Phren: A Git-Backed Project Memory System

Architecture, Governance, Concurrent Use, and Comparative Assessment

Ala Arab

March 12, 2026

Abstract

Persistent project memory becomes operationally important once work spans sessions, tools, and machines. This paper examines Phren as a markdown-native, git-backed project memory system with hook-driven capture, trust-scored retrieval, and configurable governance. The contribution is a systems-design case study and operator-workflow argument rather than a claim to invent long-term memory, progressive disclosure, or hybrid retrieval as general ideas. In Phren's default path, memory compounds as repo state while retrieval stays bounded through a local SQLite FTS5 index with no network hop, no required hosted service, and a more predictable latency profile for everyday code-memory lookup. Comparisons to Supermemory, claude-mem, and GitHub Copilot Memory are architecture-level unless explicitly labeled as measured Phren implementation notes.

1 Evaluation Frame

Agent memory systems are often compared on retrieval quality alone, even though production fit also depends on governance, portability, and operator workflow. This paper uses eight dimensions:

1. **Representation**: where memory lives and who owns it.
2. **Capture path**: how memory is created (hooks/API/workers).
3. **Retrieval path**: ranking, trust, and capped injection.
4. **Governance**: forgetting, validation, review, and policy controls.
5. **Economics**: token overhead + human maintenance burden.
6. **Portability**: migration risk and lock-in profile.
7. **Multi-agent readiness**: concurrent sessions and shared state.
8. **Operability**: CLI, shell, direct management without the model in the loop.

Phren should be read as a system in the same design space as prior memory work such as MemoryBank, MemGPT, claude-mem, Supermemory, Mem0, and Copilot Memory. The goal is not to claim invention of persistent memory itself, but to evaluate a particular combination of storage ownership, governance, and operator workflow.

2 Phren Architecture

The system has three primary subsystems: **Storage & Retrieval**, **Lifecycle Automation**, and **Governance**. Phren now supports two install modes rooted by `phren.root.yaml`: **shared** (`~/.phren/`) and **project-local** (`<repo>/.phren/`). In both modes, markdown and JSON files remain the source of truth for **findings**, **tasks**, **truths**, **fragment links**, and **session checkpoints**; the FTS5 index is a local cache. The diagram below shows the current runtime loop.

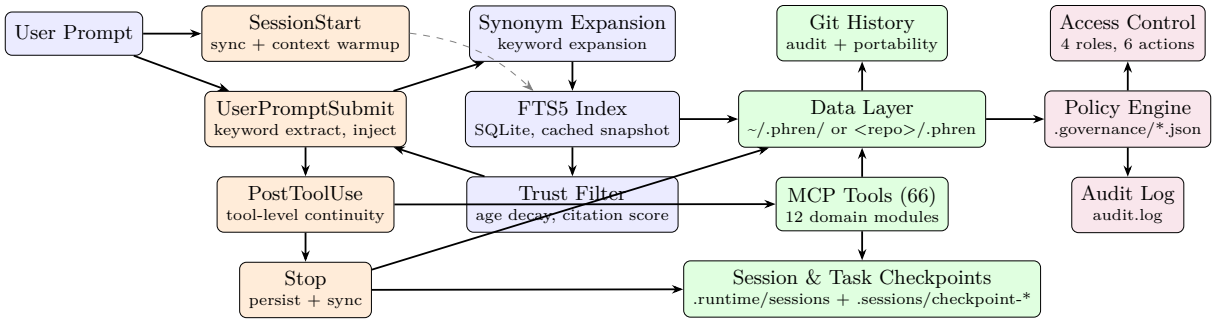

Figure 1: Phren runtime loop. Lifecycle events (`SessionStart`, `UserPromptSubmit`, optional `PostToolUse`, `Stop`) drive retrieval and persistence over a shared or project-local phren root. Findings, tasks, and checkpoint artifacts stay file-backed; FTS5 remains a local cache.

2.1 Storage and Retrieval

Phren keeps memory in local files and builds a local SQLite FTS5 index for retrieval [17, 19, 18]. In shared mode, state lives in `~/.phren`; in project-local mode, state lives in `<repo>/.phren` with a single primary project. Runtime path resolution prefers an explicit path or nearest rooted `.phren` before falling back to home scope. Markdown/JSON files remain the source of truth; the index is a cached snapshot that hook and MCP entrypoints refresh incrementally. This preserves inspectability (git history, plain-text grep, portable exports) while keeping retrieval bounded and local.

Before injection, a trust filter scores each entry by age decay and citation validity (Section 2.7), suppressing entries below a configurable confidence floor. Archived entries are stripped so consolidated history does not pollute active search. When semantic recovery is enabled, embeddings are cached locally and init can pre-warm coverage so the first real session is not starting from zero. Recovery queries use a persistent candidate index rather than re-embedding the corpus on every search.

2.2 Lifecycle Automation

Lifecycle automation is now a four-event model with tool-level continuity support [18, 20]:

1. `SessionStart`: sync + health checks + prior-session context restore.
2. `UserPromptSubmit`: keyword extraction, retrieval, trust filtering, and bounded context injection.
3. `PostToolUse`: optional hook surface for tool-call continuity and follow-up capture.
4. `Stop`: persistence path (save, sync scheduling, metrics/task finalization).

Claude uses native lifecycle hooks; Copilot/Cursor/Codex use generated wrappers + tool configs to enforce equivalent lifecycle semantics. The architectural point is unchanged: memory capture and retrieval stay inside the normal agent loop rather than becoming a separate manual habit.

2.3 MCP Module Surface

Phren MCP currently exposes **60 tools across 11 modules**: search/browse, task management, finding capture and lifecycle, memory quality, data management, fragment graph, session management, operations/review, skills management, hooks management, and extraction. Newly expanded lifecycle surfaces include contradiction/supersession tools for findings and session history/continuity tooling.

2.4 Token Budget and Progressive Disclosure

MCP tool responses consume context tokens, so retrieval quality is only half the problem; the other half is how much context the system spends to expose that memory. Phren uses progressive disclosure: automatic injection surfaces compact summaries with stable `mem:` IDs for large result sets, and agents expand only what they need. Tasks support summary-only, paginated, and single-item fetch modes.

This matters most in multi-tool coding sessions where search, task lookup, review notes, and shell output all compete for the same window. The design goal is to let project memory compound over time without turning long-lived context into prompt bloat. The same bounded path also keeps retrieval work small enough to stay responsive in daily use. A broad search may inject a compact index for five matching memories, after which the agent expands one `mem:project/file` entry instead of paying to inline every candidate up front.

2.5 Governance

The governance engine [19] provides four configurable policy layers: retention and TTL (default 120-day decay, 365-day retention), workflow approval gates for risky queue items, role-based access control (admin/maintainer/contributor/viewer with 6 action types), and canonical locks protecting truths from bulk overwrites. All governance events are appended to a timestamped audit log. These controls exist to contain familiar failure modes: stale memory, unreviewed risky writes, accidental overwrites, and unaudited shared-store drift.

2.6 Finding Lifecycle, Provenance, and Impact

Finding records now include normalized lifecycle metadata and provenance tracking. The lifecycle status set is `active`, `superseded`, `contradicted`, `stale`, `invalid_citation`, and `retracted`, with status updates and reasons persisted inline in finding metadata comments. Provenance is carried through a typed `source` field (`human`, `agent`, `hook`, `extract`, `consolidation`, `unknown`).

Impact scoring is session-linked rather than static: surfaced finding IDs are logged at retrieval time, completion outcomes are marked when session work lands, and repeatedly surfaced plus completed findings are promoted as higher-impact candidates in subsequent ranking.

2.7 Trust Decay and Confidence Scoring

Not all memories deserve equal weight. Each entry is scored before injection:

- **Age decay**: confidence degrades over four windows (`d30/d60/d90/d120`), defaulting to 1.0, 0.85, 0.65, and 0.45.
- **Citation validity**: entries with valid file/line/commit citations score higher; entries without citations receive a 0.8 multiplier.
- **Minimum confidence**: entries below 0.35 are sup-

pressed entirely.

- **Quality feedback**: helpful-count, reprompt penalty, and regression penalty are tracked per entry.

These thresholds are operational defaults rather than scientific constants; they are meant to suppress stale or weak context in day-to-day usage.

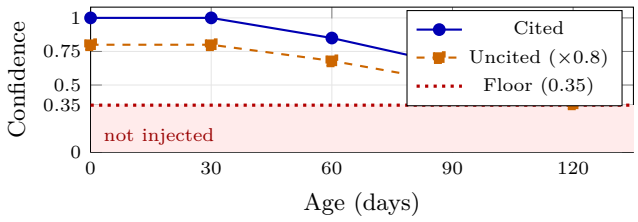

Figure 2: Trust decay. Cited entries hold full confidence for 30 days, then step down at 60/90/120. Uncited start at 0.8×. Below 0.35: suppressed. All thresholds configurable.

2.8 Extensibility

Nine custom integration hook events cover save, search, finding, indexing, session-end, and consolidation lifecycles [20]. Lifecycle registration also supports `PostToolUse` in addition to `SessionStart`, `UserPromptSubmit`, and `Stop`. Projects export and import as self-contained JSON bundles. Migration tooling mines git history, CI failures, and PR reviews to seed an initial `FINDINGS.md`.

3 Concurrent and Cross-Machine Use

Modern workflows frequently involve multiple sessions, tools, or machines touching the same project. Memory systems that assume a single serial session break under this pattern.

Phren handles multi-agent use through four mechanisms. **Dual-root storage model**: teams can share one git-backed root (`~/.phren`) or run isolated project-local roots (`<repo>/.phren`) per workspace. **Serialized writes with resilient sync**: concurrent reads are filesystem-safe; MCP writes are serialized by lock files with stale-lock cleanup; shared-mode sync uses rebase-first push retries with safe markdown auto-merge on common files. **Deterministic skill resolution**: project-scoped skills override global skills by name, and skills can be enabled/disabled without deleting source files. **Cross-session task checkpoints**: session end persists resumable checkpoint snapshots (task, edited files, failing tests, next-step hints), and session start restores them into continuity context.

This is an eventually consistent shared-memory model optimized for append-heavy agent workflows, not strong coordination in the distributed-systems sense.

3.1 Operator Surfaces

Phren ships three operator-facing review planes: `phren shell`, `phren web-ui`, and a VS Code extension. The web UI is hardened for local operation (loopback bind, per-run auth token validation, single-use CSRF tokens with TTL, CSP + anti-framing headers). The VS Code extension adds an activity-bar tree view for projects/findings/tasks/skills/hooks, an interactive fragment-graph webview, and first-run onboarding that validates local MCP/runtime wiring before enabling live operations.

4 Comparative Assessment

Evidence window: representative public documentation reviewed through March 6, 2026.

This section separates what public evidence can support from what still needs direct measurement. Documentation and product surfaces support architecture-level comparison, but not cross-vendor retrieval or latency claims on their own. Measured results in this paper are therefore limited to Phren implementation notes and shared-corpus local reruns called out explicitly. The systems below are reference points rather than an exhaustive survey.

4.1 Current comparison scope

- **Supermemory**: hosted API-first memory with metadata partitioning, graph-memory semantics, and benchmark tooling [5, 6, 7, 8, 9].
- **claude-mem**: local hook-driven memory with SQLite/FTS5 plus an optional semantic layer [10, 11, 12, 13].
- **Copilot Memory**: repository-scoped managed memory with policy controls and expiry semantics in GitHub Copilot [14, 15, 16].

Two claim types are currently defensible. Architecture-level differences such as ownership, governance, portability, and review workflow can be compared from public evidence. Direct retrieval or latency claims are defensible only for systems that have been run on shared corpora under controlled conditions. No public benchmark spans all four systems in that form. Managed API systems may offer broader centralized capabilities, but local retrieval paths can hold a real latency advantage when the query is exact-ish and the corpus already lives on disk.

4.2 Measured local comparison on a shared synthetic corpus

A fresh March 10, 2026 rerun exercised Phren and claude-mem on the same identifier-heavy synthetic corpora at 1k, 10k, and 100k memories using persisted local stores and the same query shape [22, 23, 24]. The result is operational rather than universal: Phren averaged 5.19/13.20/91.63 ms lexical query latency with exact top-hit rates of 12/12 at each size, while claude-mem averaged 221.07/147.91/225.11 ms with exact top-hit rates of 0/8. Even with `-skip-seed`, claude-mem still paid 25–4062 ms of Chroma-side backfill/check work before the query path settled. On this coding-style corpus with strong identifiers, Phren's lexical-first path was cheaper, more stable, and more exact.

4.3 Operational fit (non-empirical)

- **Phren**: favors repo-backed project memory, git auditability, and stronger governance; tradeoff is more

local setup and operator burden.

- **Supermemory**: favors a hosted API-first workflow; tradeoff is less local inspectability and portability.
- **claude-mem**: favors a lighter local hook-driven setup; tradeoff is a smaller governance and operator-tooling surface.
- **Copilot Memory**: favors teams already standardized on GitHub/Copilot; tradeoff is tighter platform scope and less control over storage and workflow.

4.4 Operational Design Differences

The systems differ less in whether they can retrieve a relevant memory and more in how they behave once memory becomes a shared, long-lived artifact.

- **Git-backed auditability**: every memory write is a commit. You can `git blame` any learning, revert a bulk operation, or audit who added what and when. Managed systems expose different audit primitives rather than equivalent git-native transparency.
- **Policy and access governance**: role-based access control, approval workflows, TTL and retention policy, and canonical locks are first-class features.
- **Trust filtering**: citation decay, confidence scores, and regression penalties suppress stale or low-quality memories before injection.
- **Human review surfaces**: `phren shell`, hardened `phren web-ui`, and the VS Code extension (tree view + graph + onboarding) let operators review, approve, and clean memories without starting an LLM session. The materials examined did not surface an equivalent model-free review path.

Any memory system can add a vector index. Governance, auditability, and review tooling require deliberate architectural choices that are harder to retrofit.

5 FTS5 vs Semantic Search

Phren is fundamentally anchored in SQLite FTS5 (BM25 ranking), with fallback layers in some paths. That choice is deliberate. In the default path, FTS5 gives Phren zero required hosted dependencies, deterministic and auditable results, low-latency lookup on typical corpus sizes, and offline operation with no retrieval-side token cost. The tradeoff is familiar: vocabulary mismatch remains real, paraphrased concepts do not align automatically, and BM25 ranks by term frequency rather than meaning. Semantic search retrieves better on paraphrase-heavy queries. FTS5 retrieves better when vocabulary is consistent, offline operation is required, or zero retrieval-side token cost matters. Phren's claim is narrower: lexical-first is often the right default for repo-backed coding memory because it is cheap, inspectable, and predictable.

5.1 Hybrid Retrieval: Current State

Phren now shares the same staged hybrid policy across hook injection and `search_knowledge`: strict FTS5 first, a relaxed lexical rescue query if the strict pass misses, cheap local fallback scoring, and vector recovery only when lexical hits are sparse or weak. The vector path uses persistent embeddings and a cached candidate index, then the shared reranker weights local lexical overlap ahead of weaker semantic tails. Benchmark claims still require published conditions because corpus size, cache warmth, and query mix change the outcome [22].

5.2 Retrieval Complexity

The staged design matters because each retrieval tier has a different cost profile. The practical hot path is sublinear because the linear fallbacks are gated and bounded rather than paid on every request.

5.3 Measured Retrieval Notes

These numbers are implementation checks on one live phren store, not a general ranking of memory systems. A single author-local benchmark on March 9, 2026 measured the current lexical-first and gated-hybrid paths against a warm Phren store with 139 indexed documents, real workflow queries, and the default 550-token hook budget [22]. Lexical latency averaged 14.71–15.93 ms; the gated hybrid path stayed in nearly the same range at 12.90–13.60 ms because the lexical stages usually satisfied the query before vector recovery was needed; injected payload remained stable at roughly 326–342 tokens; both modes hit every published query; and persistent candidate pruning still reduced the cosine stage to roughly 6.3–8.6% of the eligible corpus when vector recovery did fire. The interpretation is intentionally modest: on this corpus the lexical path was strong enough that vector recovery was usually unnecessary, so the hybrid path inherited nearly the same speed profile. Synthetic large-corpus scaling runs are documented separately because they answer a different question than this live-store measurement. The March 10 shared-corpus rerun against claude-mem reached the same practical conclusion from the other direction: on identifier-heavy coding queries, lexical-first retrieval stayed cheaper and more exact than the semantic-first alternative, even when the competing corpus had already been seeded. Taken together, the measured results support a narrow latency claim: local lexical-first retrieval is a real speed advantage for this style of code-memory workload.

6 Limitations

FTS5 retrieval ceiling. Vocabulary mismatch remains a real failure mode in large projects, partially mitigated by synonym expansion.
Hook-path indexing overhead. Index load and validation adds perceptible latency for large phren repos, even with a warm cache.
Ranking quality. BM25 is not calibrated to

agent-context relevance. The quality feedback loop requires usage history; new projects start cold.

Semantic recovery latency. On small corpora, vector candidate pruning changes asymptotic behavior but does not automatically improve wall-clock latency. Query embedding remains the dominant cost when semantic recovery fires.

Residual retrieval uncertainty. The published March 9 benchmark set no longer has misses, but that is still bounded evidence for one corpus and one query set. Broader paraphrase-heavy evaluation is still needed before making stronger semantic-completeness claims.

Governance complexity. The four-file governance system is powerful but non-trivial to configure. Initial setup is guided, but advanced policy configuration still requires direct CLI commands.

Web UI is opt-in. `phren web-ui` is not started automatically; memory triage requires an explicit invocation. This is intentional—triage is a deliberate act, not an ambient background process.

Uni-directional portability. Export/import works for project data; migrating governance configuration to another memory system is manual.

Operator burden. Local-first operation shifts responsibility to the user or team: git hygiene, policy tuning, backup expectations, and occasional conflict repair do not disappear just because the system is local.

7 Conclusion

The decision is less about "who retrieves best on a single prompt" and more about long-horizon operational fit. Phren treats memory as repo-backed project state rather than a hosted feature: trust filtering reduces noise in injected context, the governance engine adds policy control, and the shell plus review surfaces make memory maintenance possible without starting an LLM session. File locks serialize in-process writes, the background sync worker rebases and auto-merges safe markdown changes, and the shared store remains eventually consistent for append-heavy workflows.

Managed alternatives are reasonable when memory is expected to arrive bundled with a platform and data custody is not a priority. Phren fits best when the priority is repo-backed project memory: markdown as source of truth, predictable token discipline, git auditability, and enough governance to keep shared memory usable over time.

References

[1] Lewis, P. et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* https://arxiv.org/abs/2005.11401

[2] Asai, A. et al. *Self-RAG.* https://arxiv.org/abs/2310.11511

[3] Zhong, W. et al. *MemoryBank.* https://arxiv.org/abs/2305.10250

[4] Packer, C. et al. *MemGPT.* https://arxiv.org/abs/2310.08560

[5] Supermemory Docs: Introduction. https://supermemory.ai/docs/introduction

[6] Supermemory Docs: Memory API Creation. https://supermemory.ai/docs/memory-api/creation/adding-memories

[7] Supermemory Docs: Filtering. https://supermemory.ai/docs/memory-api/features/filtering

[8] Supermemory Docs: Graph Memory. https://supermemory.ai/docs/concepts/graph-memory

[9] Supermemory Docs: MemoryBench. https://supermemory.ai/docs/memorybench/quickstart

[10] claude-mem Docs: Overview. https://docs.claude-mem.ai/architecture/overview

[11] claude-mem Docs: Hooks Architecture. https://docs.claude-mem.ai/hooks-architecture

[12] claude-mem Docs: Database Architecture. https://docs.claude-mem.ai/architecture/database

[13] claude-mem Docs: Search Architecture. https://docs.claude-mem.ai/architecture/search-architecture

[14] GitHub Docs: Copilot Memory. https://docs.github.com/en/copilot/how-tos/use-copilot-agents/copilot-memory

[15] GitHub Changelog (Jan 15, 2026). github.blog

[16] GitHub Changelog (Mar 4, 2026). github.blog

[17] Phren README. https://github.com/alaarab/phren/blob/main/README.md

[18] Phren: `mcp/src/cli.ts`. https://github.com/alaarab/phren/blob/main/mcp/src/cli.ts

[19] Phren: `mcp/src/shared.ts`. https://github.com/alaarab/phren/blob/main/mcp/src/shared.ts

[20] Phren: `mcp/src/hooks.ts`. https://github.com/alaarab/phren/blob/main/mcp/src/hooks.ts

[21] Phren: `mcp/src/index.ts`. https://github.com/alaarab/phren/blob/main/mcp/src/index.ts

[22] Phren: `scripts/bench-retrieval-modes.ts`. https://github.com/alaarab/phren/blob/main/scripts/bench-retrieval-modes.ts

[23] Phren: `scripts/bench-claude-mem-synthetic.ts`. https://github.com/alaarab/phren/blob/main/scripts/bench-claude-mem-synthetic.ts

[24] Phren: `docs/benchmark-comparison-fair.md`. https://github.com/alaarab/phren/blob/main/docs/benchmark-comparison-fair.md