BEYOND FRESHNESS: CLOSURE-AWARE ADMISSION FOR AGENT MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Persistent memory enables large language model agents to accumulate and reuse information across long-running interactions and multi-step tasks. However, source updates, corrections, and retractions can invalidate derived summaries, plans, and cross-record conclusions. Such memories may violate task constraints even when individual updates are locally correct, records appear fresh, or different maintenance schedules converge to the same state. We study memory-use admission under evolving information: deciding which derived memories remain valid for a request under explicit dependencies and task constraints, while balancing correctness, service availability, and maintenance cost. Direct version checking can miss transitive invalidation; invalidation alone does not restore service; and global rejection and repeated full validation can introduce unnecessary blocking and redundant computation, respectively. We propose Closure-Aware Admission (CAA), which combines task-constraint validation with transitive dependency closure to quarantine inadmissible memories while preserving unaffected service. A persistent incremental implementation reuses validation results that remain valid after updates. Controlled language-model experiments expose the compositional failures motivating this task. Further evaluations across mechanism families, source-driven replay, and native execution on SQLite and Mem0 in raw-storage mode show that CAA matches the decisions of the full transitive-closure baseline under the evaluated contracts, avoiding unsafe admissions missed by direct version checks and unnecessary service losses caused by global rejection. Across the evaluated SQLite and Mem0 workloads, persistent CAA reduces aggregate online execution time relative to persistent full-closure validation while preserving the same admission decisions. These results demonstrate the benefits of reusing valid verification results for efficient and reliable memory maintenance.

1 INTRODUCTION

Persistent memory gives an agent continuity: observations become summaries, summaries support plans, and plans guide later actions (Park et al., 2023; Shinn et al., 2023). Reusing these results allows work to accumulate across interactions and makes earlier evidence a premise of later decisions. As evidence changes, a stored result remains useful only while its assumptions and task obligations hold. Reliable reuse therefore requires a connection between the state maintained in memory and the conditions of its next use. This requirement becomes consequential in extended workflows, where intermediate conclusions are repeatedly consumed by downstream tasks (He et al., 2026).

Existing research has substantially improved information retention and access. Memory hierarchies extend the active context (Packer et al., 2023; Kang et al., 2025); consolidation and graph-based organization support evolving records (Chhikara et al., 2025; Xu et al., 2025; Rasmussen et al., 2025); learned memory operations improve downstream answers (Yan et al., 2026). Yet evaluations continue to expose difficulty with knowledge updates, selective forgetting, and the consequences of dependent changes (Wu et al., 2024; Hu et al., 2026; Jung et al., 2026). These findings place the validity of reused information alongside retrieval quality as a central requirement for stateful agents.

The central challenge is *compositional reliability*. Each update or derivation acts on currently available information, yet its output may support many later computations. Corrections can affect records several dependency steps away, and consolidation determines what information subsequent opera-

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

tions can recover. We isolate these effects by fixing initial memory, source events, and operation precedence. Even when each operation passes its local checks, different legal orders can yield different task-relevant outcomes. This exposes a failure of maintenance composition.

Agreement between orders leaves a second obligation: records must jointly satisfy their use requirements. Independently maintained values can faithfully represent their sources while jointly violating a declared constraint, as our controlled joint-constraint experiment demonstrates. Thus, local operation validity, agreement across schedules, and constraint satisfaction describe distinct aspects of reliability. Its consequences must then be bound to the current request. Global blocking interrupts unaffected work; limited propagation leaves indirect dependencies exposed. Rechecking the entire state also incurs repeated cost during stable periods.

We formulate *memory-use admission*: deciding whether a current request is eligible to consume maintained memory under explicit constraints, dependencies, and read membership. MORPHMEM provides qualified comparisons that diagnose maintenance composition. *Closure-Aware Admission* (CAA) then connects joint-state validation to selective authorization. It combines inactive records and failed-rule supports into an isolation seed, propagates their influence through the dependency graph, and checks the current read set. This construction retains service in validated, unaffected components. A persistent implementation reuses unchanged rule outcomes and affected-region computations; a conditional equivalence result establishes preservation of completed full-closure decisions.

The evaluation demonstrates the value of this combination. On 477 controlled inputs, CAA preserves all 369 eligible services while blocking all 96 ineligible requests. The direct-version guard admits 93 ineligible requests, and global validation withholds 75 eligible requests. Against full closure with matched decisions and storage calls, persistent validation reduces cumulative online time by 22.7% on SQLite and 17.4% on Mem0 in their respective workload grids. These results connect protection, retained service, and execution cost to specific components of the admission design.

Contributions. **(1) Compositional reliability diagnosis.** We formalize qualified legal-order comparison and establish maintenance-failure structure across three Qwen configurations and a Granite confirmation condition (Section 4.2; Appendix C). **(2) Closure-aware memory-use admission.** We specify a contract combining joint constraints, transitive influence, and current reads, together with persistent validation and conditional decision preservation (Section 3.3; Proposition 1). **(3) A correctness–availability–cost evaluation.** Controlled tests, component ablations, native backends, and source-driven replay establish selective protection, budgeted recovery, and the conditions supporting online savings (Sections 4.3–4.5).

2 Related Work

Retrieval-augmented generation supplies external evidence, while reflective agents preserve intermediate conclusions for later decisions (Lewis et al., 2020; Park et al., 2023; Shinn et al., 2023). Memory hierarchies and consolidation extend this reuse across interactions (Packer et al., 2023; Zhong et al., 2024; Chhikara et al., 2025). Graph-based retrieval, evolving links, temporal validity, and learned memory operations improve access and update handling (Gutiérrez et al., 2024; 2025; Xu et al., 2025; Rasmussen et al., 2025; Yan et al., 2026). Their materialized results can outlive the evidence from which they were constructed, making downstream validity an additional requirement.

Evaluation exposes the consequences of changing evidence. MQuAKE shows that successful factual edits can leave multi-hop implications unresolved (Zhong et al., 2023). LoCoMo and LongMemEval evaluate temporal reasoning, updates, and abstention; MemoryAgentBench and MemBench add incremental interaction, reflective memory, and efficiency measurements (Maharana et al., 2024; Wu et al., 2024; Hu et al., 2026; Tan et al., 2025). Dependent changes and post-deletion state remain difficult in MEME, while MemoryArena reports a gap between recall and interdependent task performance (Jung et al., 2026; He et al., 2026). These findings connect the quality of retained information to its consequences for subsequent use.

Truth maintenance and provenance track the supports of derived information (Doyle, 1979; de Kleer, 1986; Green et al., 2007). At agent execution boundaries, MemTX validates belief commits and propagates retractions, and PlanFence checks action-relevant dependency versions (Li et al., 2026;

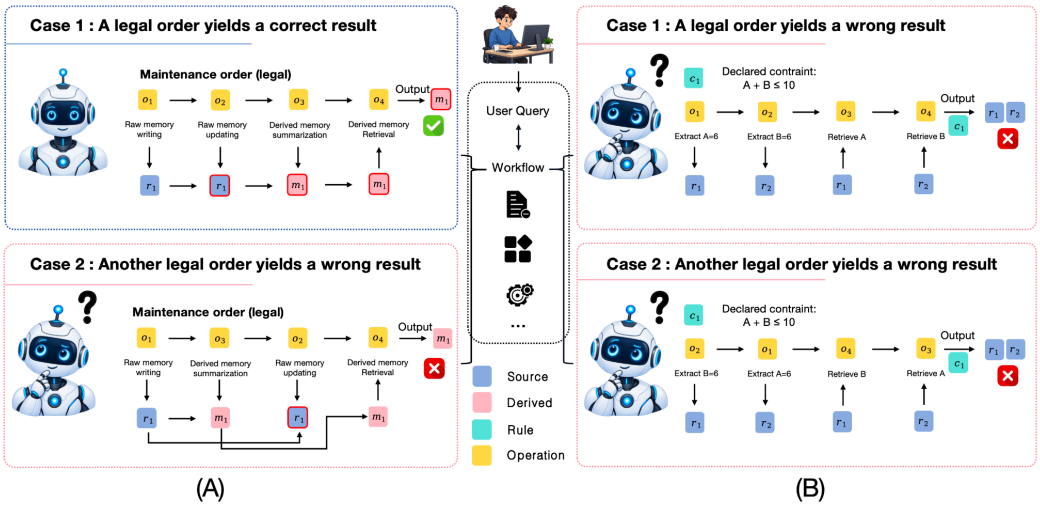


Figure 1: **Two compositional failure mechanisms.** Yellow o_i are operations, blue r_i sources, pink m_i derived records, and turquoise c_i constraints; i indexes objects. **(A)** Source update before summarization produces a current summary; the reverse order retains the previous source. **(B)** Independent writes assign $A = B = 6$ in one project and quarter. Both orders return $(6, 6)$, violating the declared constraint $A + B \leq 10$. Red crosses mark ineligible use. Appendix C specifies the contracts and comparisons.

Chen et al., 2026). Protection depends on the obligations and dependencies represented at these boundaries. Repeated checking connects this problem to incremental view maintenance, shared rule matching, and demand-driven computation (Gupta et al., 1993; Forgy, 1989; Hammer et al., 2014). Their savings depend on change locality and dependency-management cost, motivating joint evaluation of protection, retained service, and execution time. Appendix I provides additional literature context.

3 Methodology

3.1 Memory State and the Admission Task

Let R and M be finite, disjoint sets of source-record and derived-record identifiers, respectively. Write $V = R \cup M$ and $n = |V|$. At committed revision $t \in \{0, 1, \ldots\}$, the record map x_t assigns each identifier $v \in V$ its payload, version, provenance, scope, and activity flag $a_t(v) \in \{0, 1\}$. The value $a_t(v) = 1$ denotes an active record. Let $G = (V, E_G)$ be a directed dependency graph, where $(u, v) \in E_G$ means that record v depends on record u. Inactive identifiers remain in G to preserve propagation paths. With a finite set C of executable constraints, the memory state is

$$X_t = (R, M, x_t, C, G). \tag{1}$$

For a rule $c \in C$, define its evaluation $\eta_t(c) = \mathrm{Eval}(c, X_t)$ in $\{\mathsf{T}, \mathsf{F}, \mathsf{U}\}$, representing satisfaction, violation, and unresolved evaluation. The set $\mathrm{inp}(c) \subseteq V$ contains the records read by its declared bindings; $\mathrm{supp}(c) \subseteq V$ is its declared failure-localization support. An empty support marks a failure that requires global rejection. Unresolved bindings yield U. A rule's input footprint identifies the records needed for evaluation; its failure support identifies the records affected by a violation. Joint constraints can read several independently maintained sources, while source–derivative checks relate a stored result to its recorded evidence. The application supplies these declarations together with the request membership.

A request q specifies direct record reads and any query-selected members. Their union is $\mathrm{Read}_t(q) \subseteq V$. The flag $\kappa_t(q) \in \{0, 1\}$ records completeness of the required dependency declarations and query membership. Admission returns one of four outcomes: Allow authorizes service, Quarantine isolates a request exposed to a localized violation, Reject blocks use after

an unlocalizable violation, and UNKNOWN withholds authorization pending sufficient evaluation. Verification uses a supplied resource budget B_v; the separate repair budget $b \in \{0, 1, \ldots\}$ counts permitted reconstruction writes. This task is defined over the supplied rules, dependencies, and request membership.

3.2 QUALIFIED COMPOSITION DIAGNOSIS

Let $\mathcal{A}$ be a finite set of maintenance operations and $H = (\mathcal{A}, E_H)$ their directed acyclic precedence graph. The set $\Sigma(H)$ contains its legal linear extensions. For fixed initial state X_0, source-event sequence $\mathcal{E}$, and maintenance policy π, schedule $\sigma \in \Sigma(H)$ produces

$$X_\sigma = \mathrm{Execute}(X_0, \mathcal{E}, \sigma, \pi). \tag{2}$$

The predeclared observation $\mathcal{O}(X)$ maps a state to its task-relevant output, with equivalence relation $\equiv_{\mathcal{O}}$ specifying acceptable observational agreement. A qualified comparison fixes initial state, events, visibility, and precedence, and requires protocol, local-semantic, and runtime checks to pass. The semantic operator is identified by its model, representation, interface, and decoding configuration. Evaluation targets are held separately from online execution.

MORPH-MEM measures two properties: variation of $\mathcal{O}(X_\sigma)$ across legal schedules and satisfaction of task constraints by each outcome. It compares four maintenance policies: unchanged execution, invalidation of affected records, cascading refresh in dependency order, and canonical reconstruction from valid roots. These comparisons identify the information and dependencies required for reliable subsequent use. This relation-based design connects to metamorphic testing (Chen et al., 2018). Appendix C specifies qualification and controlled examples.

3.3 CLOSURE-AWARE ADMISSION

Define the inactive-record set $I_t = \{v \in V : a_t(v) = 0\}$, failed-rule set $F_t = \{c \in C : \eta_t(c) = \mathsf{F}\}$, and unresolved-rule set $U_t = \{c \in C : \eta_t(c) = \mathsf{U}\}$. The isolation seed is

$$S_t = I_t \cup \bigcup_{c \in F_t} \mathrm{supp}(c). \tag{3}$$

Starting with $Q_t^{(0)} = S_t$, iteration $k \in \{0, 1, \ldots\}$ expands the affected region by one dependency edge:

$$Q_t^{(k+1)} = Q_t^{(k)} \cup \{v \in V : \exists u \in Q_t^{(k)},\ (u, v) \in E_G\}. \tag{4}$$

Its fixed point $Q_t = \mathrm{Reach}_G^*(S_t)$ contains every seed and its descendants; Reach_G^* denotes reachability by zero or more edges. Finiteness of V guarantees termination. The fixed point is the least forward-closed set containing S_t. It therefore covers the declared consequences of each localized violation while retaining eligibility in validated, disjoint components.

Let $\chi_t \in \{0, 1\}$ indicate completion of the required validation and closure computation within B_v. For well-formed inputs, CAA applies the following priority-ordered decision:

$$\mathrm{Admit}(q, X_t) = \begin{cases} \textsc{Unknown}, & \chi_t = 0, \\ \textsc{Reject}, & \exists c \in F_t : \mathrm{supp}(c) = \varnothing, \\ \textsc{Quarantine}, & \mathrm{Read}_t(q) \cap Q_t \neq \varnothing, \\ \textsc{Unknown}, & U_t \neq \varnothing \text{ or } \kappa_t(q) = 0, \\ \textsc{Allow}, & \text{otherwise.} \end{cases} \tag{5}$$

Known violations isolate affected requests; unresolved obligations withhold authorization. Complete, resolved requests outside the isolation region receive authorization.

3.4 PERSISTENT VALIDATION AND NATIVE EXECUTION

The persistent execution contract fixes V, G, and C within a session and supplies a complete, revision-ordered change stream from one writer. Let $\Delta_t \subseteq V$ be the identifiers changed by the latest commit. The record–rule index

$$J(v) = \{c \in C : v \in \mathrm{inp}(c) \cup \mathrm{supp}(c)\} \tag{6}$$

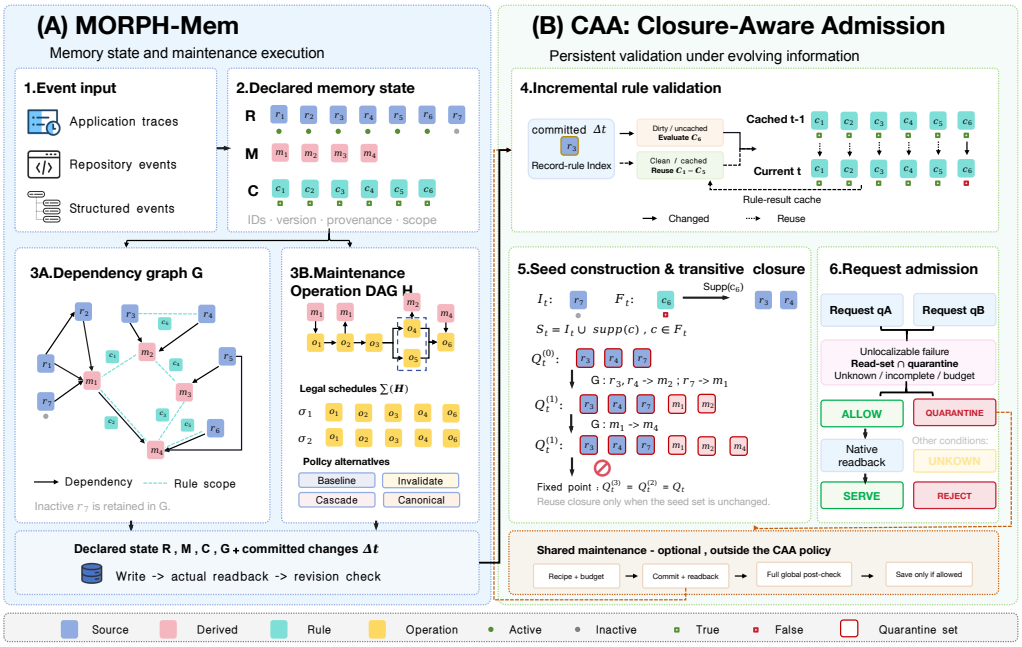


Figure 2: **Memory maintenance and closure-aware admission.** The left panel represents source records R, derived records M, constraints C, dependency graph G, and operation graph H. The right panel evaluates rules after committed changes, constructs S_t, computes the affected closure Q_t, and applies the current request decision. Persistent rule results and closure membership are reused under the conditions in Section 3.4. Shared reconstruction commits successor records and validates them through native readback.

covers both input reads and localization supports. Let $\mathcal{K}_{t-1}$ be a partial cache of rule outcomes, $\mathrm{dom}(\mathcal{K}_{t-1})$ its cached rules, $P_{t-1} \subseteq C$ the pending rules left dirty by an interrupted evaluation, and $\mathcal{B}_{t-1} \subseteq C$ the rules with unresolved bindings requiring reconsideration. The dirty set is

$$D_t = \big(C \setminus \mathrm{dom}(\mathcal{K}_{t-1})\big) \cup P_{t-1} \cup \bigcup_{v \in \Delta_t} J(v) \cup \mathcal{B}_{t-1}. \tag{7}$$

Initialization sets $\mathcal{K}_{-1} = \varnothing$ and $P_{-1} = \mathcal{B}_{-1} = \varnothing$, making every rule dirty. Completed evaluations update the cache and clear the corresponding pending flags, including for F and U outcomes. Clean results retain their existing meaning. Once the current seed is assembled, its equality with the previous seed permits reuse of Q_t on fixed G. Each request's reads and completeness conditions are checked at service time.

Proposition 1 (Completed-decision preservation)**.** *Assume deterministic rule evaluation, fixed record identifiers, graph and rule declarations, complete committed changes, an index covering every rule input and support, and identical request conditions. If persistent and full-closure evaluation both complete, they produce the same rule outcomes, isolation closure, and admission decision.*

Appendix B gives the induction and computational scope; Algorithm 1 in Appendix B.3 specifies execution.

Native execution verifies committed contents using backend-returned identifiers and revision checks. Authorized service reads the stored contents. Shared reconstruction spends up to b writes, commits successors, performs actual readback, and applies a common post-check. Budget-zero execution evaluates admission; positive budgets additionally evaluate recovery.

4 Experiments

The experiments connect three claims: maintenance failures persist across model configurations; closure-aware admission combines protection with retained service; and persistent validation re-

duces online cost at matched decision quality. The separately provided Supplementary Material gives protocols, metrics, proofs, and additional results (Appendices A–I).

4.1 Experimental Setup

Tasks and populations. The *maintenance-composition study* covers three Qwen configurations and a Granite cross-family confirmation condition, with local qualification, core mechanisms, joint constraints, and dependency-topology tests. The *admission suite* has 477 state–request inputs: 45 mechanism inputs (15 families, three value seeds) and 432 scale inputs (108 design units, four snapshots). The reference labels are 369 eligible, 96 ineligible, and 12 unresolved. The *repository-state application* executes 182 checkpoints from 79 of 94 retained external histories through a source–memo–plan cache. Its 738 events also include 492 unresolved-state and 64 out-of-scope events. Tables A1 and A5 specify sample units and coverage; Appendix F defines the application.

Methods and common conditions. The admission suite compares unconditional service, a direct-version guard, global validation, canonical reconstruction followed by global validation, full transitive quarantine, CAA reference evaluation, and snapshot-based incremental CAA. All main admission rows use $b = 0$. An independent evaluator assigns reference labels from the declared predicates, dependency reachability, and request conditions. Every method receives the same public inputs, while evaluation targets remain in the evaluation process. Full transitive quarantine (*full closure*) recomputes the rule outcomes and affected region. Native comparisons use persistent implementations of full closure and CAA with the same committed changes and storage calls. Repository replay compares $b \in \{0, 1, 2\}$ under a common copy-based reconstruction procedure, together with direct source access. Appendix A.2 specifies each operation.

Metrics. *Correct service* counts returned outputs accepted by the task evaluator. *Unsafe admission* counts ALLOW decisions on reference-ineligible inputs; *eligible blocking* counts withheld authorization on reference-eligible inputs. In model comparisons, a *substantive failure* violates a content, constraint, or observation requirement; an *availability-only failure* consists solely of withholding required output. A *joint pass* satisfies both schedules and their observation contract. Unresolved reference labels are accounted for separately. Repository replay additionally counts *incorrect service* and *withheld service*, which together with correct service partition the executable checkpoints. *Online time* covers admission and native service; *policy time* is its admission component. The *measured subtotal* adds setup, result export, independent evaluation, logging, and teardown. All times use seconds. Appendix A.3 defines the metrics, denominators, and aggregation equations.

Native workloads and repetitions. SQLite crosses $n \in \{32, 128, 512\}$ records with chain/star dependencies and four update modes: read-only, one relevant record, one unrelated record, and $\lceil n/4 \rceil$ records. Each of the 24 workloads has 16 steps and five reinitialized repetitions. Mem0 uses six workloads at $n = 32$, the first three update modes, and two repetitions. Its integration uses `mem0ai==2.0.20`, `infer=False`, nonsemantic hash embeddings, and exact returned identifiers. Repository timing uses four SQLite and two Mem0 repetitions; outcome counts represent one traversal.

4.2 Reliability across Models and Maintenance Policies

The core task contains six change mechanisms and a decision-preserving control. On Qwen3-8B's 420 qualified pairs per policy, baseline produces 360 substantive failures. Invalidation reduces these to 60 and introduces 300 availability-only failures. Cascade retains 60 substantive failures, while canonical reconstruction achieves 420 joint passes (Table A8). These outcomes separate suppressing invalid use from reconstructing usable information.

Qwen3-14B and Qwen3.8-27B-FP8 reproduce the family–policy correctness and repair pattern. The latter's 42 pairs per policy yield substantive failure rates of 85.7%, 14.3%, 14.3%, and 0%. Granite and its Qwen control confirm the structure on worlds disjoint from selection of a shared natural-language representation. Eight-order coverage comprises 1,344 main-study pairs and 896 per confirmation model, with zero qualification failures. A separate constraint experiment yields equal, invalid allocations across legal orders. This replication establishes maintenance composition and

Table 1: **Maintenance outcomes across model scale and execution stack.** Core compares six change mechanisms and one decision-preserving control. Qwen3-8B uses 420 qualified pairs per policy; Qwen3.8 uses 42. Rates (%) use Equation A6. Substantive failure, availability-only failure, and joint pass are defined in Section 4.1. Qwen3-14B and Granite confirmation coverage appears in Table A5.

	Qwen3-8B / BF16			Qwen3.8-27B / FP8		
Policy	Substantive ↓	Availability	Joint ↑	Substantive ↓	Availability	Joint ↑
Baseline	85.7	0.0	14.3	85.7	0.0	14.3
Invalidation	14.3	71.4	14.3	14.3	71.4	14.3
Cascade	14.3	0.0	85.7	14.3	0.0	85.7
Canonical	0.0	0.0	100.0	0.0	0.0	100.0

Table 2: **Admission correctness on the finite-contract suite.** Each method receives 477 inputs and repair budget $b = 0$. The independent labels comprise 369 eligible, 96 ineligible, and 12 unresolved inputs. Correct service, unsafe admission, and eligible blocking are defined in Section 4.1 and Appendix A.3. Full-closure and CAA methods withhold all 12 unresolved inputs.

Method	Correct service ↑	Unsafe admission ↓	Eligible blocking ↓
Unconditional service	369	96	0
Direct-version guard	369	93	0
Global validation	294	0	75
Canonical + global validation	294	0	75
Full transitive quarantine	**369**	**0**	**0**
CAA reference	**369**	**0**	**0**
CAA snapshot incremental	**369**	**0**	**0**

Table 3: **Service continuity in repository-state replay.** Outcomes cover the same 182 executable checkpoints; each row agrees across SQLite and Mem0. b is the permitted number of shared reconstruction writes. Correct, incorrect and withheld service partition the checkpoints. Direct access reads the current source. Appendix F gives event interpretation, native calls and timing.

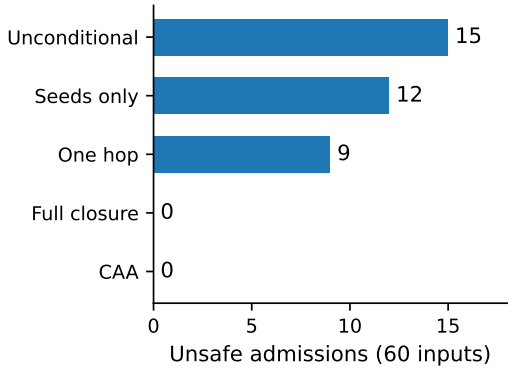
Method	b	Correct ↑	Incorrect ↓	Withheld
Unconditional service	0	166	16	0
CAA / full closure	0	166	0	16
CAA / full closure	1	175	0	7
CAA / full closure	2	**182**	**0**	0
Direct source access	0	182	0	0

joint-state validity as distinct requirements. Appendix C gives model settings, mechanisms, representation sensitivity, and model-call costs.

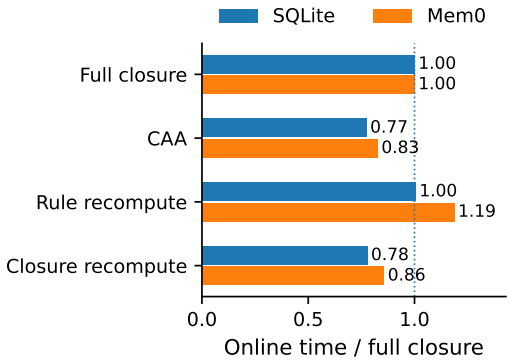
4.3 Request-Level Correctness and Service Continuity

Selective isolation retains eligible requests. On the finite-contract suite, the direct-version guard admits 93 of 96 ineligible inputs. Global validation blocks every ineligible input and also blocks 75 of 369 eligible inputs, an eligible-blocking rate of 20.3%. Both CAA implementations retain all 369 correct services and achieve zero unsafe admissions and zero eligible blocking (Table 2). All 954 full-closure comparisons agree on decisions, isolation sets, rule outcomes, and served content. Selective isolation retains 75 additional eligible services over global validation at zero unsafe admission. Full-closure agreement fixes the decision target for the efficiency comparison.

Shared reconstruction restores affected service. Unconditional repository replay returns 166 correct and 16 incorrect outputs. Budget-zero CAA preserves the 166 correct outputs and withholds the affected requests. Shared reconstruction raises correct service to 175 at $b = 1$ and 182 at $b = 2$, with zero incorrect service (Table 3). Full closure matches these outcomes at every budget on both backends. The 16 unprotected errors occur within three histories. Direct source access serves



(a) Isolation reach and unsafe admission.



(b) Validation reuse and online time.

Figure 3: **Ablations of protection and computation.** (a) Unsafe admissions in the 60-input propagation diagnostic; lower is better. (b) Online time divided by full-closure time within each backend's workload grid; lower is better. Rule recomputation refreshes every rule outcome, and closure recomputation rebuilds the affected region at every step. Both retain the reference decision semantics.

Table 4: **Native efficiency with matched decisions and storage calls.** Times are cumulative seconds over 24 SQLite workloads with five repetitions and six Mem0 workloads with two repetitions. Policy time is included in online time. The measured subtotal includes initialization and teardown.

Backend	Method	Policy	Online	Subtotal	Storage calls
SQLite	Full closure	9.941	36.239	117.425	83,040
	CAA persistent	1.949	**28.026**	110.158	83,040
Mem0	Full closure	0.170	0.750	12.092	1,056
	CAA persistent	0.043	**0.620**	10.236	1,056

all checkpoints with 356 storage calls, compared with 728 for budget-two CAA, establishing an efficient alternative for this copy-based task. Appendix F reports the complete execution and cost results.

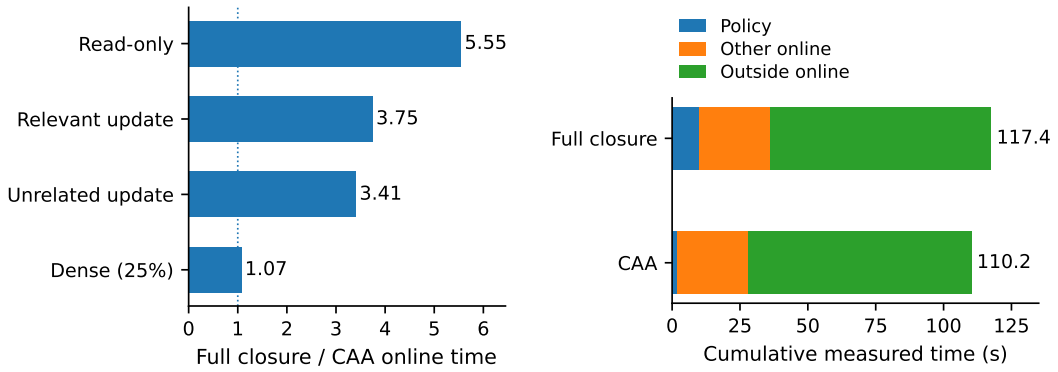
4.4 Component Ablations

Propagation depth determines protection. The 60-input diagnostic crosses five chain depths, four state conditions, and three value seeds. The conditions cover a consistent state, an updated root, a reconstructed chain, and a request to an unrelated component. Seed-only isolation retains rule evaluation and isolates the initial supports. One-hop isolation additionally follows one dependency edge; full closure follows the complete reachable region. Seed-only and one-hop isolation yield 12 and nine unsafe admissions, respectively; full closure and CAA yield zero (Figure 3a). One-hop fails at depths 3, 8, and 32. All methods retain eligible service, isolating propagation reach (Appendix D).

Rule-result reuse explains the online savings. We separately force rule recomputation and closure recomputation while preserving the persistent input interface. On SQLite, rule recomputation raises cumulative online time from 28.026 to 36.415 seconds, close to full closure's 36.239 seconds. All persistent variants retain identical decisions and storage calls (Figure 3b). Thus, rule-result reuse supplies the principal measured online savings, while the propagation diagnostic establishes the protective role of full reachability. The two ablations assign separate roles to the propagation semantics and the reuse of completed computation.

4.5 Protection-Preserving Cost and Workload Sensitivity

Persistent methods match all service outcomes, with zero unsafe admission and eligible blocking (Appendix E). Under this matched behavior, CAA reduces cumulative online time by 22.7% on



(a) Online benefit across update modes.

(b) SQLite execution-cost components.

Figure 4: **Workload sensitivity and cost attribution.** (a) Full-closure time divided by CAA time in each of four categorical update conditions; higher indicates greater savings. (b) Policy time, other online work, and work outside online execution partition the measured subtotal. Appendix A.3 gives the exact aggregation and component definitions.

SQLite and 17.4% on Mem0 (Table 4), using each backend's grid and repetition protocol (Section 4.1).

Closure caching adds a smaller reuse benefit. With rule-result caching retained, closure reuse reduces cumulative online time by an additional 0.507% on SQLite and 3.493% on Mem0 relative to recomputing full closure (Table A12). Both variants preserve complete propagation. These descriptive differences isolate closure-cache reuse.

Update locality controls reusable validation. On SQLite, the ratio of full-closure to CAA online time is 5.55 for read-only sequences, 3.75 for relevant single-record updates, 3.41 for unrelated single-record updates, and 1.07 for dense updates (Figure 4a). Each mode contains 480 requests per method. Dense changes affect more rules and reduce the reusable portion of validation. Appendix E separates cold, changed, and unchanged requests.

Lifecycle accounting identifies the realized savings. SQLite's measured subtotal decreases from 117.425 to 110.158 seconds (6.2%; Figure 4b). Initialization accounts for 95.2% of Mem0's subtotal difference, making its online comparison the direct efficiency estimate. Appendix E separates lifecycle costs; Appendix E.5 quantifies snapshot-interface overhead.

Measurement scope. Times aggregate five SQLite and two Mem0 repetitions. Small closure-cache differences are descriptive point estimates with unquantified repeat-level dispersion (Appendix E.4).

Applicability. The deployment setting comprises repeatedly queried derivatives, explicit constraints, and complete single-writer changes over a fixed graph. Benefits arise from selective isolation and stable or localized updates. Semantic dependency discovery, concurrent updates, and general incremental-rule comparisons extend this scope.

5 Conclusion

Memory-use admission connects evolving information to its conditions of use. MORPH-Mem establishes cross-model evidence for maintenance-composition failures. CAA combines explicit constraints, transitive dependencies, and current reads into selective authorization, with persistent validation preserving completed full-closure decisions. The experiments establish protection, retained service, budgeted recovery, and workload-dependent online savings, providing an operational foundation for reliable reuse in stateful agents.

REPRODUCIBILITY STATEMENT

Section 3 defines the state, rule, dependency, request, and change contracts. Appendix A specifies the experimental populations, comparison methods, metric equations, and repetition protocols. Appendix B provides the conditional equivalence proof. Appendices C–F give the controlled operations, component diagnostics, full cost tables, and executable application semantics. Each quantitative figure is accompanied by a numeric table and plotting script in the manuscript project.

ETHICS STATEMENT

The repository-state application uses externally sourced event histories to drive a typed memory task. Admission decisions express eligibility under the declared task contract. Evaluation jointly reports incorrect use and withheld eligible service to account for the consequences of both failure modes. Publication of source-derived materials requires review of permissions, personal information, credentials, and anonymization. Deployment in consequential settings requires application-specific validation of rules, dependency completeness, and recovery behavior.

AI USE STATEMENT

Generative AI assisted with conceptual framing, experiment design, implementation and debugging, analysis, manuscript preparation, and figure design. The model-based maintenance experiments use separately specified semantic operators. Quantitative tables use the experiment results supplied for this study, and finite-contract evaluation uses an independent executable evaluator. The authors are responsible for the final claims, experimental artifacts, source attribution, and disclosure.

REFERENCES

Umut A Acar, Guy E Blelloch, and Robert Harper. Adaptive functional programming. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 28(6):990–1034, 2006.

Carlos E. Alchourrón, Peter Gärdenfors, and David Makinson. On the Logic of Theory Change: Partial Meet Contraction and Revision Functions. *The Journal of Symbolic Logic*, 50(2):510–530, 1985. doi: 10.2307/2274239.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avi Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. In *International Conference on Learning Representations*, 2024.

Peter Bailis, Alan Fekete, Michael J Franklin, Ali Ghodsi, Joseph M Hellerstein, and Ion Stoica. Coordination avoidance in database systems. *Proceedings of the VLDB Endowment*, 8(3):185–196, 2014.

Evan Chen, Shiqiang Wang, and Christopher G Brinton. Fresh Memory, Stale Plans: Dependency-Scoped Validation for Distributed LLM-Agent Memory. *arXiv preprint arXiv:2609.03340*, 2026.

Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, TH Tse, and Zhi Quan Zhou. Metamorphic testing: A review of challenges and opportunities. *ACM Computing Surveys (CSUR)*, 51(1):4:1–4:27, 2018.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory. *arXiv preprint arXiv:2504.19413*, 2025.

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

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating prompt injections by design. *arXiv preprint arXiv:2503.18813*, 2025.

Jon Doyle. A truth maintenance system. *Artificial Intelligence*, 12(3):231–272, 1979.

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, et al. LightMem: Lightweight and Efficient Memory-Augmented Generation. *arXiv preprint arXiv:2510.18866*, 2025.

Charles L Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. In *Readings in artificial intelligence and databases*, pp. 547–559. Elsevier, 1989.

Todd J Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 31–40, 2007.

Ashish Gupta, Inderpal Singh Mumick, and Venkatramanan Siva Subrahmanian. Maintaining views incrementally. *ACM SIGMOD Record*, 22(2):157–166, 1993.

Bernal J Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models. *Advances in Neural Information Processing Systems*, 37:59532–59569, 2024.

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. From RAG to Memory: Non-Parametric Continual Learning for Large Language Models. *arXiv preprint arXiv:2502.14802*, 2025.

Matthew A Hammer, Khoo Yit Phang, Michael Hicks, and Jeffrey S Foster. Adapton: Composable, demand-driven incremental computation. *ACM SIGPLAN Notices*, 49(6):156–166, 2014.

Zexue He, Yu Wang, Churan Zhi, Yuanzhe Hu, Tzu-Ping Chen, Lang Yin, Ze Chen, Tong Arthur Wu, Siru Ouyang, Zihan Wang, et al. MemoryArena: Benchmarking Agent Memory in Interdependent Multi-Session Agentic Tasks. *arXiv preprint arXiv:2602.16313*, 2026.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating Memory in LLM Agents via Incremental Multi-Turn Interactions. In *International Conference on Learning Representations*, 2026.

Seokwon Jung, Alexander Rubinstein, Arnas Uselis, Sangdoo Yun, and Seong Joon Oh. MEME: Multi-entity & Evolving Memory Evaluation. *arXiv preprint arXiv:2605.12477*, 2026.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory OS of AI Agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25961–25970, 2025.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

Xiaoyang Li, Yiqi Wang, Haohui Lu, Zhi Chen, Mo Li, Pingan Song, Mingkai Zheng, and Taotao Cai. MemTX: Transactional Belief Commit for Stateful Agent Memory. *arXiv preprint arXiv:2607.23929*, 2026.

Lei Liu, Xiaoyan Yang, Yue Shen, Binbin Hu, Zhiqiang Zhang, Jinjie Gu, and Guannan Zhang. Think-in-Memory: Recalling and Post-thinking Enable LLMs with Long-Term Memory. *arXiv preprint arXiv:2311.08719*, 2023.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating Very Long-Term Conversational Memory of LLM Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870, 2024.

Frank McSherry, Derek Gordon Murray, Rebecca Isaacs, and Michael Isard. Differential Dataflow. In *CIDR*, 2013.

Andrey Mokhov, Neil Mitchell, and Simon Peyton Jones. Build systems à la carte. *Proceedings of the ACM on Programming Languages*, 2(ICFP):1–29, 2018.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. MemGPT: Towards LLMs as Operating Systems. *arXiv preprint arXiv:2310.08560*, 2023.

Christos H Papadimitriou. The serializability of concurrent database updates. *Journal of the ACM (JACM)*, 26(4):631–653, 1979.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A Temporal Knowledge Graph Architecture for Agent Memory. *arXiv preprint arXiv:2501.13956*, 2025.

Rana Salama, Jason Cai, Michelle Yuan, Anna Currey, Monica Sunkara, Yi Zhang, and Yassine Benajiba. MemInsight: Autonomous Memory Augmentation for LLM Agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 33136–33152, 2025.

Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *Symposium on Self-Stabilizing Systems*, pp. 386–400. Springer, 2011.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. *arXiv preprint arXiv:2303.11366*, 2023.

Haoran Tan, Zeyu Zhang, Chen Ma, Xu Chen, Quanyu Dai, and Zhenhua Dong. MemBench: Towards More Comprehensive Evaluation on the Memory of LLM-based Agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 19336–19352, 2025.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A Controllable World of Apps and People for Benchmarking Interactive Coding Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory. *arXiv preprint arXiv:2410.10813*, 2024.

Rongwu Xu, Zehan Qi, Zhijiang Guo, Cunxiang Wang, Hongru Wang, Yue Zhang, and Wei Xu. Knowledge Conflicts for LLMs: A Survey. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8541–8565, 2024.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic Memory for LLM Agents. *Advances in Neural Information Processing Systems*, 38, 2025.

Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. Corrective retrieval augmented generation. *arXiv preprint arXiv:2401.15884*, 2024.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z Pan, et al. Memory-R1: Enhancing Large Language Model Agents to Manage and Utilize Memories via Reinforcement Learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12805–12825, 2026.

Michael Zhang and Eunsol Choi. SituatedQA: Incorporating Extra-Linguistic Contexts into QA. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7371–7387, 2021.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing Large Language Models with Long-Term Memory. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19724–19731, 2024.

Zexuan Zhong, Zhengxuan Wu, Christopher D Manning, Christopher Potts, and Danqi Chen. MQuAKE: Assessing Knowledge Editing in Language Models via Multi-Hop Questions. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 15686–15702, 2023.