Temporal Validity on Real Software Histories

Eliminating Stale-Fact Errors in Code-Assistant Memory over GitHub Fixes

A deterministic supersession memory, validated end-to-end on SWE-bench buggy→fixed pairs

Neeraj Yadav
MemStrata.dev — Called It Inc. (Enterprise)
memstrata@gmail.com

Draft v1

Companion to "Temporal Validity in Retrieval Memory" (Paper 1), which established the result on synthetic evolving benchmarks. This paper validates it end-to-end on real GitHub histories. All numbers are from the locked, paired run on identical cached scenarios (`REPORT_PAPER2.md`, `REPORT_PAPER2_forced.md`); local, deterministic (temperature 0, fixed seeds, no network). For double-blind submission, anonymize the author block and product identifiers.

Abstract

Retrieval-augmented generation (RAG) has no model of time: when a fact changes across a coding session — a function is renamed, an endpoint moves, a dependency is bumped — RAG retrieves both the old and new value with near-identical similarity and cannot tell which is current, so it serves the superseded value. Paper 1 showed, on synthetic single-value benchmarks, that a deterministic (subject, relation, object) supersession memory eliminates this failure. Here we validate it *end-to-end on real software history.* From 707 real GitHub issues (SWE-bench Lite + Verified) we extract 130 clean atomic state transitions — a fix that changes one identifiable value from a pre-fix to a post-fix form — and render each marker-free (the stale and current statements differ only in the value). On this set, MemStrata reaches **0.91** answer accuracy versus RAG's **0.57–0.59**; and, the structural result, when forced to answer RAG serves the superseded value **36–38%** of the time (an LLM reranker does not help) while MemStrata drives this to **≈0**, at RAG retrieval latency (∼2.1 s vs ∼18 s for the reranker). We are explicit about scope: only ∼18% of real fixes are clean atomic transitions; Paper 2 isolates the *memory mechanism* on that class, and extraction *coverage* of the remaining fixes is the orthogonal problem we defer to follow-on work. A real product bug surfaced and was fixed during the study (a case/punctuation-insensitive value comparison), with the moat property (deterministic-supersession accuracy on clean code mutations) preserved and verified.

1 Introduction

A coding assistant that persists memory across a session accumulates facts about a codebase — the name of a handler, a config value, a pinned version, an endpoint path. The binding difficulty is not recall but *currency*: these facts change, often within the same session, and an assistant that confidently reports last week's function name is worse than useless. Retrieval-augmented generation

[Lewis et al., 2020], the dominant memory mechanism, stores statements and retrieves by embedding similarity. It has no representation of time, so when a value changes it keeps both versions — "the login handler is `authenticate_user`" and "the login handler is `login`" — which sit close together in any embedding space. Retrieval surfaces both; the model cannot tell which is current; it abstains or serves the stale value.

Paper 1 demonstrated, on synthetic single-value benchmarks (code mutation, config migration, dependency bumps, API evolution), that this is a *structural* failure of similarity-based memory and that a deterministic supersession rule eliminates it. The natural objection is that synthetic benchmarks may flatter the method. This paper answers it: we validate the same mechanism **end-to-end on real GitHub buggy→fixed histories** drawn from SWE-bench [Jimenez et al., 2023].

Contributions.

1. **A real-data longitudinal benchmark.** From 707 real GitHub issues we extract 130 clean atomic state transitions (a verified pre-fix value → post-fix value), rendered marker-free so the only currency signal is order. The extraction, selection criterion, and marker-free invariant are explicit and reproducible (Section 3, Appendix A).

2. **An end-to-end win on real history.** MemStrata reaches 0.91 accuracy vs RAG's 0.57–0.59; forced to answer, RAG serves the superseded value 36–38% of the time and an LLM reranker does not help, while MemStrata reaches ≈0 — at RAG latency, with ∼48% bounded-growth compression (Section 5).

3. **A precise scope and an honest methodology.** We separate the *memory mechanism* (Paper 2) from *extraction coverage* (follow-on work), report a metrics ladder that leads with answer-level stale-fact-error, pair the abstention-allowed and forced regimes on identical scenarios, and disclose a product bug found and fixed mid-study with the moat verified (Section 6).

2 Related Work

Memory for LLM agents and RAG. Persistent-memory systems [Mem0; Chhikara et al., 2025] [MemGPT/Letta; Packer et al., 2023] [Park et al., 2023] and graph-structured RAG [Edge et al., 2024] emphasize recall over long contexts [LoCoMo; Maharana et al., 2024]; none introduces a notion of fact currency. MemStrata is orthogonal: a deterministic supersession rule over a bi-temporal ledger, evaluated under knowledge *evolution* rather than static recall.

SWE-bench and code evolution. SWE-bench [Jimenez et al., 2023] pairs real GitHub issues with their gold patches to evaluate whether models can *resolve* bugs. We repurpose its buggy→fixed pairs for a different question: not "can the model write the fix" but "once a fix changes a fact, does the memory keep the current value." To our knowledge this longitudinal, currency-focused use of SWE-bench is new.

Temporal knowledge and the synthetic precedent. Bi-temporal modeling (valid time vs transaction time) is long established in databases; Paper 1 adapts it to LLM memory and reports the synthetic result this paper validates on real data. We reuse Paper 1's architecture verbatim (Section 3).

3 Method

3.1 The memory mechanism (recap)

MemStrata stores facts like RAG, preserving recall, but routes each value-bearing turn through a deterministic assertion path: a clean (subject, relation, object) triple whose (subject, relation) key matches an active assertion with a *different* object **supersedes** it — the old assertion's validity interval is closed in a bi-temporal ledger and the new one opened, with no cosine threshold and no LLM judge. Retrieval surfaces only currently-valid rows. We use the published Paper 1 configuration (`temporal_v6`) unchanged, plus the one fix of Section 6.

3.2 Constructing real longitudinal scenarios

Each scenario is one buggy→fixed *atomic state transition* mined from a SWE-bench record (problem statement + gold patch) in three stages:

1. **Extraction (LLM).** A local model reads the problem statement and unified diff and emits a single change $(\text{subject}, \text{state\_a}, \text{state\_b}, \text{question})$ — the pre-fix and post-fix value of one identifiable atomic quantity (identifier, number, version, path, endpoint, config constant) — or declines when the patch is a multi-file refactor, a control-flow change, or carries several values.

2. **Deterministic guard.** A scenario is rejected unless the two values differ, are atomic ($\leq$40 chars, $\leq$4 tokens), and the phrased turns carry no recency tell.

3. **Self-validation (the selection criterion).** The scenario is kept *only if the production triple extractor keys the state-A and state-B sentences identically with objects equal to the two values* — i.e., the fix is one the supersession mechanism can engage cleanly. The selection is therefore principled (an atomic transition the temporal layer is defined for), not hand-picked.

Marker-free invariant. The state-A and state-B turns are textually identical except for the changed value ("The {subject} is {value}."); no old/new/current/deprecated wording. State-A is ingested before state-B; the question asks the current value. The only signal of currency is order, which only a temporal mechanism can exploit. **Yield:** 130 clean scenarios from 707 records (18.4%); the rest are real fixes that are not atomic transitions (Section 7).

4 Experimental Setup

All runs are local and deterministic (temperature 0, fixed seeds, no network, enforced by test). Answer model Qwen2.5-Coder-7B; correctness and fabrication judges Qwen2.5-Coder-3B (distinct from the answer model and each other, no self-grading); embedder nomic-embed-text (768-d).

Data. SWE-bench Lite (300) + Verified (500), human-curated real GitHub issues, fetched and sha256-pinned, deduplicated by instance id to 707 records.

Conditions (4). `no_memory` (floor), `naive_rag` (cosine top-k), `advanced_rag` (+ LLM reranker), and `temporal_v6` (the method). All four ingest the same turns and answer the same questions.

Regimes (2). *allowed* (the model may abstain) and *forced* (no abstention — exposes the stale-commitment that abstention hides). Both run on the **same cached 130 scenarios** (paired protocol below).

Metrics ladder. (i) *primary* — answer-level stale-fact-error (fraction of contradiction questions answered with the superseded value); (ii) *secondary* — accuracy; (iii) *tertiary* — conditional fabrication and memory compression; (iv) *mechanism diagnostic* — supersession-correctness. We

note that the ledger-level `stale_survivors` count is coarse (inflated by cross-scenario value collisions) and is not the headline; the answer-level stale-fact-error is.

Paired-sample protocol. Extraction is LLM-driven, so the first run caches the 130 scenarios (with instance ids) and every subsequent regime loads that cache. The allowed regime, re-run from the cache, reproduced `naive_rag` 0.569 / `advanced_rag` 0.585 / `temporal_v6` 0.908 exactly; allowed and forced are therefore paired on identical instances.

5 Results

metric	`naive_rag`	`advanced_rag`	`temporal_v6`
accuracy — allowed	0.569	0.585	**0.908**
accuracy — forced	0.615	0.592	**0.985**
stale-fact-error — allowed	0.262	0.262	**0.023**
stale-fact-error — forced	0.361	0.377	**≈0.00**
conditional fabrication — allowed	0.290	0.291	**0.168**
conditional fabrication — forced	0.654	0.651	**0.341**
memory (active facts)	260	260	**135**
compression	0%	0%	**48%**
mean retrieval latency	2.16 s	18.1 s	**2.13 s**

Table 1: Paper-2 result on 130 real GitHub scenarios (paired; allowed and forced on identical instances). The forced temporal stale-fact-error is reported as ≈0 because it read 0.000–0.015 across runs (a single answer-model flip on 130; see **Determinism**); the structural gap to RAG's 36–38% is invariant to that noise. Mechanism diagnostic: supersession-correctness 0.985, bounded-growth ratio 1.023.

The headline. Forced to commit, RAG serves the superseded value 36.1% of the time (26.2% even when it may abstain), and the LLM reranker does *not* help — it serves stale slightly more (37.7%), because reranking reorders retrieved chunks but cannot distinguish a stale value from a current one. MemStrata drives the stale-fact-error to ≈0 (0.023 allowed; forced 0.000–0.015 across runs, a single answer-model flip on 130 — see **Determinism** below), because the stale value is retired from the store before retrieval. The claim that survives run-to-run noise is the *structural gap* — RAG 36–38% versus MemStrata below 2.5% — not the exact zero. This reproduces Paper 1's synthetic "RAG 15–40% stale, temporal ∼0" on **real GitHub history**, at RAG latency, with ∼48% bounded-growth compression. Accuracy follows: 0.91/0.99 (allowed/forced) for MemStrata vs 0.57–0.62 for RAG.

Determinism. The pipeline — extraction, deterministic supersession, retrieval, and the ledger contents — is fully deterministic and reproduces exactly (the allowed matrix re-ran from cache bit-for-bit). The 7B answer model has minor run-to-run variance on the local runtime at temperature 0 (the temporal forced stale-error read 0.015 in one run and 0.000 in another — a single answer flip on 130); we therefore report the temporal forced stale-error as ≈0. The structural gap to RAG's 36–38% is unaffected.

6 Discussion

Mechanism versus coverage (the scope, stated as a decoupling). Paper 2 measures temporal *correctness conditional on clean extraction*: given a fix expressed as an atomic state transition, does deterministic supersession keep the current value? Whether an *arbitrary* GitHub fix can be reduced to such a transition — extraction *coverage* — is the orthogonal problem, and it is the subject of follow-on work. We evaluate the ∼18% of real fixes that are clean atomic transitions. This is scoping, not selection-for-victory: **RAG fails on the same selected subset** (0.57 accuracy, 36% stale), so the subset defines the regime where the temporal-memory problem exists, not the regime where our method happens to win. The selection criterion (Section 3.2) uses the production extractor, so the precise reading is "the mechanism, given clean extraction, on real data."

A product bug found and fixed, with the moat preserved. During validation we found that the assertion path compared values with a normalization that lowercases and strips punctuation, so a value changing *only* in punctuation or case (`Status('Good')`→`Status['GOOD']`, `/API`→`/api`) was misread as a duplicate and the stale value survived. The fix (`strict_object_supersede`, Appendix C) makes the comparison case- and punctuation-sensitive. It is moat-safe by construction — it can only ever supersede *more* pairs, never fewer — and we verify this: on the synthetic `code_mutation` benchmark the flag leaves accuracy at 1.000 and stale-fact-error at 0.000, and the full unit-test suite stays green. We report this transparently because finding and fixing such a bug mid-study, with the safety property checked, is part of the evidence that the result is real rather than tuned.

The negative result on reranking. `advanced_rag` (a learned reranker over retrieved chunks) tracks `naive_rag` throughout and serves stale slightly more when forced. Reranking improves *which* relevant chunks surface; it has no temporal signal, so it cannot solve currency. This mirrors Paper 1 and closes a path a reasonable designer might take.

7 Limitations

- **Coverage, not mechanism.** We evaluate the ∼18% of real fixes that are clean atomic transitions; the selection uses the production extractor, so Paper 2 is precisely "the mechanism, given clean extraction, on real data." Extending to multi-value / logic / behavior fixes is extraction-robustness work (follow-on).
- **Sample size.** 130 real scenarios — a focused mechanism result, not a leaderboard ranking; scaling to more records is straightforward future work.
- **Single 7B local model** on consumer hardware (as Paper 1); larger or cloud models may shift absolute baselines, not the structural gap.
- **Residual.** The answer model's ∼1-question run-to-run variance is disclosed (Section 5); the deterministic pipeline is unaffected.

8 Conclusion

For coding assistants over evolving codebases, the binding memory failure is currency, and RAG cannot maintain it by construction. Paper 1 showed this synthetically; Paper 2 shows it *end-to-end on real GitHub buggy→fixed histories*: on clean atomic transitions, a deterministic supersession memory reaches 0.91–0.99 accuracy where RAG reaches 0.57–0.62, and reduces the stale-fact-error

RAG serves 36–38% of the time to ≈0, at RAG latency, with bounded growth and the moat preserved. The memory mechanism generalizes from synthetic to real data; extending its extraction *coverage* to arbitrary fixes is the natural next step.

Reproducibility Statement

Deterministic (temperature 0, fixed seeds, no network, enforced by test). Pipeline: `eval/fetch_swebench.py` (sha256-pinned data) → `eval/swe_extract.py` (LLM patch reader + self-validation) → `eval/run_paper2.py` (matrix, two regimes, scenario cache so both regimes are paired). Sources: `REPORT_PAPER2.md` (allowed), `REPORT_PAPER2_forced.md` (forced). Gate-fix unit tests: `tests/memory/test_v6_gate_assertions.py`; moat guardrail: `eval/diag_moatcheck.py`; end-user demonstration: `eval/demo_stale_correction.py`.

References

Prateek Chhikara et al. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025. URL https://arxiv.org/abs/2504.19413.

Darren Edge et al. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024. URL https://arxiv.org/abs/2404.16130.

Carlos E. Jimenez et al. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023. URL https://arxiv.org/abs/2310.06770. ICLR 2024.

Patrick Lewis et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL https://arxiv.org/abs/2005.11401.

Adyasha Maharana et al. Evaluating very long-term conversational memory of llm agents. *arXiv preprint arXiv:2402.17753*, 2024. URL https://arxiv.org/abs/2402.17753.

Charles Packer et al. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. URL https://arxiv.org/abs/2310.08560.

Joon Sung Park et al. Generative agents: Interactive simulacra of human behavior. In *ACM CHI Conference on Human Factors in Computing Systems*, 2023. URL https://arxiv.org/abs/2304.03442.

A Scenario construction and examples

Each kept scenario is a verified atomic transition rendered marker-free. Three real examples (subject abbreviated), drawn from the cached set:

```
state-A : The function that handles login in api/auth.py is named authenticate_user.
state-B : The function that handles login in api/auth.py is named login.
question: What function handles login in api/auth.py?          gold: login

state-A : The API base path for the project is /api/v1.
state-B : The API base path for the project is /api/v2.
question: What is the API base path for the project?           gold: /api/v2
```

```
state-A : The fastapi version pinned in requirements.txt is 0.95.2.
state-B : The fastapi version pinned in requirements.txt is 0.110.0.
question: What fastapi version is pinned in requirements.txt?   gold: 0.110.0
```

State-A is ingested first; the two turns differ only in the value; the question targets the current (state-B) value. A scenario is kept only when the production triple extractor keys both turns identically with objects equal to the two values (Section 3.2).

B Extraction prompt

`swe_extract_change_v1.md` (abridged) — the patch reader. It emits a single atomic change or declines; a clean (subject, relation, object) re-extraction of both rendered turns must then agree before the scenario is admitted.

```
You read ONE real GitHub fix (problem statement + unified-diff patch) and extract a
SINGLE marker-free longitudinal change, IF the patch changes exactly one identifiable
atomic value (a renamed function, a changed default/constant/config value, a bumped
version, a moved endpoint, a changed parameter name). OLD value = before the fix (a '-'
line); NEW value = after the fix (a '+' line). Return ONLY JSON:
  {"is_change": true, "subject": "<stable thing; no value>",
   "state_a": "<old value>", "state_b": "<new value>",
   "question": "<present-tense, answerable by the new value, no recency words>"}
or {"is_change": false} for multi-file refactors, logic changes, or several values.
RULES: state_a != state_b, both ATOMIC; subject excludes both values; subject has no
leading article (the harness renders "The {subject} is {value}.").
```

C The `strict_object_supersede` fix and the moat guardrail

Bug. The assertion path's same-value test used a normalization that lowercases and strips punctuation, so an object changing only in punctuation/case normalized equal to the prior object, was judged a duplicate, and the stale assertion was reinforced — the change was dropped.

Fix. `strict_object_supersede` makes the object comparison whitespace-collapsed but case- and punctuation-sensitive, so such a change supersedes. It can only ever supersede *more* pairs (never fewer), so it cannot leave a stale value the prior path retired.

Guardrail (moat preserved). On the synthetic `code_mutation` benchmark, with the flag ON: accuracy 1.000 (unchanged) and stale-fact-error 0.000 (unchanged); the full unit-test suite stays green; the flag changes no `code_mutation` routing because its values differ in word characters. We promoted the flag to default-on after this guardrail held, and pinned the Paper-1 runner to the pre-fix comparison so its locked numbers reproduce exactly. Unit tests pin both the bug (flag off) and the fix (flag on).