Consolidation Leakage and Lineage-Closure Projection in Shared Multi-Agent Memory

Marvin Danig
Achiral Research
marvin@achiral.ai

Architecture and analysis preprint
November 2026

Abstract

Shared memory for teams of people and agents does not only store records. It derives new ones: summaries consolidated from many conversations, semantic memories generalized from episodes, and procedures learned from repeated action traces. Access control in these systems usually checks the record being read. A derived memory, however, can depend on sources the reader is not allowed to see, so it can pass restricted information to that reader even though no restricted record is ever read. We call this *consolidation leakage*. We define it against a counterfactual memory built without the restricted sources, show why record-level filtering can miss it, and describe *lineage-closure projection*, a filed design that builds each requester's view of memory by checking a memory's ancestry as well as the memory itself. We analyze what projection prevents and what it still discloses, and we apply the same evidence lens to promoting learned procedures into actions an agent may take unattended. The paper reports no empirical measurements; its only numbers are exact binomial calculations. The central claim is falsified if record-level filtering shows no leakage above the counterfactual baseline under realistic consolidation, and narrowed if per-scope consolidation matches lineage-closure projection on leakage and utility at equal cost. If the claim holds, access control for shared AI memory has to follow lineage, not only records.

Copyright 2026 Achiral Research. Patent rights reserved. No patent license is granted by publication of this article.

1 Introduction

Memory for language model systems is becoming shared. Assistants keep memory across sessions, managed agents keep memory across tasks, and team products give many people and agents one memory to read from and write to [1, 2, 3, 4]. Shared memory is only useful because it does work on what it stores. It consolidates many episodes into one semantic memory, reflects on interactions to produce lessons [5], and learns reusable procedures from what worked [8, 9]. Each of these operations reads several memories and writes a new one.

Access control has not kept up with that work. A typical shared memory checks whether a requester may read a record, using the record's own scope label: its tenant, project, role, or sensitivity class. The label of a derived memory is usually the label of the place it was written to, not the intersection of the labels of everything it was derived from. When a consolidation job summarizes a support team's tickets, the summary may draw on internal notes that only some readers may see; when it is filed under the support project, everyone with project access reads it.

Consider a concrete case, adapted from the worked example in the filing this paper accompanies. A support team resolves six synchronization tickets. Internal notes on customer accounts, which contractors may not read, explain that five failures were fixed by refreshing an expired authorization token and replaying the sync, and that one failed because the account was suspended. The memory system generates a procedure from the tickets' action-result traces: *if the account is active and the token has expired, refresh the token and replay.* A contractor who may read procedures but not customer notes now holds knowledge whose only source is the notes. The procedure may expose a condition learned from restricted notes. Whether it reveals a protected fact depends on its wording and on what

the reader already knows: the generic condition *if the account is active* does not, by itself, show that any particular account was suspended.

No restricted record was read. The contractor read one permitted record. The mistake was made when the procedure was written, by placing information in a record whose audience is broader than that information's allowed audience; disclosure occurs when the information reaches a reader.

This paper names that failure, defines it so that it can be measured, and describes one design that addresses it. The central claim is narrow: in shared memory that derives new memories from old ones, access decisions must follow a memory's lineage, not only its own label. The claim is falsified if record-level filtering shows no measurable leakage above a counterfactual baseline under realistic consolidation, and narrowed if a simpler design, consolidating separately within each permission scope, matches lineage-aware projection on leakage and on the utility of permitted knowledge at equal cost.

Contributions. We (i) define consolidation leakage and lineage disclosure against a counterfactual memory (Section 2); (ii) argue that record-level filtering can miss consolidation leakage (Section 3); (iii) describe lineage-closure projection and the governed memory evolution it depends on, as specified in a filed design (Sections 4 and 5); (iv) analyze what projection prevents and what it still discloses (Section 6); (v) compute exact pass rates for evidence gates that promote learned procedures into actions (Section 7); and (vi) specify an evaluation that would test each claim (Section 8). This architecture-and-analysis preprint reports no empirical measurements, and it is not a description of a deployed system.

2 Setting and Definitions

A shared memory holds a set V of memory entities written by people, agents, and tools within one tenant. Lineage edges $E \subseteq V \times V$ record derivation: $(a, m) \in E$ means that memory m was derived in part from memory a, for example by consolidation, summarization, reflection, or proceduralization. Lineage is acyclic, and we write $\mathrm{anc}(m)$ for the set of all memories from which m was transitively derived.

Each memory carries a scope label $\ell(m)$ drawn from the tenant's access policy. A requester r, a person or an agent, has a permission scope $P(r)$, and we write $\mathrm{vis}(m, r)$ when $\ell(m)$ falls within $P(r)$. A *projection* for r is the subset of memories, and of the edges among them, that the system exposes to r in response to a request.

Record-level filtering exposes m to r if and only if $\mathrm{vis}(m, r)$. It never consults $\mathrm{anc}(m)$.

Definition 1 (consolidation leakage). Let M be a memory built from an event history H, and let $R \subseteq H$ be the events whose resulting memories r may not see. Let $M^{\setminus R}$ be the counterfactual memory built by the same system from $H \setminus R$, as if the restricted events had never occurred. For a target fact f carried only by R, consolidation leakage for r is the increase in how well an adversary with access to r's projection of M can recover f, relative to the same adversary with access to r's projection of $M^{\setminus R}$. Leakage is zero exactly when r's projection reveals nothing about f beyond what the counterfactual reveals.

The counterfactual baseline matters. Without it, a measurement would count facts that the requester could have learned from permitted sources alone, and would overstate leakage.

Definition 2 (lineage disclosure). Lineage disclosure is the analogous increase in an adversary's ability to infer the *existence*, count, or kind of restricted ancestors of a visible memory, rather than their content. It is a membership-inference question [19] posed about memory ancestry.

The two definitions separate what a reader learns about restricted content from what a reader learns about restricted structure. Section 6 shows that a design can reduce one without reducing the other.

3 Why Record-Level Filtering Misses It

Consolidation is a function of its inputs. If a consolidated memory m has a restricted ancestor a, and the content of m depends on the content of a, then exposing m transmits information about a. Record-level filtering decides

Symbol	Meaning
V, E	Memory entities in one tenant, and lineage (derivation) edges among them
$\mathrm{anc}(m)$	All memories from which m was transitively derived
$\ell(m)$, $P(r)$	Scope label of memory m; permission scope of requester r
$\mathrm{vis}(m, r)$	Memory m's label falls within r's permission scope
H, R	Event history; the restricted events within it
M, $M^{\setminus R}$	Memory built from H; counterfactual memory built from $H \setminus R$
n_c, a_c	Independent executions of candidate procedure c; its observed success ratio
e_c, s_c, x_c, b_c	Efficacy score, safety score, exception ratio, and rollback indicator of c
$n_{\min}$, $a_{\min}$, $e_{\min}$, $s_{\min}$, $x_{\max}$	Policy thresholds on the gate terms
p	True success probability of a procedure

Table 1: Notation. Every symbol is also defined in the text at first use.

whether to expose m from $\ell(m)$ alone. Unless the system sets $\ell(m)$ to the most restrictive label among m's ancestors, the label of a derived memory is uninformative about what the memory carries, and record-level filtering has no input on which to detect the leak.

Setting $\ell(m)$ to the most restrictive ancestor label at write time is one remedy, and a reasonable baseline; it is the high-water-mark rule of lattice-based information flow [15]. It has two costs. A consolidated memory with one restricted ancestor becomes invisible to everyone outside that scope, including for the parts derived from permitted sources. And a write-time label becomes stale if it is not updated when source permissions change; propagating labels or recomputing authorization can address that, at additional cost.

This is an instance of the inference problem studied in database security, where permitted queries combine to reveal protected values [16, 17], and it is related to the extraction of training data from language models [18]. The difference here is that the derivation is performed continuously by the memory system itself, often by a language model, across many agents and permission scopes, and its outputs are written back as first-class memories that are themselves consolidated again.

4 Governed Memory Evolution

Lineage-aware access control requires lineage that is complete and trustworthy. The filed design provides it through governed memory evolution, summarized here only as far as projection depends on it.

Events from conversations, documents, tickets, and tool executions are first canonicalized into event envelopes that carry a tenant, an actor, source timestamps, an access scope, a sensitivity class, and provenance fields. Candidate memories are extracted from envelopes and assigned a state from a finite set: candidate, active episodic, consolidated semantic, procedural candidate, active procedural, superseded, and archived. A transition between states is allowed only when an explicit, versioned policy predicate holds, and a rejected transition is recorded rather than silently dropped.

Every accepted transition is written as an append-only lineage transaction that names the predecessor and successor, the transition type, the predicate values that were evaluated, the actor, the policy version, and a validity interval. Transactions are committed atomically with the successor memory and its lineage edges, and are idempotent under retry; this follows the established event-sourcing pattern [22] and records provenance in the spirit of the W3C provenance model [23]. When new evidence contradicts an existing memory above a threshold, the system creates a competing branch scoped to a context, such as a time window, project, customer segment, or policy regime, instead of overwriting the earlier memory, much as an assumption-based truth maintenance system keeps beliefs that hold in different contexts [13].

For this paper, the essential property is that $\mathrm{anc}(m)$ is recorded for every derived memory, cannot be rewritten after the fact, and survives contradiction and supersession.

5 Lineage-Closure Projection

Lineage-closure projection builds each requester's view by walking lineage from the memories a request selects. For every memory it reaches, it checks the memory's own scope; for every lineage edge, it applies an edge disclosure policy that takes one of four actions:

1. *include*: expose the edge and continue the walk to its other endpoint;
2. *tokenize*: replace a restricted endpoint with a redacted token that carries an opaque identifier, the edge type and direction, a coarse source class, and an integrity value, but no restricted content or identifiers;
3. *omit successor*: withhold a memory that depends on an ancestor the requester may not see;
4. *mark incomplete*: expose the memory with an explicit marker that part of its lineage is withheld.

The design does not infer that a requester may see an ancestor because the requester may see a descendant. Before a projection is returned, it is checked so that every exposed edge has two exposed endpoints or an authorized token, no token contains restricted content, and every exposed derived memory either carries the lineage the policy requires or is marked as incomplete. Unresolved contradictions that affect an exposed memory are flagged in the projection.

Each projection is keyed by a scope hash computed from the requester, tenant, roles, projects, sensitivity ceiling, and policy version, and is invalidated when any of these changes or when a branch in its lineage is resolved. The same lineage graph therefore yields different, internally consistent views for different requesters, and a change in permissions takes effect at the next request rather than at the next consolidation.

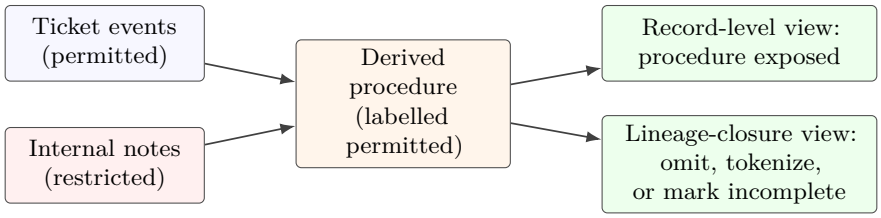


Figure 1: Consolidation leakage and the lineage-closure response. Boxes on the left are sources (blue: the requester may read them; red: the requester may not). The procedure in the middle is derived from both but labelled with the permitted scope. A record-level view (top right) exposes it because only its own label is checked; a lineage-closure view (bottom right) also checks its ancestry and applies one of the closure actions of Section 5. The figure is a schematic of the running example in Section 1, not a measurement.

6 What Projection Prevents and What It Discloses

The four closure actions trade leakage against utility differently, and none of them is safe in every case.

Omission is expected to reduce content leakage, at a cost. Withholding every derived memory with a restricted ancestor is expected to reduce consolidation leakage carried through lineage toward the counterfactual baseline (H2), but it also withholds what the memory learned from permitted sources. In the running example, the contractor loses the token-refresh procedure entirely, even though five of the six tickets are permitted.

Tokenization hides lineage, not content. A redacted token conceals what a restricted ancestor said, but the derived memory that the token accompanies is still exposed. If that memory's content depends on the restricted ancestor, tokenization reduces lineage detail without reducing consolidation leakage. In the running example, the exception for suspended accounts appears in the procedure itself, whatever the tokens conceal. Tokenization is safe for content only when the exposed memory is independent of its restricted ancestors, and the design leaves that judgment to policy rather than computing it.

Tokens disclose existence and count. Even when they carry no content, tokens reveal that a restricted ancestor exists, how many there are, and their coarse source class. In the running example, the contractor learns that the

procedure rests on several independent restricted sources. Because a token identifier is derived deterministically within a scope, the same restricted ancestor yields the same token across projections for that scope, which lets a reader link exposures. This is lineage disclosure in the sense of Definition 2, and it is a cost of the design, not a property it removes.

Release is a policy decision. Deciding that a derived memory may be released to a reader despite a restricted ancestor is a declassification decision. The design records and enforces that decision; it does not make it. How to decide when derived content is independent enough of restricted sources to release is, in our view, the central open question this paper leaves unanswered.

Per-scope consolidation is the natural baseline. A system can avoid consolidation leakage by never consolidating across scopes: it consolidates each permission scope's memories separately. This costs duplicate work and fragments knowledge that spans scopes, but it needs no lineage at all. Lineage-closure projection must show that it preserves more permitted knowledge than per-scope consolidation, at comparable leakage and cost, to justify its complexity.

7 When a Derived Procedure May Act

Proceduralization is the most consequential form of derivation. The output is not a fact to read but an action an agent may take, so its evidence matters as much as its access scope. The filed design gates promotion of a procedural candidate on a conjunction of indicator conditions:

$$G_c = I[n_c \geq n_{\min}] \cdot I[a_c \geq a_{\min}] \cdot I[e_c \geq e_{\min}] \cdot I[s_c \geq s_{\min}] \cdot I[x_c \leq x_{\max}] \cdot I[b_c = 1], \tag{1}$$

where n_c is the number of independent executions of candidate procedure c, a_c its observed success ratio, e_c an efficacy score, s_c a safety score, x_c an exception ratio, b_c an indicator that a rollback step exists, and each $I[\cdot]$ is one when its condition holds. Passing the gate makes a procedure eligible for approval, not automatically executable. The thresholds $n_{\min}$, $a_{\min}$, $e_{\min}$, $s_{\min}$ and $x_{\max}$ are policy settings. The filing's representative values include $n_{\min} = 5$ and $a_{\min} = 0.80$; these are illustrative, not calibrated.

The support and success terms are small-sample statistics, and at small n they cannot separate a reliable procedure from an unreliable one. Table 2 gives the exact probability that a procedure with true success probability p passes the support and success conditions alone, assuming n independent executions with identical success probability.

Gate	$p = 0.5$	$p = 0.7$	$p = 0.8$	$p = 0.9$	$p = 0.95$
Observed ratio ≥ 0.80, $n = 5$	0.188	0.528	0.737	0.919	0.977
Observed ratio ≥ 0.80, $n = 6$	0.109	0.420	0.655	0.886	0.967
Observed ratio ≥ 0.80, $n = 20$	0.006	0.238	0.630	0.957	0.997
Wilson lower bound ≥ 0.80, $n = 20$	0.000	0.001	0.012	0.122	0.358
Wilson lower bound ≥ 0.80, $n = 40$	0.000	0.001	0.028	0.423	0.862

Table 2: Probability that a procedure with true success probability p passes the support and success conditions of the promotion gate. Rows vary the gate: an observed success ratio of at least 0.80 at n executions, or a 95% Wilson score lower bound [24] of at least 0.80. Values are exact binomial probabilities under independent, identically distributed executions, so they carry no sampling uncertainty; they ignore the efficacy, safety, exception, and rollback conditions, which can only lower them. Takeaway: at the representative $n = 5$, a procedure that works half the time passes 18.8% of the time, while a lower-bound gate that blocks such procedures promotes a 90%-reliable procedure only 12% to 42% of the time at 20 to 40 executions.

Three points follow. First, in the filing's own worked example, five successes in six executions give a 95% Wilson interval of $[0.44, 0.97]$ for the true success rate: the observation is consistent with a procedure that fails more than half the time. Second, the filing lists confidence intervals, Bayesian posteriors, and sequential tests [25] as alternatives to the point ratio; Table 2 shows the price of using one, since a lower-bound gate at 0.80 needs 16 consecutive successes, or more than 60 executions at a 90% observed rate, before it can pass. Third, executions are

rarely independent. Tickets from one incident, or runs by one agent on similar inputs, share causes, so the effective sample size is smaller than n, and the pass rates in Table 2 are best cases.

The efficacy term e_c is the gate's most important and least specified component. A success ratio says how often outcomes were good when the procedure ran; it does not say whether the procedure caused them, which requires comparing against outcomes when it did not run. The July filing names an efficacy score without specifying its estimator. The November 9 filing describes matched-comparison and independent-group estimators, which this paper does not detail.

8 Evaluation and Falsification

We propose an evaluation on synthetic organizational memory, where restricted facts can be planted and the counterfactual memory $M^{\setminus R}$ can be built exactly. A generator produces a tenant with several permission scopes, an event history in which a known fraction of events is restricted, and target facts that appear only in restricted events. Each memory policy is run twice, on H and on $H \setminus R$, with the same consolidation operator, model, decoding settings, and seeds.

Policies to compare:

1. record-level filtering with derived memories labelled by where they are written;
2. record-level filtering with derived memories labelled by their most restrictive ancestor;
3. per-scope consolidation;
4. lineage-closure projection with tokenization;
5. lineage-closure projection with omission;
6. lineage-closure projection with release decided by an explicit independence policy.

Metrics: consolidation leakage per Definition 1, measured as the difference in an adversary's target-fact recovery rate between M and $M^{\setminus R}$; lineage disclosure per Definition 2, measured as accuracy at inferring the existence and count of restricted ancestors; utility, measured as recall of permitted facts and task success for permitted readers; and cost, measured as consolidation compute, projection latency, and storage. The adversary should include both scripted queries and a language model given the requester's projection; any automated judge of recovery should be checked against human labels on a random subset, and prompts and rubrics should be published verbatim. Results should be reported by consolidation depth and by restricted fraction.

For procedure promotion, simulated execution traces with known true success probabilities, including correlated traces, allow the false-promotion rate and the time to promotion to be measured for each gate in Table 2.

We would regard the following as confirming or refuting the claims:

- **H1.** Record-level filtering shows leakage above the counterfactual baseline, increasing with consolidation depth and restricted fraction. *Falsified* if leakage is indistinguishable from zero under realistic consolidation.
- **H2.** Lineage-closure projection with omission reduces leakage to the baseline, at a measurable utility cost. *Narrowed* if per-scope consolidation achieves equal leakage and utility at equal or lower cost.
- **H3.** Tokenization reduces lineage detail but not content leakage when derived content depends on restricted ancestors. *Refuted* if tokenized projections show baseline-level content leakage.
- **H4.** Redacted tokens enable existence and count inference above chance. *Refuted* if an adversary cannot beat chance at inferring restricted ancestors from tokens.

9 Limitations and Scope

No measurements. Every benefit and cost described here is a hypothesis until the evaluation in Section 8 is run. The only numbers in this paper are exact binomial calculations whose assumptions are stated with them.

Not a deployed system. The mechanisms described are a filed design. This paper does not describe the behavior of any production memory system, including Achiral's.

Leakage depends on the adversary. Definition 1 measures recovery by a particular adversary. A weak adversary understates leakage; for example, a scripted query set may miss a fact that a language model recovers by combining two exposed memories. Results should be read as lower bounds tied to the adversary used.

Consolidation is often nondeterministic. When a language model performs consolidation, M and $M^{\setminus R}$ differ by sampling noise as well as by the restricted events. A summary may omit a permitted fact in one run and include it in another, which inflates measured differences unless runs are repeated and the noise is reported.

Synthetic data may not transfer. Planted facts are cleaner than real organizational knowledge, where a restricted fact can often be partly inferred from permitted context. Real corpora may show less measurable leakage, or leakage concentrated in a few kinds of fact.

Independence assumptions. Table 2 assumes independent executions. Correlated executions, the common case, make every point-ratio gate less reliable than the table shows.

Disclosure boundary. The paper describes the filed design at the level of mechanisms. It withholds production thresholds, storage schemas, and implementation details on purpose, and it omits the detailed release-controller and efficacy-estimation recipes, some of which are disclosed in the November 9 provisional. Omission here is not a statement that the filed specification lacks them.

10 Related Work and Claim Boundary

Agent memory systems persist and organize memories across sessions: memory streams with reflection [5], operating-system-style memory tiers [4], temporal knowledge graphs with validity intervals [6], and self-organizing memory networks [7]. Skill libraries and workflow memories learn reusable procedures from agent experience [8, 9], as process mining learns models from event logs [10] and ACT-R compiles productions from practice [11]. Truth maintenance systems and belief revision study how beliefs change and coexist under new evidence [12, 13, 14]. Inference control in databases [16, 17], training-data extraction from language models [18], and membership inference [19] study how permitted outputs reveal protected inputs. The nearest work addresses governance of shared memory directly. Collaborative Memory [20] gives multi-user, multi-agent memory dynamic asymmetric permissions, immutable provenance, and retrospective permission checks with transformed views. Lineage-Aware Memory Governance [21] attaches full derivation graphs to cached analytical results and gates retrieval on authorization for the source columns, with guarantees conditional on its lineage assumptions.

This paper does not claim to invent consolidation, lineage, append-only logs, context-scoped belief branches, provenance records, procedure learning, or access control. It claims two things. First, a problem statement for recursively consolidated shared memory: consolidation leakage, defined against a counterfactual so that it can be measured, together with the distinction between hiding a memory's ancestry and sanitizing the content derived from it. Provenance- and lineage-based governance of shared memory is the subject of the two works above, not a contribution of this paper. Second, as described in a filed design by Marvin Danig at Achiral, the ordered combination that addresses it: governed state transitions with append-only lineage, context-scoped contradiction branches, evidence-gated proceduralization, and lineage-closure projections that check ancestry as well as records and stay consistent per requester. Whether that combination outperforms per-scope consolidation is the open empirical question.

11 Conclusion

In a memory that derives new memories from old ones, a summary is a read of its sources. Access control that checks only the summary lets restricted information through the consolidation step, without any restricted record being read. Lineage-closure projection makes ancestry part of every access decision; it is expected to reduce content leakage toward the counterfactual baseline (H2), at a utility cost, while its redaction tokens still disclose that restricted sources exist. The same evidence questions apply when consolidation produces procedures that act:

small-sample gates promote unreliable procedures at rates that are easy to compute and easy to overlook. Both problems are measurable, and the evaluation in Section 8 is designed to measure them.

Declarations

Author contributions. Marvin Danig conceived the design, developed the analysis, and wrote the manuscript.

Funding. This research received no external funding.

Competing interests. The author is the founder of Achiral, which builds shared memory systems for teams and agents. The author is the named inventor on the U.S. provisional patent application identified below.

Patent and rights notice. The mechanisms summarized in Sections 4 and 5, and the promotion gate of Section 7, are the subject of a U.S. provisional patent application, *Emergent Organizational Memory Formation, Lineage-Locked State Transitions, and Proceduralization for Multi-Agent Large Language Model Systems*, filed July 26, 2026, and a supplemental U.S. provisional patent application, *Lineage-Aware Control of Consolidation Leakage and Evidence-Gated Procedure Promotion in Shared Multi-Agent Memory Systems*, filed November 9, 2026. Copyright 2026 Achiral Research. Patent rights reserved. No patent license is granted by publication of this article.

Data, code, and materials availability. This paper introduces no dataset or code. The binomial calculations in Section 7 can be reproduced from the stated assumptions. A companion research essay is at `achiral.ai/research/consolidation-leakage`, and preprints are listed at `achiral.ai/papers`.

Ethics and impact. Describing a leakage channel can help attackers as well as defenders. We describe the channel at the level of system design, without exploit techniques against any named system, because the defenses require the channel to be measured.

References

[1] Anthropic. Memory for Claude Managed Agents: agents that learn across sessions. 2026. claude.com.

[2] Anthropic. Bringing memory to Claude. 2025. claude.com.

[3] OpenAI. Dreaming: Better memory for a more helpful ChatGPT. 2026. openai.com.

[4] Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., and Gonzalez, J. E. MemGPT: Towards LLMs as Operating Systems. 2023. arXiv:2310.08560.

[5] Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. Generative Agents: Interactive Simulacra of Human Behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023. arXiv:2304.03442.

[6] Rasmussen, P., Paliychuk, P., Beauvais, T., Ryan, J., and Chalef, D. Zep: A Temporal Knowledge Graph Architecture for Agent Memory. 2025. arXiv:2501.13956.

[7] Xu, W., Liang, Z., Mei, K., Gao, H., Tan, J., and Zhang, Y. A-MEM: Agentic Memory for LLM Agents. 2025. arXiv:2502.12110.

[8] Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. Voyager: An Open-Ended Embodied Agent with Large Language Models. 2023. arXiv:2305.16291.

[9] Wang, Z. Z., Mao, J., Fried, D., and Neubig, G. Agent Workflow Memory. 2024. arXiv:2409.07429.

[10] van der Aalst, W. *Process Mining: Data Science in Action*. 2nd ed. Springer, 2016.

[11] Anderson, J. R. and Lebiere, C. *The Atomic Components of Thought*. Lawrence Erlbaum Associates, 1998.

[12] Doyle, J. A Truth Maintenance System. *Artificial Intelligence*, 12(3):231–272, 1979.

[13] de Kleer, J. An Assumption-Based TMS. *Artificial Intelligence*, 28(2):127–162, 1986.

[14] Alchourrón, C. E., Gärdenfors, P., and Makinson, D. On the Logic of Theory Change: Partial Meet Contraction and Revision Functions. *Journal of Symbolic Logic*, 50(2):510–530, 1985.

[15] Denning, D. E. A Lattice Model of Secure Information Flow. *Communications of the ACM*, 19(5):236–243, 1976.

[16] Denning, D. E. and Schlörer, J. Inference Controls for Statistical Databases. *Computer*, 16(7):69–82, 1983.

[17] Farkas, C. and Jajodia, S. The Inference Problem: A Survey. *ACM SIGKDD Explorations Newsletter*, 4(2):6–11, 2002.

[18] Carlini, N., Tramèr, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., Roberts, A., Brown, T., Song, D., Erlingsson, Ú., Oprea, A., and Raffel, C. Extracting Training Data from Large Language Models. In *30th USENIX Security Symposium*, 2021. arXiv:2012.07805.

[19] Shokri, R., Stronati, M., Song, C., and Shmatikov, V. Membership Inference Attacks against Machine Learning Models. In *IEEE Symposium on Security and Privacy*, 2017. arXiv:1610.05820.

[20] Rezazadeh, A., Li, Z., Lou, A., Zhao, Y., Wei, W., and Bao, Y. Collaborative Memory: Multi-User Memory Sharing in LLM Agents with Dynamic Access Control. 2025. arXiv:2505.18279.

[21] Sangaraju, V. M. and Vissa, S. Lineage-Aware Memory Governance: A Derivation-Gated Framework for Privacy-Preserving Column-Level Access Control in Enterprise AI Agents. 2026. arXiv:2610.07258.

[22] Fowler, M. Event Sourcing. 2005. martinfowler.com.

[23] Moreau, L. and Missier, P. (eds.). PROV-DM: The PROV Data Model. W3C Recommendation, 2013. w3.org.

[24] Wilson, E. B. Probable Inference, the Law of Succession, and Statistical Inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

[25] Wald, A. Sequential Tests of Statistical Hypotheses. *Annals of Mathematical Statistics*, 16(2):117–186, 1945.