regarding the data and code used in this research can be obtained from the corresponding author upon reasonable request.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ASI	Agentic Security Issue (OWASP Agentic Top 10)
CAR	Capability-Aware Reducer
CapTok	Capability Token
CI	Contextual Integrity
CVE	Common Vulnerabilities and Exposures
EUF-CMA	Existential Unforgeability under Chosen-Message Attack
HMAC	Hash-based Message Authentication Code
IPI	Indirect Prompt Injection

LLM	Large Language Model
LR	Leak Rate
MoE	Mixture of Experts
NER	Named Entity Recognition
NIST	National Institute of Standards and Technology
OWASP	Open Worldwide Application Security Project
PHI	Protected Health Information
ProvChain	Provenance Chain
TSR	Task Success Rate
V/S	Verbatim/Salient Detector tiers

References

1. Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; Cao, Y. ReAct: Synergizing Reasoning and Acting in Language Models. In Proceedings of the ICLR, Kigali, Rwanda, 1–5 May 2023. Available online: https://openreview.net/forum?id=WE_vluYUL-X (accessed on 24 May 2026).
2. Schick, T.; Dwivedi-Yu, J.; Dessì, R.; Raileanu, R.; Lomeli, M.; Hambro, E.; Zettlemoyer, L.; Cancedda, N.; Scialom, T. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*; Curran Associates, Inc.: Red Hook, NY, USA, 2023; Volume 36, pp. 68539–68551. Available online: https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html (accessed on 24 May 2026).
3. Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Li, B.; Zhu, E.; Jiang, L.; Zhang, X.; Zhang, S.; Liu, J.; et al. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversations. In Proceedings of the Conference on Language Modeling (COLM), Philadelphia, PA, USA, 7–9 October 2024. Available online: <https://openreview.net/forum?id=BAakY1hNKS> (accessed on 24 May 2026).
4. LangChain. LangGraph Documentation, 2024. Available online: <https://docs.langchain.com/oss/python/langgraph/overview> (accessed on 24 May 2026).
5. Greshake, K.; Abdelnabi, S.; Mishra, S.; Endres, C.; Holz, T.; Fritz, M. Not What You’ve Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. In Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec), Copenhagen, Denmark, 30 November 2023; pp. 79–90. <https://doi.org/10.1145/3605764.3623985>.
6. Liu, X.; Yu, Z.; Zhang, Y.; Zhang, N.; Xiao, C. Automatic and Universal Prompt Injection Attacks against Large Language Models. *arXiv* **2024**, arXiv:2403.04957. <http://arxiv.org/abs/2403.04957>.
7. Juneja, G.; Pasupulati, J.N.S.; Albalak, A.; Hua, W.; Wang, W.Y. MAGPIE: A Benchmark for Multi-Agent Contextual Privacy Evaluation. In Proceedings of the NeurIPS 2025 Workshop on Socially Responsible and Trustworthy Foundation Models (ResponsibleFM), San Diego, CA, USA, 6–7 December 2025. Available online: <https://openreview.net/forum?id=vZgdho8Vx0> (accessed on 24 May 2026).
8. El Yagoubi, F.; Badu-Marfo, G.; Al Mallah, R. AgentLeak: A Benchmark for Internal-Channel Privacy Leakage in Multi-Agent LLM Systems. *IEEE Access* **2026**, *14*, 94960–94978. <https://doi.org/10.1109/ACCESS.2026.3704541>.
9. Staufer, L.; Feng, K.; Wei, K.; Bailey, L.; Duan, Y.; Yang, M.; Ozisik, A.P.; Casper, S.; Kolt, N. The 2025 AI Agent Index: Documenting Technical and Safety Features of Deployed Agentic AI Systems. In Proceedings of the 2026 ACM Conference on Fairness, Accountability, and Transparency (FAccT), Montreal, QC, Canada, 25–28 June 2026. <https://doi.org/10.1145/3805689.3806728>.
10. Tenable. CVE-2025-67644, 2026. SQL Injection Vulnerability in Langgraph-Checkpoint-Sqlite; CVSS 7.8. Available online: <https://www.tenable.com/cve/CVE-2025-67644> (accessed on 24 May 2026).
11. OWASP GenAI Security Project. OWASP Top 10 for Agentic Applications for 2026, 2025. Available online: <https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/> (accessed on 24 May 2026).
12. U.S. Department of Commerce. Request for Information Regarding Security Considerations for Artificial Intelligence Agents, 2026. Available online: <https://www.federalregister.gov/documents/2026/01/08/2026-00206/request-for-information-regarding-security-considerations-for-artificial-intelligence-agents> (accessed on 24 May 2026).
13. Raza, S.; Sapkota, R.; Karkee, M.; Emmanouilidis, C. TRiSM for Agentic AI: A Review of Trust, Risk, and Security Management in LLM-based Agentic Multi-Agent Systems. *AI Open* **2026**, *7*, 71–95. <https://doi.org/10.1016/j.aiopen.2026.02.006>.
14. Bellare, M.; Canetti, R.; Krawczyk, H. Keying Hash Functions for Message Authentication. In *Proceedings of the Advances in Cryptology—CRYPTO ’96*; Springer: Berlin/Heidelberg, Germany, 1996; Volume 1109, pp. 1–15. https://doi.org/10.1007/3-540-68697-5_1.
15. Krawczyk, H.; Bellare, M.; Canetti, R. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, Internet Engineering Task Force (IETF), February 1997. Available online: <https://www.rfc-editor.org/info/rfc2104/> (accessed on 24 May 2026).

16. Merkle, R.C. *Secrecy, Authentication, and Public Key Systems*. Ph.D. Thesis, Stanford University, Stanford, CA, USA, 1979.
17. Kong, D.; Lin, S.; Xu, Z.; Wang, Z.; Li, M.; Li, Y.; Zhang, Y.; Peng, H.; Chen, X.; Sha, Z.; et al. A Survey of LLM-Driven AI Agent Communication: Protocols, Security Risks, and Defense Countermeasures. *arXiv* **2025**, arXiv:2506.19676.
18. Wang, H.; Mang, Q.; Cheung, A.; Sen, K.; Song, D. How We Broke Top AI Agent Benchmarks: And What Comes Next, 2026. Available online: <https://rdi.berkeley.edu/blog/trustworthy-benchmarks-cont/> (accessed on 24 May 2026).
19. Mireshghallah, N.; Kim, H.; Zhou, X.; Tsvetkov, Y.; Sap, M.; Shokri, R.; Choi, Y. Can LLMs Keep a Secret? Testing Privacy Implications of Language Models via Contextual Integrity Theory. In Proceedings of the ICLR, Vienna, Austria, 7–11 May 2024. Available online: <https://openreview.net/forum?id=gmg7t8b4s0> (accessed on 24 May 2026).
20. Chen, S.; Piet, J.; Sitawarin, C.; Wagner, D. StruQ: Defending Against Prompt Injection with Structured Queries. In Proceedings of the USENIX Security, Seattle, WA, USA, 13–15 August 2025. Available online: <https://www.usenix.org/conference/usenixsecurity25/presentation/chen-sizhe> (accessed on 24 May 2026).
21. Bagdasarian, E.; Yi, R.; Ghalebikesabi, S.; Kairouz, P.; Gruteser, M.; Oh, S.; Balle, B.; Ramage, D. AirGapAgent: Protecting Privacy-Conscious Conversational Agents. In Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24), Salt Lake City, UT, USA, 14–18 October 2024; pp. 3868–3882. <https://doi.org/10.1145/3658644.3690350>.
22. Hines, K.; Lopez, G.; Hall, M.; Zarfati, F.; Zunger, Y.; Kiciman, E. Defending Against Indirect Prompt Injection Attacks with Spotlighting. In Proceedings of the Conference on Applied Machine Learning in Information Security (CAMLIS), Arlington, VA, USA, 24–25 October 2024; pp. 48–62. Available online: <https://ceur-ws.org/Vol-3920/paper03.pdf> (accessed on 24 May 2026).
23. Wang, S.; Yu, F.; Liu, X.; Qin, X.; Zhang, J.; Lin, Q.; Zhang, D.; Rajmohan, S. Privacy in Action: Towards Realistic Privacy Mitigation and Evaluation for LLM-Powered Agents. In Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2025, Suzhou, China, 4–9 November 2025; pp. 17055–17074. <https://doi.org/10.18653/v1/2025.findings-emnlp.925>.
24. Shi, T.; Zhu, K.; Wang, Z.; Jia, Y.; Cai, W.; Liang, W.; Wang, H.; Alzahrani, H.; Lu, J.; Kawaguchi, K.; et al. PromptArmor: Simple yet Effective Prompt Injection Defenses. *arXiv* **2025**, arXiv:2507.15219.
25. Shi, T.; He, J.; Wang, Z.; Li, H.; Wu, L.; Guo, W.; Song, D. Progent: Securing AI Agents with Privilege Control. *arXiv* **2025**, arXiv:2504.11703.
26. Bhagwatkar, R.; Kasa, K.; Puri, A.; Huang, G.; Rish, I.; Taylor, G.W.; Dvijotham, K.D.; Lacoste, A. Indirect Prompt Injections: Are Firewalls All You Need, or Stronger Benchmarks? In Proceedings of the NeurIPS 2025 Workshop on Preventing Unauthorized Knowledge Use from LLMs (Lock-LLM), San Diego, CA, USA, 6–7 December 2025. Available online: <https://neurips.cc/virtual/2025/128597> (accessed on 24 May 2026).
27. Li, E.; Mallick, T.; Rose, E.; Robertson, W.; Oprea, A.; Nita-Rotaru, C. ACE: A Security Architecture for LLM-Integrated App Systems. In Proceedings of the NDSS, San Diego, CA, USA, 23–27 February 2026. Available online: <https://www.ndss-symposium.org/ndss-paper/ace-a-security-architecture-for-llm-integrated-app-systems/> (accessed on 24 May 2026).
28. Debenedetti, E.; Shumailov, I.; Fan, T.; Hayes, J.; Carlini, N.; Fabian, D.; Kern, C.; Shi, C.; Terzis, A.; Tramèr, F. Defeating Prompt Injections by Design. In Proceedings of the IEEE Conference on Secure and Trustworthy Machine Learning (SaTML), Munich, Germany, 23–25 March 2026. Available online: <https://floriantramer.com/publications/camel25/> (accessed on 24 May 2026).
29. Wu, Y.; Roesner, F.; Kohno, T.; Zhang, N.; Iqbal, U. IsolateGPT: An Execution Isolation Architecture for LLM-Based Agentic Systems. In Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 24–28 February 2025. Available online: <https://www.ndss-symposium.org/ndss-paper/isolategpt-an-execution-isolation-architecture-for-llm-based-agentic-systems/> (accessed on 24 May 2026).
30. Syros, G.; Suri, A.; Ginesin, J.; Nita-Rotaru, C.; Oprea, A. SAGA: A Security Architecture for Governing AI Agentic Systems. In Proceedings of the NDSS, San Diego, CA, USA, 23–27 February 2026. Available online: <https://www.ndss-symposium.org/ndss-paper/saga-a-security-architecture-for-governing-ai-agentic-systems/> (accessed on 24 May 2026).
31. Zhang, X.; Xu, H.; Ba, Z.; Wang, Z.; Hong, Y.; Liu, J.; Qin, Z.; Ren, K. PrivacyAsst: Safeguarding User Privacy in Tool-Using Large Language Model Agents. *IEEE Trans. Dependable Secur. Comput.* **2024**, *21*, 5242–5258. <https://doi.org/10.1109/TDSC.2024.3372777>.
32. Karthikeyan, H.; Guo, Y.; de Castro, L.; Polychroniadou, A.; Ardon, L.; Sehwal, U.M.; Ganesh, S.; Veloso, M. AgentCrypt: Advancing Privacy and (Secure) Computation in AI Agent Collaboration. In Proceedings of the NeurIPS 2025 Workshop on Regulatable ML (RegML), San Diego, CA, USA, 6–7 December 2025. Available online: <https://openreview.net/forum?id=vheyiwBRfD> (accessed on 24 May 2026).
33. Alshammari, A.; Assiri, S.; Bahşi, H. Authenticated and Offline-Verifiable Agent-to-Agent Messaging for LLM Agents. In Proceedings of the 2026 IEEE 16th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 5–7 January 2026. <https://doi.org/10.1109/ccwc67433.2026.11393740>.
34. Lampson, B.W. Protection. *ACM SIGOPS Oper. Syst. Rev.* **1974**, *8*, 18–24. <https://doi.org/10.1145/775265.775268>.
35. Hardy, N. KeyKOS Architecture. *ACM SIGOPS Oper. Syst. Rev.* **1985**, *19*, 8–25. <https://doi.org/10.1145/858336.858337>.
36. Klein, G.; Andronick, J.; Elphinstone, K.; Murray, T.; Sewell, T.; Kolanski, R.; Heiser, G. Comprehensive Formal Verification of an OS Microkernel. *ACM Trans. Comput. Syst.* **2014**, *32*, 1–70. <https://doi.org/10.1145/2560537>.

37. Miller, M.S. Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control. Ph.D. Thesis, Johns Hopkins University, Baltimore, MD, USA, 2006. Available online: <https://jscholarship.library.jhu.edu/items/b8376e55-208e-4fcc-81a2-ad5a7de0acbb> (accessed on 24 May 2026).
38. Bellare, M. New Proofs for NMAC and HMAC: Security without Collision-Resistance. In *Proceedings of the Advances in Cryptology—CRYPTO 2006*; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4117, pp. 602–619. https://doi.org/10.1007/11818175_36.
39. Laurie, B.; Messeri, E.; Stradling, R. Certificate Transparency Version 2.0. RFC 9162, Internet Engineering Task Force (IETF), December 2021. Available online: <https://www.rfc-editor.org/info/rfc9162/> (accessed on 24 May 2026).
40. Cui, J.; Li, Z.; Wang, C.; Xing, L.; Liao, X. Maris: A Formally Verifiable Privacy Policy Enforcement Paradigm for Multi-Agent Collaboration Systems. *arXiv* **2025**, arXiv:2505.04799.
41. Chen, Z.; Xiang, Z.; Xiao, C.; Song, D.; Li, B. AgentPoison: Red-teaming LLM Agents via Poisoning Memory or Knowledge Bases. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*; Curran Associates, Inc.: Red Hook, NY, USA, 2024; Volume 37, pp. 130185–130213. Available online: https://proceedings.neurips.cc/paper_files/paper/2024/hash/eb113910e9c3f6242541c1652e30dfd6-Abstract-Conference.html (accessed on 24 May 2026).
42. Dong, S.; Xu, S.; He, P.; Li, Y.; Tang, J.; Liu, T.; Liu, H.; Xiang, Z. Memory Injection Attacks on LLM Agents via Query-Only Interaction. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*; Curran Associates, Inc.: Red Hook, NY, USA, 2025; Volume 38, pp. 46697–46731. Available online: https://proceedings.neurips.cc/paper_files/paper/2025/hash/42a97bbd9844d2bf68596730af80bcd6-Abstract-Conference.html (accessed on 24 May 2026).
43. Costa, M.; Köpf, B.; Kolluri, A.; Pavard, A.; Russinovich, M.; Salem, A.; Tople, S.; Wutschitz, L.; Zanella-Béguelin, S. Securing AI Agents with Information-Flow Control. *arXiv* **2025**, arXiv:2505.23643.
44. Tsai, L.; Bagdasarian, E. Contextual Agent Security: A Policy for Every Purpose. In *Proceedings of the 2025 Workshop on Hot Topics in Operating Systems (HotOS '25)*, ACM, Banff, AB, Canada, 14–16 May 2025; pp. 8–17. <https://doi.org/10.1145/3713082.3730378>.
45. Li, R.; Allal, L.B.; Zi, Y.; Muennighoff, N.; Kocetkov, D.; Mou, C.; Marone, M.; Akiki, C.; Li, J.; Chim, J.; et al. StarCoder: May the Source Be with You! *arXiv* **2023**, arXiv:2305.06161. Available online: <https://openreview.net/forum?id=KoFOg41haE> (accessed on 24 May 2026).
46. Kotevski, D.P.; Smee, R.I.; Field, M.; Nemes, Y.N.; Broadley, K.; Vajdic, C.M. Evaluation of an Automated Presidio Anonymisation Model for Unstructured Radiation Oncology Electronic Medical Records in an Australian Setting. *Int. J. Med. Inform.* **2022**, *168*, 104880. <https://doi.org/10.1016/j.ijmedinf.2022.104880>.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.