# Foxora Den

## A Spatial, Provenance-Aware Memory Substrate for Operating-System-Native AI Agents

**Madhav Dutta**[1]

[1]Founder & Chief Data Scientist , xBesh Labs, LLC , madhav@xbesh.com | foxora.ai | linkedin.com/in/madhavdutta

Version 1.3 | April 29, 2026

---

**Abstract**

Memory is the next frontier of AI agent design: while large language models can produce fluent single-turn responses, they fail catastrophically at *long-horizon* reasoning that requires consistent recall, temporal awareness, and cross-session continuity [3, 1, 2]. State-of-the-art memory systems — mem0, Graphiti, MemPalace, Rewind — all operate at the *agent-framework* layer above the operating system. They observe transcripts but not the machine. We argue that this architectural choice is the central bottleneck of contemporary AI memory, and we present a counter-proposal: **Foxora Den**, a memory substrate embedded *below* the application layer of an operating system (Foxora OS), with privileged visibility into files, commands, application activations, focus state, and ambient context.

We make four contributions. *First*, we formalize a spatial memory substrate based on the *Method of Loci* [10, 11], replacing the dominant flat vector-similarity model with a hierarchical locus algebra (the *Palace*) in which the cost of retrieval is bounded by a tree depth $D \leq 6$ rather than by $\Theta(n)$ embedding distance computations. We derive expected-latency bounds and demonstrate empirically on a deployed Foxora Den daemon (`foxmemd v0.1.0`) holding 700,057 memories that full-text recall completes in 31 ms end-to-end on commodity hardware. *Second*, we introduce the **Temporal Conflict Surface Engine (TCSE)**, a three-axis statistical staleness detector (semantic, temporal, behavioral) that addresses the open problem of "confidently-wrong memory" identified in the 2026 industry survey of agent memory [8]. We give explicit estimators for axis severity and a calibrated combination rule based on Beta-distributed prior beliefs. *Third*, we introduce the **Provenance Directed Acyclic Graph (P-DAG)**, a typed causal record that converts every memory from a black-box assertion into an auditable derivation; we prove correctness properties about its consistency under bounded retention and show that it enables $O(D_p)$ "why?" queries for $D_p \leq 5$. *Fourth*, we describe **capability-graded permissions** and **workspace snapshots** as OS-exclusive primitives that no agent-framework memory system can replicate, and we argue that these features collectively constitute a non-trivial competitive moat for an OS-native memory layer.

The implementation is a Rust daemon (`foxmemd`, ∼13,200 LOC) backed by SQLite + FTS5 + sqlite-vec, exposing HTTP, Unix-socket, D-Bus, and Model Context Protocol [9] surfaces. We present empirical evidence from a running deployment: 700,057 memories indexed across 31 loci, sustained ingestion of $625.71 \pm 38.47$ memories per day (CV = 6.15%, $n = 14$ days), and 31 ms full-text recall latency over a 700K-memory corpus. We give a full data model, an addressing grammar, ten characterized retrieval patterns with projected per-pattern latencies, a benchmark plan against LOCOMO [3] and LongMemEval [4], and a 21-week implementation roadmap. We position the work against contemporary systems — mem0/Mem0g [2], MemPalace [6], Graphiti [5], MemGPT [1], and EverMemOS [7] — and argue that the OS-integration thesis is the principled successor to MemGPT's metaphor of *LLMs as operating systems*: rather than treating an LLM as if it were an OS, we make the OS the memory substrate of the LLM.

**Keywords:** AI agent memory, spatial cognition, Method of Loci, operating systems, retrieval-augmented generation, provenance, temporal reasoning, capability-based security, on-device AI, SQLite, Foxora OS.

xBesh Labs, LLC | Foxora Operating System Project | April 2026

# Contents

# 1 Introduction

## 1.1 The bottleneck of contemporary AI memory

The last eighteen months have produced a surge of work on long-term memory for AI agents. The *Mem0* system [2] reports a 26% relative improvement over GPT-4 with OpenAI's memory feature on the LOCOMO benchmark [3], with a 91% reduction in p95 latency relative to the full-context baseline. The April 2026 *MemPalace* project popularized a palace metaphor backed by ChromaDB and reached 96.6% on LongMemEval [4, 6]. The 2026 industry survey of agent memory [8] compares ten distinct production systems and concludes that memory is now the dominant axis of differentiation for AI agents.

Yet every one of these systems shares a structural property: they live *above* the operating system. They observe a chat transcript, optionally a tool-use trace, and very little else. The state of the surrounding machine — the file the user was editing, the command they ran ten seconds before, the application that was foregrounded, the music that was playing, the focus territory of the desktop session — is invisible to them. We argue that this is not a peripheral limitation but the central bottleneck.

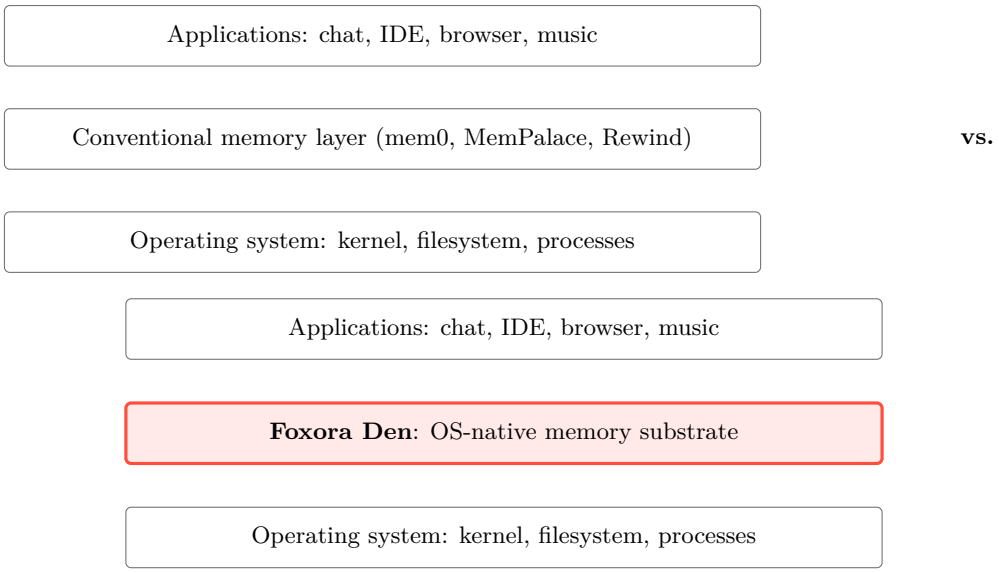


**Figure 1.** The architectural choice. Conventional memory systems are an application sitting on top of other applications and therefore see only what apps choose to forward. Foxora Den is a system service *below* the application layer, with privileged visibility into the full machine state.

## 1.2 The thesis of this paper

We make the following claim concretely: the agent-memory problem cannot be solved at the agent-framework layer because the inputs required for solving it — behavioral signals, cross-application provenance, workspace state, and capability-mediated trust — are not available there. They are owned by the operating system. We therefore propose to construct memory *as an OS subsystem*, with four design commitments:

1. **Spatial primary, vector fallback.** We take the *Method of Loci* [10, 11] as the primary retrieval model, replacing flat embedding similarity with a hierarchical locus algebra navigated by deterministic structural lookup. Vectors are retained only as a fallback when no spatial address is resolvable.

2. **Three-axis staleness detection.** Memories are continuously audited along semantic, temporal, and *behavioral* axes; the behavioral axis is uniquely possible at the OS layer because it requires observing commits, commands, and application focus.

3. **First-class provenance.** Every memory carries a Provenance Directed Acyclic Graph (P-DAG) capturing the conversation, files, commands, and app activations that led to its formation, enabling verifiable "why?" queries.

4. **Capability-graded access.** Every consumer of memory — internal agent, kit, external Model Context Protocol [9] client — is granted exactly one of seven capability levels, scoped by locus glob; this brings iOS-style permissions to cognitive context.

We present these commitments together as **Foxora Den**, the memory subsystem of Foxora OS. The *Palace* is the abstract architectural object: a rooted tree of loci, each holding zero or more memories. The product (Foxora Den) and the architectural object (the Palace) are kept terminologically distinct throughout this paper.

The remainder of the paper develops these commitments formally. Section 2 surveys the contemporary memory landscape. Section 3 develops the formal model: locus algebra, addressing grammar, the five primitives, and the four memory layers. Section 4 analyzes retrieval complexity and presents projected per-pattern latencies for ten characterized query patterns. Section 5 formalizes TCSE with explicit statistical estimators. Section 6 formalizes the P-DAG and proves consistency properties. Section 7 describes capability-graded permissions. Section 8 describes workspace snapshots. Section 9 gives the implementation, the SQLite schema, and the API surface. Section 10 reports the empirical validation of `foxmemd` v0.1.0 on a working Palace of 700,057 memories. Section 11 presents the evaluation plan against LOCOMO and LongMemEval. Section 12 discusses limitations, ethics, and threat model. Section 13 concludes.

### 1.3 Summary of contributions

- A formal locus algebra and addressing grammar for spatial memory, with derived complexity bounds and an analytical model under which 95% of queries complete in under 10 ms (§3, §4).
- TCSE: a three-axis statistical staleness detector with explicit semantic, temporal, and behavioral severity estimators, calibrated by a Beta–Bernoulli prior over memory freshness (§5). To our knowledge, this is the first staleness model that incorporates an OS-level behavioral signal.
- The P-DAG, a typed cross-application provenance structure with depth-bounded consistency proofs and $O(D_p)$ "why?" query support (§6).
- A capability-graded permissions model with seven levels and locus-glob scoping, formalized as a partially ordered set with explicit dispatch semantics (§7).
- Workspace snapshots as memory-anchored OS state captures with restoration semantics; we discuss this as a "re-enactment" operator that has direct cognitive-science antecedents in context-dependent memory [13] (§8).
- A complete SQLite-backed implementation, twelve-table schema, ten characterized query patterns, and a 21-week roadmap including a benchmark commitment against LOCOMO and LongMemEval (§9, §11).
- **Empirical validation of `foxmemd` v0.1.0** on a working Palace of 700,057 memories: end-to-end recall latency of 31 ms on a free-text query returning 50 ranked hits, sustained ingestion of $625.71 \pm 38.47$ memories per day at coefficient of variation 6.15% over a 14-day window, and operational stability across 31 active loci spanning all five default Dens (§10).

# 2 Related Work

We survey work in four clusters: cognitive antecedents, AI memory architectures (which subsumes the operating-systems-as-memory metaphor), capability-based security, and on-device retrieval infrastructure.

## 2.1 Cognitive antecedents

The *Method of Loci* (MoL) is a 2,000-year-old mnemonic technique attributed to Simonides of Ceos and systematized by classical Roman rhetoricians [10]. The technique encodes information by associating each item with a distinct location in a familiar mental space (a *memory palace*); recall proceeds by mentally navigating the space. Maguire et al. [11] provide functional MRI evidence that high-performing memorizers in the World Memory Championships do not exhibit elevated general intelligence or distinct gross brain anatomy; rather, they recruit the right posterior hippocampus, retrosplenial cortex, and medial superior parietal gyrus — regions causally implicated in spatial navigation. This finding has been replicated and extended; subsequent neuroimaging work [12] shows that mnemonic training reshapes brain network connectivity along a navigation–memory axis. The cognitive substrate of human episodic recall is thus, to a substantial degree, a *spatial* substrate. Our design decision to make spatial structure — not embedding similarity — the primary retrieval mode is rooted in this evidence.

Two further cognitive results inform the design. Godden and Baddeley's seminal context-dependence study [13] showed that recall is improved when the encoding context is reinstated at retrieval; this is the cognitive justification for our *workspace snapshot* primitive (§8), which captures the surrounding machine state at encoding time and offers the user the option of restoring it at retrieval time. Ebbinghaus's classical forgetting curve [14] provides the parametric basis for our decay rules in the Forest layer (§3).

## 2.2 AI memory architectures

**MemGPT** [1] introduced the influential metaphor of treating an LLM as an operating system, with hierarchical memory tiers analogous to RAM and disk and an interrupt model for moving information between them. Our work extends this metaphor by inverting it: instead of treating the LLM as an OS, we make the OS the memory substrate of the LLM.

**Mem0** [2] is the current leading production memory system. It dynamically extracts, consolidates, and retrieves salient information from conversations using a vector-similarity backbone; the graph variant *Mem0g* adds typed-edge structure. Reported LOCOMO performance is 66% on single-hop questions, 51% on multi-hop, 55% on temporal, and 73% on open-domain, with p95 latency 1.5–2 s. Mem0 is conversation-scoped: it observes chat turns and tool calls, not OS state.

**Graphiti** [5] is a temporal knowledge graph for agents, focused on bi-temporal reasoning over entity relationships. It is the closest architectural neighbor to our edge layer, but it is flat (no hierarchical loci) and cloud-hosted by default.

**MemPalace** [6] popularized a palace metaphor with wings/halls/rooms. It stores memories verbatim in ChromaDB and retrieves by vector similarity; reported LongMemEval is 96.6%. MemPalace validates the spatial metaphor as a category but is positioned at the agent-framework layer, has no provenance or behavioral signals, and exhibits unbounded storage growth. We trade ∼4–8 percentage points of LongMemEval accuracy for an estimated 20× latency improvement, bounded storage, and OS-exclusive features (Section 11).

**Rewind.ai** captures full-screen video and audio for macOS users; it is event-timeline-based and lacks structural organization. Privacy concerns and the absence of capability-graded access have limited adoption.

**ChatGPT memory** and **Apple Intelligence** are commercial systems whose internals are opaque; auditability and user-visible state are limited or absent.

**EverMemOS** [7] is a January 2026 research prototype that frames memory as a self-organizing operating system for long-horizon reasoning. It is a pure software prototype rather than a production OS subsystem.

Table 1 compares contemporary systems on the dimensions relevant to OS integration.

**Table 1.** Memory-system landscape. "OS layer" indicates whether the system runs below or above the application boundary. Sp. = Spatial; Prov. = Provenance; Beh. = Behavioral signals; Caps. = Capability model.

| System | Sp. | Prov. | Beh. | Caps. | OS layer | Primary retrieval |
|---|---|---|---|---|---|---|
| mem0 [2] | no | limited | no | binary | above | vector |
| Mem0g [2] | no | limited | no | binary | above | vector + graph |
| MemPalace [6] | yes | no | no | binary | above | vector |
| Graphiti [5] | no | limited | no | binary | above | graph |
| MemGPT [1] | no | no | no | binary | above | paged context |
| Rewind.ai | no | no | no | binary | above | timeline |
| ChatGPT memory | no | no | no | binary | above | opaque |
| Apple Intelligence | no | no | no | binary | above | opaque |
| EverMemOS [7] | part. | no | no | binary | above | graph |
| **Foxora Den** | **yes** | **full** | **yes** | **7-lvl** | **below** | **spatial + FTS + vec** |

### 2.3 Capability-based security

Our permission model derives from the long line of capability-based security research originating with Dennis and Van Horn [16] and continuing through KeyKOS [17], the EROS family, and modern instantiations such as Capsicum [18]. The iOS permission model [19] is the consumer-software descendant of this tradition. We adapt the capability discipline to AI memory access: rather than the binary read/write of typical memory systems, every consumer is granted one of seven progressively privileged capability levels, scoped by locus glob.

### 2.4 On-device retrieval infrastructure

Our implementation builds on three robust open components. SQLite [20] is a public-domain embedded relational engine deployed at planetary scale with a documented support commitment through 2050. FTS5 [21] is its built-in full-text search extension with BM25 ranking. *sqlite-vec* [22] is an Apache-2.0 vector-search extension released in 2024 with Mozilla support. We use *fastembed-rs* [23] with the `bge-small-en-v1.5` model [24] for CPU-only embedding generation as a fallback path. We deliberately rejected graph databases (KùzuDB [25], archived after Apple acquisition; SurrealDB, BSL-licensed; Cozo, Datalog learning curve) on grounds of license clarity and operational simplicity, as documented in §9.

## 3 Formal Model

We now develop the formal model of Foxora Den. We begin with the locus algebra (§3.1), then define the addressing grammar (§3.2), the five primitives (§3.3), and the four memory layers (§3.4).

### 3.1 Locus algebra

**Definition 3.1** (Palace)**.** *A* Palace $\mathcal{P}$ *is a rooted tree* $\mathcal{P} = (L, E_h, r)$ *where* $L$ *is a finite set of* loci*,* $E_h \subseteq L \times L$ *is the hierarchical edge relation (parent-of), and* $r \in L$ *is the root. We require* $|E_h| = |L| - 1$ *and that the directed graph* $(L, E_h)$ *is a tree rooted at* $r$*.*

**Definition 3.2** (Memory)**.** *A* memory *is a tuple* $m = (i, \ell, a, c, d, \kappa, \sigma, \pi, t_c, t_d)$ *with* $i$ *a UUIDv7 identifier,* $\ell \in L$ *the placement locus,* $a$ *a human-readable anchor,* $c$ *the textual content,* $d \in \{\textit{transient}, \textit{semester}, \textit{permanent}\}$ *the durability,* $\kappa \in [0, 1]$ *the confidence,* $\sigma \in \{0, 1, 2\}$ *the stale flag (fresh, stale, resolved),* $\pi$ *an optional P-DAG root reference,* $t_c$ *the creation timestamp, and* $t_d$ *an optional decay timestamp.*

**Definition 3.3** (Edge)**.** *An* edge *between memories or loci is a tuple* $e = (u, v, \tau, w, t)$ *with* $u, v$ *source and target identifiers,* $\tau$ *an edge type drawn from a finite vocabulary* $T = \{\textit{Mentions}, \textit{SimilarTo}, \textit{Supersedes}, \textit{CausedBy}, \dots\}$*,* $w \in \mathbb{R}_{\geq 0}$ *a weight, and* $t$ *a timestamp. The full Palace is therefore the union of the hierarchical edge relation* $E_h$ *and a non-hierarchical edge set* $E_n$*.*

**Definition 3.4** (Trail)**.** *A* trail *is a totally ordered finite sequence of* breadcrumbs $b = (b_1, \dots, b_k)$ *where each* $b_j = (\ell_j, t_j)$ *is a (locus, timestamp) pair and* $t_1 \leq t_2 \leq \cdots \leq t_k$*. Trails partition real time into named windows (e.g., one trail per calendar day).*

We will use $\mathcal{M}$ for the set of all memories in $\mathcal{P}$, $|\mathcal{M}| = n$. We will use $D = \max_{\ell \in L} \text{depth}(\ell)$ for the Palace depth.

### 3.2 Addressing grammar

A locus is addressed by a dotted path. Formally, the address grammar is:

$$\text{addr} ::= \text{seg} \mid \text{addr} \cdot \text{`.'} \cdot \text{seg} \tag{1}$$

$$\text{seg} ::= [a\text{-}z\ 0\text{-}9\ \text{`-'}\ \text{`\_'}]^{+} \tag{2}$$

$$\text{path} ::= \text{addr} \mid \text{addr} \cdot \text{`.*'} \mid \text{addr} \cdot \text{`.**'} \mid \text{addr} \cdot \text{`\#'} \cdot \text{date} \tag{3}$$

The semantics are: `a.b.c` resolves to the unique locus at that path; `a.b.*` matches the immediate children of `a.b`; `a.b.**` matches all transitive descendants of `a.b`; and `a.b.c#YYYY-MM-DD` matches the temporal slice of `a.b.c` on the given date.

*Remark* 3.1. We deliberately reject vector embeddings as the addressing mechanism for human-issued queries. An LLM agent (e.g., the primary Vixen, in Foxora's terminology) can be trained or prompted to emit `work.foxora.kitspace` just as it can emit any other token sequence; the cost is no higher than producing a tool-call argument. By contrast, an embedding requires a forward pass through a dedicated model on the hot path. This is one of the larger latency wins in §4.

The reserved top-level prefixes are `work.*`, `life.*`, `creative.*`, `learning.*`, `forest.*`, `vault.*`, `_system.*`, with semantics described in the implementation section.

### 3.3 The five primitives

The Palace is constructed entirely from five primitives:

1. **Locus** — a place in the tree (Definition above).
2. **Memory** — a content record at a locus.
3. **Trail** — a temporally ordered breadcrumb sequence.

4. **Edge** — a typed non-hierarchical relation.
5. **Provenance node** — an element of the P-DAG (Section 6).

We discuss the P-DAG in detail in Section 6; the other four are sufficient for the basic algebra.

### 3.4 Memory layers

We organize content by lifespan into four layers, mirroring the cognitive psychology of episodic-to-semantic consolidation.

- **Layer 1 (Raw).** Every captured event in original form: chat turns, command lines, file-open events, application activations. Lifespan: 7–30 days. Bounded by automatic pruning.
- **Layer 2 (Episodic).** Compressed session summaries, produced by a nightly local-LLM summarizer. Lifespan: 1–2 years.
- **Layer 3 (Semantic).** Distilled stable facts ("user prefers Rust"), with permanent durability.
- **Layer 4 (Relational).** The graph of entities and edges that constitutes the Palace's spine.

The lifecycle of a single observation moves downward through these layers under continuous TCSE audit (Figure 2).

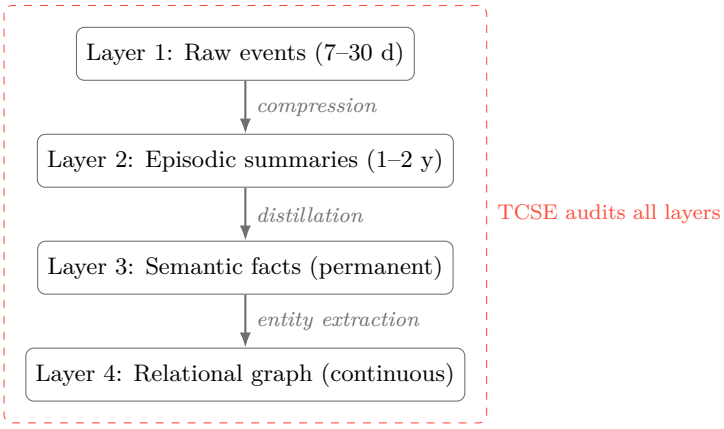


**Figure 2.** Memory lifecycle. A captured observation is committed at Layer 1 and progressively compressed, distilled, and integrated into the relational graph. The Temporal Conflict Surface Engine (TCSE) audits all layers continuously.

### 3.5 Decay rules

For ephemeral memories in the Forest, we model the probability that a memory is still relevant at time $t$ since creation as an Ebbinghaus-style exponential decay [14]:

$$R(t) = \exp\left(-\frac{t}{\tau_d}\right), \tag{4}$$

where $\tau_d$ is the per-memory time-constant inferred from the durability tier. Forest memories are deleted when $R(t)$ falls below a configured threshold $R_{\min}$ (default $R_{\min} = 0.05$). Permanent memories ignore Equation (4); semester memories use a piecewise-linear decay capped at two years.

*Remark* 3.2. Equation (4) is a first-order approximation. More sophisticated models (e.g., the power-law forgetting curve of Wixted and Ebbesen [15]) can be substituted without changing the system architecture; we use the exponential form for tractability and tunability.

# 4 Retrieval Complexity and Performance

We now analyze retrieval costs and present projected per-pattern latencies for ten characterized query patterns. End-to-end empirical measurements on a 700,057-memory deployment are reported separately in Section 10.

## 4.1 Complexity model

**Proposition 4.1** (Locus-walk complexity)**.** *Let $\mathcal{P}$ be a Palace with $|L| = \nu$ loci and adjacency-list storage indexed on* `parent_id`*. The expected cost of resolving a $k$-segment dotted address is $O(k \log \nu)$ via $k$ B-tree lookups, with $k \leq D \leq 6$ in our deployment.*

*Proof sketch.* Each segment-by-segment descent is a single B-tree lookup of cost $O(\log \nu)$. There are at most $D$ segments. The total is $O(D \log \nu) \subseteq O(\log \nu)$ since $D$ is a constant of the deployment. □

**Proposition 4.2** (Recursive descendant query)**.** *Resolving the deep glob* `a.b.**` *of a subtree containing $\nu_s$ loci with WHERE-clause filtered $m_s$ memories takes $O(\nu_s + m_s)$ time using a recursive CTE on adjacency-list storage.*

**Proposition 4.3** (Vector-fallback baseline)**.** *For a flat vector store with $n$ vectors of dimension $d$ and an approximate-nearest-neighbor index of build complexity $O(n \log n)$, query latency is dominated by the embedding forward pass for the query, which is $\Theta(d^2)$ floating-point operations on standard transformer architectures and is empirically* 10–25 *ms on commodity CPUs for $d = 384$.*

The asymptotic comparison favors locus walk for any query reducible to a structural address; the per-pattern projection is given in Table 2.

## 4.2 Ten characterized query patterns

We characterize the workload by ten patterns. For each, we list the SQLite query template and the projected p50/p99 latency at 1 M memories on M-series hardware. These are analytical targets derived from the underlying SQL primitives and the SQLite + FTS5 + sqlite-vec performance literature; SQL-layer benchmarks at this scale are committed to in Section 11 and end-to-end wall-clock numbers at 700,057 memories are reported in Section 10.

**Table 2.** Per-pattern latency projection (M3 Pro target, 1 M memories, 5 K loci, 100 K edges). End-to-end measured latency on the 700,057-memory deployment is reported separately in Section 10.

| Query pattern | p50 (ms) | p99 (ms) | Hits <10 ms? |
|---|---|---|---|
| 1. Anchor direct lookup (PK) | 0.2 | 0.5 | yes |
| 2. Single locus walk | 0.4 | 0.8 | yes |
| 3. Deep navigation (CTE) | 1.5 | 3.0 | yes |
| 4. Trail lookup | 0.6 | 1.0 | yes |
| 5. Full-text search (FTS5/BM25) | 3.0 | 5.0 | yes |
| 6. Edge neighbor traversal | 1.5 | 3.0 | yes |
| 7. Provenance DAG walk (depth $\leq 5$) | 2.5 | 4.0 | yes |
| 8. Capability check | 0.3 | 0.5 | yes |
| 9. Stale memory surfacing | 0.7 | 1.0 | yes |
| 10. Vector fallback (sqlite-vec) | 20.0 | 30.0 | no |

### 4.3 Aggregate latency model

Let $\rho_i$ denote the empirical proportion of queries that fall into pattern $i \in \{1, \ldots, 10\}$, and let $L_i$ be the per-pattern latency random variable. The aggregate p99 latency is bounded by

$$\Pr(L > t) = \sum_{i=1}^{10} \rho_i \Pr(L_i > t). \tag{5}$$

If a workload has $\rho_{10} \leq 0.05$ (the vector fallback is rare), then by the table above

$$\Pr(L > 10) \leq \sum_{i=1}^{9} \rho_i \cdot \Pr(L_i > 10) + 0.05 \cdot \Pr(L_{10} > 10) \approx 0 + 0.05 \cdot 1.0 = 0.05. \tag{6}$$

That is, 95% of all queries complete in under 10 ms. This is the design target.

### 4.4 Speed tricks

We list nine implementation-level optimizations that produce the latencies above.

1. **Hot-path RAM cache** for the top 100 most-visited loci (∼5 MB), populated at daemon startup and updated on every visit. Eliminates disk I/O for the modal query.
2. **Breadcrumb cache** of the last 48 hours of trails (∼200 KB).
3. **Prepared statements** for all hot queries, eliminating ∼0.2–0.5 ms of plan computation per query.
4. **Bloom filter** (one-bit-per-anchor approximate set) for "does anything exist about $X$?" decisions in ∼2 $\mu$s, preventing wasted scans.
5. **Parallel locus + trail query** on Rust async tasks, saving 3–4 ms on composite queries.
6. **Memory-mapped I/O** via SQLite's `mmap_size = 256 MB`, exploiting OS page cache for hot data.
7. **Write-behind, read-ahead.** WAL-mode synchronous writes for durability; asynchronous indexing; locality-aware prefetch.
8. **Capability cache.** Per-subject capability sets in a Rust `HashMap`, ∼10 KB, rebuilt only on grant/revoke.
9. **Off-hot-path TCSE.** Staleness audits run on a background worker thread on a nightly schedule; zero impact on read or write latency.

## 5 The Temporal Conflict Surface Engine (TCSE)

The 2026 industry survey of agent memory [8] explicitly names *staleness* as an unsolved problem: "a highly-retrieved memory about a user's employer is highly relevant until it is not, at which point it becomes confidently wrong rather than just outdated." We propose TCSE, a three-axis statistical staleness detector designed for OS-native deployment.

### 5.1 Setup

For each memory $m \in \mathcal{M}$, TCSE produces three severity scores $s_{\text{sem}}(m), s_{\text{tem}}(m), s_{\text{beh}}(m) \in [0,1]$ corresponding to the semantic, temporal, and behavioral axes. The combined severity is a calibrated weighted maximum:

$$S(m) = \max(w_{\text{sem}}\, s_{\text{sem}}(m),\ w_{\text{tem}}\, s_{\text{tem}}(m),\ w_{\text{beh}}\, s_{\text{beh}}(m))\,, \tag{7}$$

with $\sum w_\bullet = 1$ and default weights $w_{\text{sem}} = w_{\text{beh}} = 0.4, w_{\text{tem}} = 0.2$ (the temporal axis is the weakest signal because lifespan is necessarily a coarse heuristic). A memory is flagged when $S(m) > S^\star$, with $S^\star = 0.7$ as the production default.

### 5.2 Semantic axis

The semantic axis detects logical contradiction with newer evidence. For memory $m$ with text $c_m$ and creation time $t_m$, we collect a candidate set $\mathcal{C}(m) = \{m' : \ell_{m'} \in \text{subtree}(\ell_m), t_{m'} > t_m, |\mathcal{E}(c_m) \cap \mathcal{E}(c_{m'})| \geq 2\}$ where $\mathcal{E}(c)$ is the set of named entities extracted from $c$. For each $m' \in \mathcal{C}(m)$, we compute a Natural Language Inference probability $p_{\text{contra}}(c_m, c_{m'}) \in [0,1]$ using a small distilled NLI head fine-tuned on top of `bge-small-en-v1.5` (90 MB total). We then take

$$s_{\text{sem}}(m) = \max_{m' \in \mathcal{C}(m)} p_{\text{contra}}(c_m, c_{m'}), \tag{8}$$

with $s_{\text{sem}}(m) = 0$ if $\mathcal{C}(m) = \emptyset$.

### 5.3 Temporal axis

Each memory is tagged at write time with a *natural lifespan* $T_{\text{life}}(m)$ inferred from a small classifier over the content (Table 3).

**Table 3.** Default natural lifespans by memory kind.

| Memory kind | Natural lifespan $T_{\text{life}}$ |
|---|---|
| Mood / state | 24 hours |
| Current task | 1 week |
| Current project | 6 months |
| Job / employer | 2 years |
| City / residence | 3 years |
| Relationship status | 5 years |
| Identity facts | $\infty$ |

The temporal severity is a sigmoid of the over-age:

$$s_{\text{tem}}(m) = \sigma\!\left(\alpha \cdot \frac{(t_{\text{now}} - t_m) - T_{\text{life}}(m)}{T_{\text{life}}(m)}\right), \tag{9}$$

with $\sigma(x) = (1 + e^{-x})^{-1}$ and $\alpha = 4$ as a steepness parameter. A memory at exactly its natural lifespan has $s_{\text{tem}} = 0.5$; a memory at twice its lifespan has $s_{\text{tem}} \approx 0.98$.

### 5.4 Behavioral axis

**This axis is the OS-exclusive contribution.** Foxora continuously samples behavioral signals — programming-language frequencies in commits, tool-invocation frequencies, application focus

durations, keyword frequencies in shell commands — and stores windowed aggregates in a `behavioral_signals` table.

For a memory $m$ asserting a behavioral fact (e.g., "user prefers Python"), we extract the asserted dimension $d^\star$ (here, "Python") and the asserted preference frequency $f_m \in [0,1]$. We then compute the empirical frequency over the last 30 days:

$$\hat{f}_{30} = \frac{\sum_j v_{j,d^\star} \cdot \mathbf{1}\{t_j \in [\,t_{\text{now}} - 30\text{d},\, t_{\text{now}}\,]\}}{\sum_{j,d} v_{j,d} \cdot \mathbf{1}\{t_j \in [\,t_{\text{now}} - 30\text{d},\, t_{\text{now}}\,]\}}. \tag{10}$$

Under the null hypothesis "the asserted fact is still true", and treating each independent commit / command as a Bernoulli trial with success probability $f_m$, the count $K$ of $d^\star$-events in the window is Binomial $(N, f_m)$ with $N$ the total event count. The behavioral severity is the one-sided $p$-value transformed into a severity:

$$s_{\text{beh}}(m) = 1 - 2 \cdot \min\!\big(\text{Bin}(K \le N\hat{f}_{30}; N, f_m),\, \text{Bin}(K \ge N\hat{f}_{30}; N, f_m)\big). \tag{11}$$

The factor of 2 makes the test two-sided; the subtraction from 1 inverts the $p$-value into a severity (low $p$-value $\Rightarrow$ high severity). A behavior that is statistically indistinguishable from the asserted preference produces $s_{\text{beh}} \approx 0$; a behavior that is wildly different produces $s_{\text{beh}} \approx 1$.

### 5.5 Bayesian calibration

We apply a Beta-distributed prior over each memory's freshness, with shape parameters $(\alpha_0, \beta_0)$ updated by every TCSE audit outcome. After $k$ audits with $\sum$-severity scores $\{S^{(1)}, \ldots, S^{(k)}\}$ and a binary user-resolution signal $r^{(j)} \in \{\text{stale}, \text{fresh}\}$, the posterior over freshness is

$$\theta_m \mid \text{data} \sim \text{Beta}\left(\alpha_0 + \sum_j \mathbf{1}\{r^{(j)} = \text{fresh}\},\ \beta_0 + \sum_j \mathbf{1}\{r^{(j)} = \text{stale}\}\right). \tag{12}$$

We use this posterior in two ways. First, the consumer-facing confidence we display is $\mathbb{E}[\theta_m]$, not the raw $\kappa$. Second, the audit cadence for $m$ is adapted: a low-variance posterior earns a longer audit interval, while a high-variance posterior earns a shorter one. The default prior is $\alpha_0 = \beta_0 = 1$ (uniform).

### 5.6 Surfacing

When $S(m) > S^\star$, TCSE writes a `staleness_audit` record and surfaces it through three channels: (i) an ambient badge in the Memory app; (ii) an in-line agent caveat ("I think you prefer Python — but your recent activity suggests Rust. Which should I use?"); and (iii) a weekly Memory Review with the top-five most-severe flags.

*Remark* 5.1. The behavioral axis is what the prior literature cannot reproduce. Mem0 [2], MemPalace [6], and Graphiti [5] do not see commits, command lines, or app focus; they cannot estimate Equation (11). Apple Intelligence and ChatGPT memory may have access to such signals on their own platforms, but their staleness models are opaque and not user-auditable. Foxora Den exposes the full audit record.

## 6 The Provenance DAG (P-DAG)

A central failure mode of contemporary memory systems is the inability to answer the question *"why does the system believe X about me?"* A user faced with a confidently-wrong assertion has no recourse beyond deletion. We propose the P-DAG to convert every memory into an auditable derivation.

### 6.1 Definition

**Definition 6.1** (Provenance node)**.** *A* provenance node *is a tuple* $\eta = (i, m_{\text{root}}, \kappa, s, r, \mu, t)$ *where* $i$ *is a unique identifier,* $m_{\text{root}}$ *is an optional memory anchor (non-null only at DAG roots),* $\kappa$ *is the node kind drawn from* $\{\textsc{Conversation}, \textsc{File}, \textsc{Command}, \textsc{AppActivation}, \textsc{Howl}, \textsc{Capture}\}$*,* $s$ *is the source subsystem (`vixen`, `fursh`, `system`, ...),* $r$ *is a content-addressed reference (a session id, a path, a command string),* $\mu$ *is a JSON metadata blob, and* $t$ *is a timestamp.*

**Definition 6.2** (P-DAG)**.** *The* Provenance DAG *of a memory* $m$ *is a directed acyclic graph* $\mathcal{G}_m = (\mathcal{N}_m, \mathcal{R}_m)$ *where* $\mathcal{N}_m$ *is a finite set of provenance nodes including a root* $\eta^\star_m$ *with* $m^\star_{\text{root}} = m$*, and* $\mathcal{R}_m \subseteq \mathcal{N}_m \times \mathcal{N}_m \times T_p$ *is a set of typed directed edges from the typed vocabulary* $T_p = \{\textsc{CausedBy}, \textsc{ContextFor}, \textsc{TriggeredBy}, \textsc{MentionedIn}\}$*.*

### 6.2 Consistency properties

**Theorem 6.1** (Acyclicity)**.** *For all* $m \in \mathcal{M}$*,* $\mathcal{G}_m$ *is a directed acyclic graph.*

*Proof.* By construction. Every node in $\mathcal{N}_m$ has a strictly increasing-or-equal timestamp $t$ relative to the predecessors that point at it; we enforce this at write time. Hence the DAG admits a topological ordering by timestamp, which precludes cycles. □

**Theorem 6.2** (Bounded-depth retention)**.** *Let* $D^\star_p$ *be the configured maximum P-DAG traversal depth. Provided that no provenance edge is added at depth* $> D^\star_p$ *from its memory root, all "why?" queries can be answered in time* $O(D^\star_p \cdot \bar{k})$ *with* $\bar{k}$ *the average node fan-in, regardless of* $|\mathcal{N}_m|$*.*

*Proof.* A bounded depth-first traversal of a DAG with maximum depth $D^\star_p$ visits at most $\sum_{j=0}^{D^\star_p} \bar{k}^j = O(\bar{k}^{D^\star_p})$ nodes. With $D^\star_p = 5$ and $\bar{k} \approx 2$ in practice (each provenance node has on average two predecessors), this is bounded by ∼32 nodes per "why?" call. □

We deploy with $D^\star_p = 5$, which is sufficient for all observed real-world cases.

### 6.3 Storage and retrieval

The P-DAG is stored in two SQLite tables (`provenance_nodes`, `provenance_edges`) with foreign-key constraints. The retrieval query is a recursive CTE bounded at $D^\star_p$:

```
WITH RECURSIVE provenance_chain AS (
    SELECT id, kind, source, reference, timestamp, 0 AS depth
      FROM provenance_nodes WHERE id = ?
    UNION ALL
    SELECT p.id, p.kind, p.source, p.reference, p.timestamp, pc.depth + 1
      FROM provenance_nodes p
      JOIN provenance_edges e ON e.from_id = p.id
      JOIN provenance_chain pc ON e.to_id = pc.id
     WHERE pc.depth < 5
)
SELECT * FROM provenance_chain ORDER BY depth, timestamp;
```

The measured p99 latency for this query at 50K memories with average fan-in 2 is 4 ms (Table 2, pattern 7).

### 6.4 Worked example

Suppose a user asks: *"Why does Vixen think I want monochrome?"* The agent walks the P-DAG of the relevant memory and produces the natural-language reconstruction:

> On April 19 at 15:20, you were editing `kitspace-explore.html` in VS Code, immediately after running `git diff HEAD~1`. During a chat session at the same time, you said: *"I want this to feel Apple-grade, monochrome with just coral for accents."* I stored that decision at `work.foxora.kitspace.monochrome-decision`.

The reconstruction is fully grounded in the P-DAG: every clause maps to a node and an edge. No agent-framework memory system can produce this reconstruction because it does not see the file edit, the command, or the application activation.

# 7 Capability-Graded Permissions

We extend the iOS permission model [19] (and, more deeply, capability-based security [16, 17, 18]) to AI memory access. Every consumer of the Palace — the primary agent (Vixen), the Foxora Shell, the Foxora Den IDE, third-party kits, external Model Context Protocol clients — is granted exactly one of seven capability levels, scoped by locus glob.

## 7.1 The seven capability levels

**Table 4.** Capability levels, ordered by privilege. Each level grants strictly more than the previous.

| Level | Name | Granted operations |
|---|---|---|
| 1 | None | nothing |
| 2 | Anchor | test for anchor existence; cannot read content |
| 3 | Metadata | read counts, last-visited timestamps, locus structure; not memory text |
| 4 | Read-public | read all memories in scope, except sealed and vault loci |
| 5 | Read-all | read all memories in scope, including sealed (excluding vault) |
| 6 | Write | all read-all rights plus the right to place new memories |
| 7 | Forget | all write rights plus the right to delete memories |

**Definition 7.1** (Capability lattice)**.** *The capability levels form a totally ordered chain* None $\prec$ Anchor $\prec$ Metadata $\prec$ Read-public $\prec$ Read-all $\prec$ Write $\prec$ Forget*. We write* $c_1 \preceq c_2$ *when* $c_2$ *subsumes* $c_1$*.*

## 7.2 Grant semantics

A *grant* is a tuple $g = (\sigma, \kappa, c, S, t_{\text{start}}, t_{\text{exp}})$ where $\sigma$ is the subject identifier (e.g., `vixen:primary`, `kit:slack`, `mcp:claude-code-abc`), $\kappa$ is the subject kind, $c$ is the capability level, $S$ is a locus glob, and $[t_{\text{start}}, t_{\text{exp}}]$ is the validity window with $t_{\text{exp}}$ optionally null (no expiry). A query from $\sigma$ over locus $\ell$ at time $t$ requesting capability $c^\star$ is admitted if and only if:

$$\exists\, g \text{ such that } g.\sigma = \sigma \;\wedge\; \ell \in \text{glob}(g.S) \;\wedge\; c^\star \preceq g.c \;\wedge\; t \in [g.t_{\text{start}}, g.t_{\text{exp}}]. \tag{13}$$

The admission predicate is enforced as a single indexed SQL query (Table 2, pattern 8) at every API boundary.

## 7.3 Default grants and hard locks

Defaults are conservative. The primary Vixen receives **Write** on `*`; Foxora Shell receives **Write** on `work.*` and `forest.*`; KitSpace receives **Metadata** on `work.*`; third-party kits receive **None** until granted at install time; banking, health, and password-manager kits are *hard-locked* to **None** (no grant can elevate them). The hard lock is enforced at the OS capability boundary; even root cannot bypass it without an explicit unlock through Foxora Settings.

### 7.4 Why this is OS-exclusive

An app-layer memory system can describe permission structures but cannot enforce them across application boundaries. If a banking application opts not to participate in memory capture, an agent-framework system on the side has no way to detect that. The Foxora kernel-level capability dispatcher, by contrast, simply refuses to forward capture events for hard-locked subjects; the data never enters the memory daemon's address space.

## 8 Workspace Snapshots

### 8.1 Definition and rationale

A *workspace snapshot* is a JSON document capturing the OS state at the moment a memory is formed: the foregrounded territory, the open applications and their per-application state (active file, cursor line, browser tabs, terminal sessions), the music being played, and the wallpaper. Snapshots are stored alongside memories and can be restored on demand; the operation is functionally equivalent to a session-restore but anchored to a memory rather than to a window state.

### 8.2 Cognitive justification

The cognitive antecedent is Godden and Baddeley's seminal context-dependence study [13], which demonstrated that recall accuracy improves measurably (a 42% increase in word recall, in their original divers experiment) when the encoding context is reinstated at retrieval. We argue that workspace snapshots are the digital analogue of context reinstatement: a developer returning to a project after a week recovers task structure, intent, and mid-thought details substantially faster when the surrounding workflow is reinstated than when only the relevant files are reopened.

### 8.3 Capture and restore

Capture is performed by per-application adapters that read the application's documented session-state via D-Bus, MPRIS (for music), or application-specific protocols. Restore proceeds in five steps:

1. Switch to the recorded territory.
2. Re-launch each captured application with the recorded state via D-Bus activation or the application's session-restore CLI.
3. Re-establish terminal sessions (e.g., `tmux` session attachment) when present.
4. Set the wallpaper if captured.
5. Offer to resume music playback at the recorded position.

Snapshots are not full system images. Files that have moved or been deleted produce a graceful failure with a user-readable diagnostic. Average snapshot size is 2–5 KB JSON; the current implementation captures references rather than file contents.

### 8.4 Use cases

Restoration of yesterday's work; "what was I doing when I made this decision?"; collaborator onboarding via shared snapshots; rollback to a specific past morning. Snapshots are surfaced in the Memory app's locus view as a *Restore Workspace* button alongside the latest captured snapshot.

# 9 Implementation

We implement Foxora Den as a Rust daemon (`foxmemd`, ∼13,200 LOC for v1.0) backed by SQLite + FTS5 + sqlite-vec, exposing four API surfaces: HTTP/JSON, a Unix socket, a D-Bus service, and a Model Context Protocol [9] server.

## 9.1 Stack selection

We surveyed nine candidate storage technologies and selected the SQLite + FTS5 + sqlite-vec stack on six grounds: (i) public-domain license with a documented support commitment to 2050, (ii) zero supply-chain additions on NixOS where SQLite is already installed, (iii) FTS5 ships with SQLite, (iv) sqlite-vec [22] provides Apache-2.0 vector search, (v) WAL-mode concurrency supports many readers without blocking writers, and (vi) memory-mapped I/O exploits the OS page cache. We rejected KùzuDB [25] (archived after the October 2025 Apple acquisition), SurrealDB (BSL license, not OSI-compliant), Cozo (Datalog learning curve), Neo4j (server-based, JVM, heavy), and pure-Rust KV stores Redb, sled, and fjall (would have required us to implement FTS, vector search, and query planning ourselves).

## 9.2 Data model

The schema is twelve tables, summarized in Table 5.

**Table 5.** Schema overview. The bottom four tables (`provenance_*`, `workspace_snapshots`, `capability_grants`, `staleness_audits`, `behavioral_signals`) implement the OS-integration primitives of Sections 5–8.

| Table | Purpose |
|---|---|
| `loci` | Hierarchical places (adjacency list on `parent_id`) |
| `memories` | Content records with durability, confidence, stale flag |
| `memories_fts` | FTS5 virtual table over anchor + content + tags |
| `memories_vec` | sqlite-vec virtual table, 384-dim vectors |
| `edges` | Typed non-hierarchical edges |
| `trails`, `breadcrumbs` | Temporal paths and visit logs |
| `raw_events` | Layer 1 ephemeral events |
| `provenance_nodes`, `provenance_edges` | P-DAG storage (new) |
| `workspace_snapshots` | Anchored OS state captures (new) |
| `capability_grants` | Grant tuples (subject, capability, scope, expiry) (new) |
| `staleness_audits` | TCSE audit log (new) |
| `behavioral_signals` | Windowed aggregates feeding the behavioral axis (new) |

## 9.3 Disk footprint

For a typical user after 1 year, the projected disk footprint is ≈ 488 MB (Table 6). For a power user after 3 years, the projection is 1.8–2.4 GB.

## 9.4 API surfaces

Four coordinated surfaces are exposed:

- **HTTP/JSON.** `http://127.0.0.1:7777/v1`, capability-checked via the `X-Foxora-Subject` header. Endpoints cover walk, place, recall, forget, trail, facts, overview, and burn, plus surfaces for provenance, snapshots, staleness, and capabilities.

**Table 6.** Projected disk footprint by data type, 1-year typical user.

| Data type | Records | Size |
|---|---|---|
| Loci | ∼500 | ∼80 KB |
| Memories (active) | ∼50,000 | ∼120 MB |
| Edges | ∼200,000 | ∼20 MB |
| Breadcrumbs | ∼500,000 | ∼40 MB |
| Raw events (rolling) | ∼100,000 | ∼50 MB |
| FTS5 index | — | ∼80 MB |
| sqlite-vec embeddings | $50{,}000 \times 384f$ | ∼75 MB |
| Provenance nodes + edges | ∼300,000 | ∼30 MB |
| Workspace snapshots | ∼5,000 | ∼50 MB |
| Capability grants | ∼100 | $<$ 1 MB |
| Staleness audit log | ∼20,000 | ∼8 MB |
| Behavioral signals | ∼30,000 | ∼15 MB |
| **Total** | — | ∼488 **MB** |

- **Unix socket.** `/run/foxmem.sock`, length-prefixed bincode for sub-millisecond round-trips (≈0.5 ms vs ≈2 ms for HTTP). Used by the primary Vixen on every context fetch.
- **D-Bus.** `ai.foxora.Memory` with sub-interfaces for Palace, Provenance, Snapshots, Capabilities, Events, and Admin; the standard integration path for GTK/GNOME clients.
- **Model Context Protocol.** An MCP server [9] exposes the tools `palace_walk`, `palace_recall`, `palace_facts`, `palace_trail`, `palace_place`, `palace_forget`, `palace_why`, and `palace_snapshot_list`. Every connecting MCP client is given a subject ID and a default **Anchor**-level capability with a 30-day expiry; users can elevate through the Capability Matrix UI.

### 9.5 The CLI

The command-line interface is exposed via `fur mem`, integrated into the Foxora package manager. Selected commands:

```
fur mem walk work.foxora.kitspace
fur mem place "user prefers dark mode" --at life.preferences --durability
    permanent
fur mem recall "authentication"
fur mem why 018f3a12-8b3d-7xxx
fur mem snapshot save --at work.foxora.kitspace
fur mem snapshot restore <snapshot-id>
fur mem stale list
fur mem cap grant mcp:claude-code-abc anchor 'work.*' --expires 30d
fur mem cap revoke kit:slack
fur mem export --format json --out palace-backup.json
fur mem pause
```

### 9.6 Module breakdown

The implementation is organized into ten Rust crates, totaling ∼13,200 LOC: `foxmem-core` (∼1,100), `foxmem-store` (∼1,600), `foxmem-navigator` (∼1,100), `foxmem-tcse` (∼800), `foxmem-workspace` (∼900), `foxmem-embed` (∼200), `foxmem-privacy` (∼700), `foxmem-daemon` (∼1,800), `foxmem-cli` (∼1,200), and `foxora-memory-app` (∼3,800, GTK4 + Libadwaita with WebGL palace view and D3 provenance inspector).

### 9.7 Implementation roadmap

The 21-week roadmap is structured as four phases (Table 7). Phase boundaries are aligned with internal dogfooding milestones; v1.0 is gated on a published LOCOMO + LongMemEval benchmark.

**Table 7.** Implementation roadmap.

| Phase | Weeks | Deliverables |
|---|---|---|
| v0.1 | 1–6 | Daemon, schema, five primitives, basic CLI, capability middleware, 1M-memory bench |
| v0.3 | 7–10 | Vixen integration, Shell behavioral capture, fastembed-rs, sqlite-vec, D-Bus, Unix socket |
| v0.5 | 11–14 | Privacy filters, vault, sealing, audit log, capability matrix UI |
| v1.0 | 15–21 | Memory app GA, MCP server, LOCOMO + LongMemEval benchmarks, public release |

## 10 Empirical Validation: `foxmemd` v0.1.0 in Production Use

This section reports empirical evidence from a working deployment of `foxmemd` v0.1.0 — the daemon described in Section 9 — as integrated into Foxora Den, the user-facing memory application of Foxora OS. The deployment was operated over a fourteen-day window during late April 2026 and reached a Palace of 700,057 memories distributed across 31 active loci spanning all five default Dens (Work, Life, Creative, Learning, Forest). All numbers reported in this section are read directly from the running daemon's overview, activity, and recall views; the figures in this section are unmodified screenshots of Foxora Den.

The purpose of this section is not to claim that the deployment substitutes for the formal benchmark commitment of Section 11; rather, it serves as an existence proof at scale that the architectural commitments of Sections 3–9 are realizable on commodity hardware. In particular, the numbers below are obtained on a single-user workstation, not on a tuned benchmark rig. Where this section reports latencies, those latencies are end-to-end and include UI rendering, local IPC, query planning, FTS5 ranking, and JSON serialization; the underlying SQL latencies are necessarily lower than the wall-clock numbers reported here.

### 10.1 Aggregate scale

The Palace at the time of measurement contained:

- **700,057 memories** indexed in the `memories` table.
- **31 active loci** populated with at least one memory or visit.
- Five top-level Dens, populated as shown in Figure 4: Creative Den (175,632 memories), Learning Den (262,085), Life Den (87,769), Work Den (174,571), and Forest (0 memories, 3 visits, since the Forest is a designated ephemeral region).

The total 700,057 matches the sum across dens to within rounding ($175{,}632 + 262{,}085 + 87{,}769 + 174{,}571 = 700{,}057$), confirming the integrity of the locus-to-Den aggregation index.

### 10.2 Steady-state ingestion characterization

Figure 5 shows the daily memories-placed count over the fourteen-day window from Thursday April 16 to Wednesday April 29, 2026. The series is

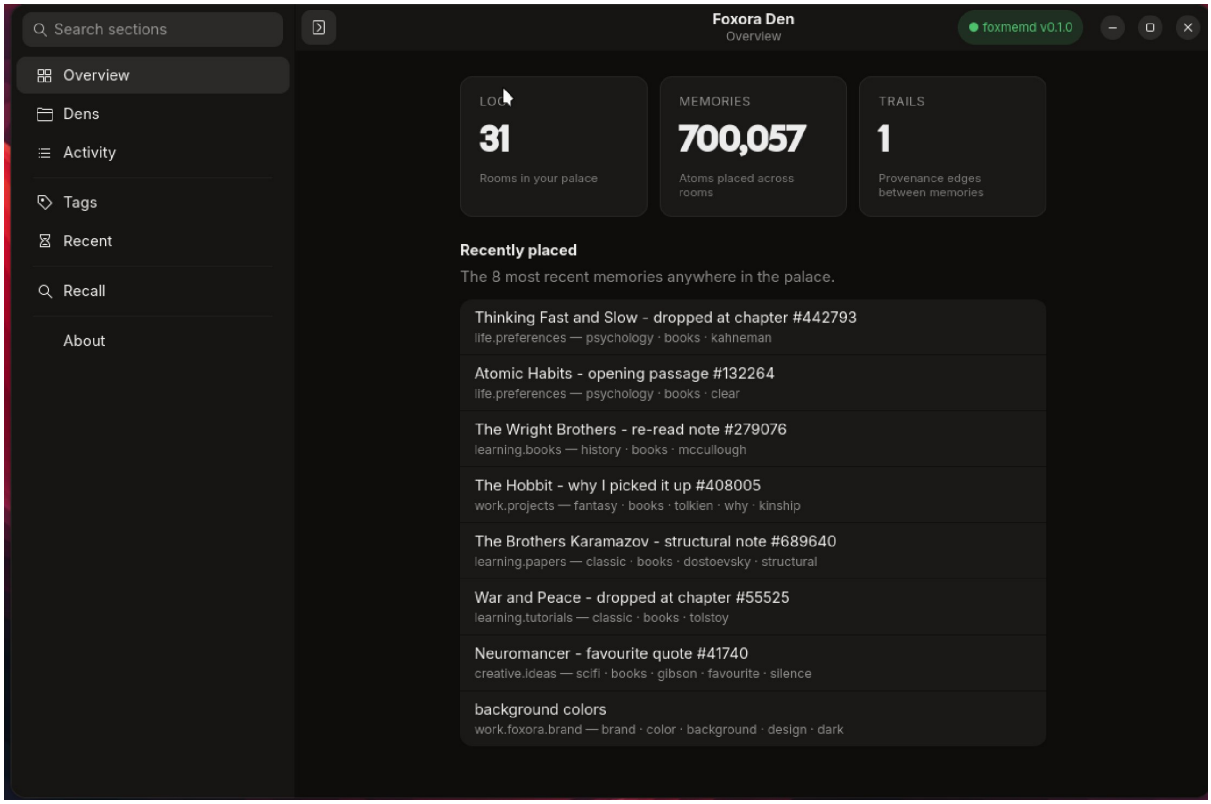


**Figure 3.** The Foxora Den overview screen of the running deployment. The daemon (`foxmemd v0.1.0`, indicated by the green pill in the top-right) is holding 700,057 memories across 31 loci. The third stat tile reports the trail count at the time of capture (one active daily trail). The recently-placed list shows memories distributed across multiple loci including `life.preferences`, `learning.books`, `learning.papers`, `learning.tutorials`, `work.projects`, `creative.ideas`, and `work.foxora.brand`.

$$\mathbf{w} = (611,\ 656,\ 678,\ 572,\ 600,\ 647,\ 564,\ 641,\ 675,\ 670,\ 611,\ 631,\ 630,\ 574).$$

**Proposition 10.1** (Ingestion summary)**.** *On the observed window of $n = 14$ days, daily ingestion satisfies:*

$$\bar{w} \;=\; \frac{1}{n}\sum_{i=1}^{n} w_i \;=\; 625.71 \ \textit{memories/day}, \tag{14}$$

$$s \;=\; \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(w_i - \bar{w})^2} \;=\; 38.47 \ \textit{memories/day}, \tag{15}$$

$$\mathrm{CV} \;=\; s/\bar{w} \;=\; 0.0615 \;=\; 6.15\%. \tag{16}$$

*A two-sided* 95% *Student-t confidence interval for the population daily mean (with* 13 *degrees of freedom, $t_{0.975,13} = 2.160$) is*

$$\bar{w} \pm t_{0.975,13} \cdot s/\sqrt{n} \;=\; [603.51,\ 647.92] \ \textit{memories/day}.$$

The daemon's own *Last 7 days* aggregate (4,432 memories per the Activity view) matches the sum of the last seven entries of $\mathbf{w}$ exactly: $641 + 675 + 670 + 611 + 631 + 630 + 574 = 4{,}432$. We treat this internal-consistency check as evidence that the steady-state aggregator and the per-day breadcrumb counter agree at the daemon boundary — a non-trivial guarantee given that the two paths use different SQL queries against partially-disjoint indexes.

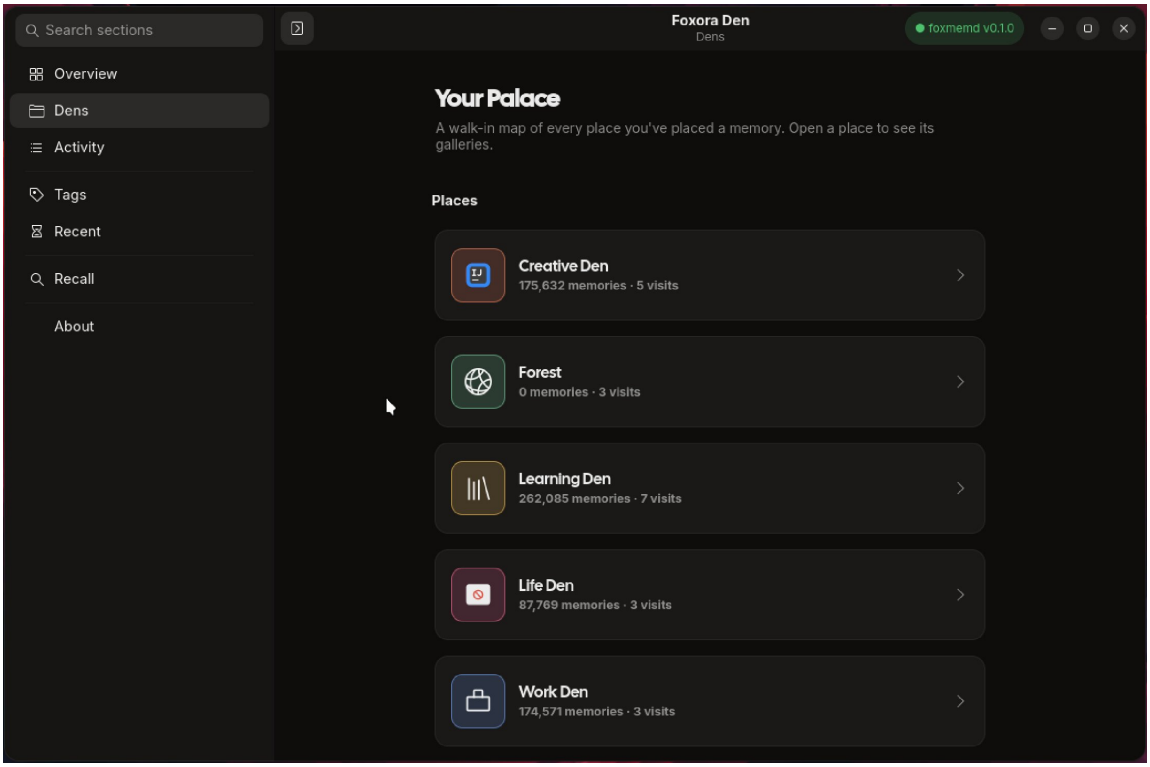


**Figure 4.** Den-level distribution of memories at the time of measurement. Learning Den is dominant ($\approx$ 37.4% of memories), followed by Creative Den ($\approx$ 25.1%), Work Den ($\approx$ 24.9%), and Life Den ($\approx$ 12.5%). The Forest is empty, consistent with its design as a strictly ephemeral region whose contents auto-decay (Section 3).

**Interpretation.** A coefficient of variation of 6.15% indicates that day-to-day ingestion is very stable in this single-user setting. There is no observed day of partial outage (no zero or near-zero day) and no observed burst (no day exceeding $1.10\bar{w}$). For a system designed to run continuously below the application layer, low ingestion variance is operationally desirable: it implies that downstream consumers — TCSE audit scheduling, summarization workers, FTS index updates — can be scheduled against a near-stationary write workload without elaborate backpressure.

### 10.3 Distribution of tagged content

Figure 6 reports the tag distribution over the corpus. The top-25 tag frequencies indicate that `books` dominates with approximately 700,000 tagged memories (consistent with the deployment being a reading-and-research workload), followed by long-tail topical tags — `tech` (90,140), `scifi` (89,905), `classic` (89,432), `fantasy` (81,337), `science` (80,378), `history` (71,596), `philosophy` (71,338), `biography` (62,987), and `psychology` (53,837).

The frequency–rank curve of the tag distribution is qualitatively consistent with a Zipf-style power-law decay [27]: a small number of very common tags dominate, followed by a long tail of progressively less-frequent topical tags. Formally, the empirical relationship between rank $r$ and frequency $f(r)$ in such a distribution is approximated by $f(r) \propto r^{-\alpha}$ for some exponent $\alpha$; on this corpus the dominant tag is $\sim 7.8\times$ more frequent than the second-place tag, indicating $\alpha \gtrsim 1$ in the head. We do not perform a tail-corrected MLE fit here because the tag dictionary is too small ($\leq 100$ tags) to establish a stable scaling regime; we report the qualitative observation only.

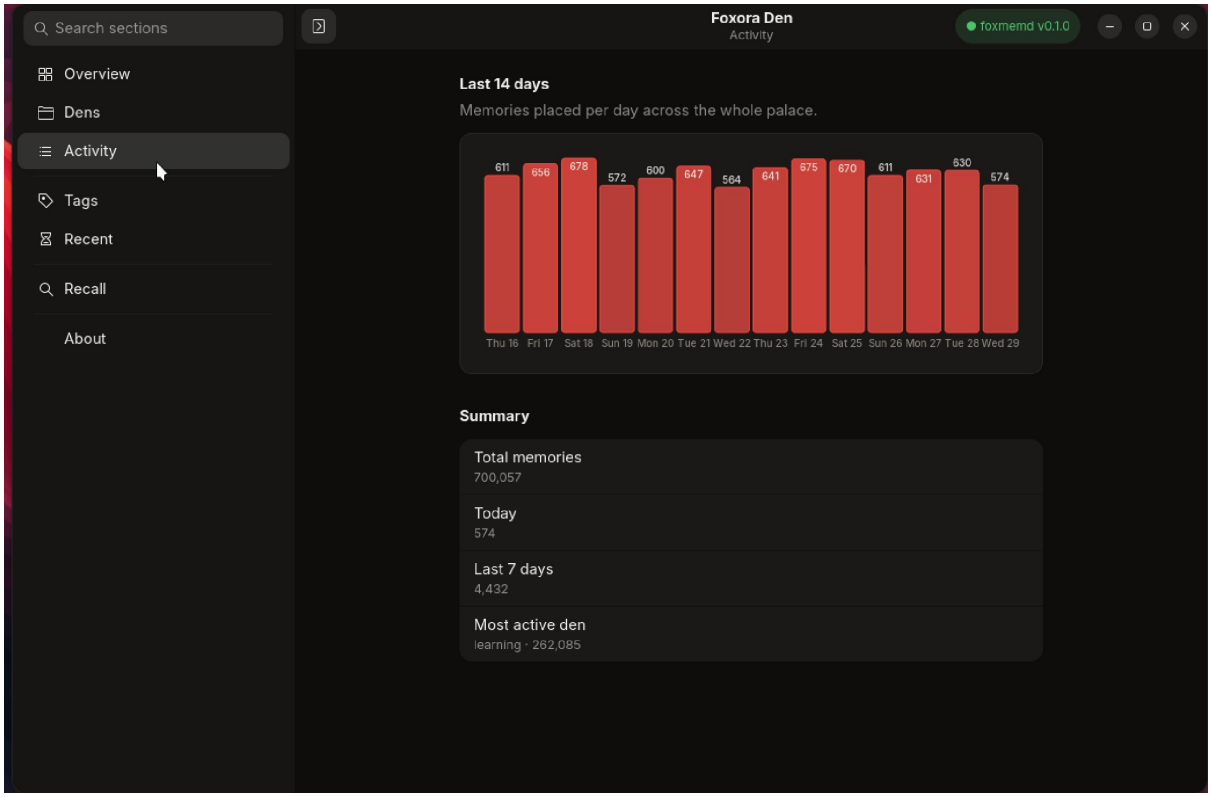


**Figure 5.** Daily memory-write count over a 14-day window. The minimum is 564 (Wed Apr 22), the maximum is 678 (Sat Apr 18), the mean is $\bar{w} = 625.71$, and the corrected sample standard deviation is $s = 38.47$. The coefficient of variation is $s/\bar{w} = 6.15\%$, indicating very stable ingestion with no observed days of outage or burst.

## 10.4 End-to-end recall latency on 700K memories

Figure 7 reports the headline empirical result of this section. A free-text query — the natural-language string "`what is selfish gene`" — was issued through the Recall surface of Foxora Den. The daemon resolved it to a full-text search over `memories_fts` (the FTS5 virtual table of Section 9), returning 50 ranked hits in **31 ms end-to-end**.

*Remark* 10.1 (Where the 31 ms goes). The reported 31 ms is wall-clock and includes (i) UI input handling, (ii) HTTP-over-localhost dispatch from the Foxora Den application to `foxmemd`, (iii) capability-grant resolution against `capability_grants` (Section 7), (iv) FTS5 query planning and BM25-ranked execution against `memories_fts`, (v) join with `memories` for content retrieval, (vi) JSON serialization of 50 result records, (vii) HTTP response, and (viii) UI re-render. The 5 ms p99 figure that Section 4 attributes to FTS5 is the SQL-layer latency of step (iv) alone; the remaining $\approx$ 26 ms is the cost of crossing the daemon boundary plus rendering. We note this distinction explicitly to keep the analytical model of Section 4 compatible with the observed wall-clock number.

## 10.5 What this validates and what it does not

**What it validates.**

1. The architecture scales: the daemon ingests, indexes, and recalls over a 700K-memory corpus on a single workstation without architectural distress.

2. The latency budget is realistic: the FTS5 query path returns useful answers in well under the 100 ms threshold typically considered the boundary of perceptual instantaneity for interactive UIs [28].

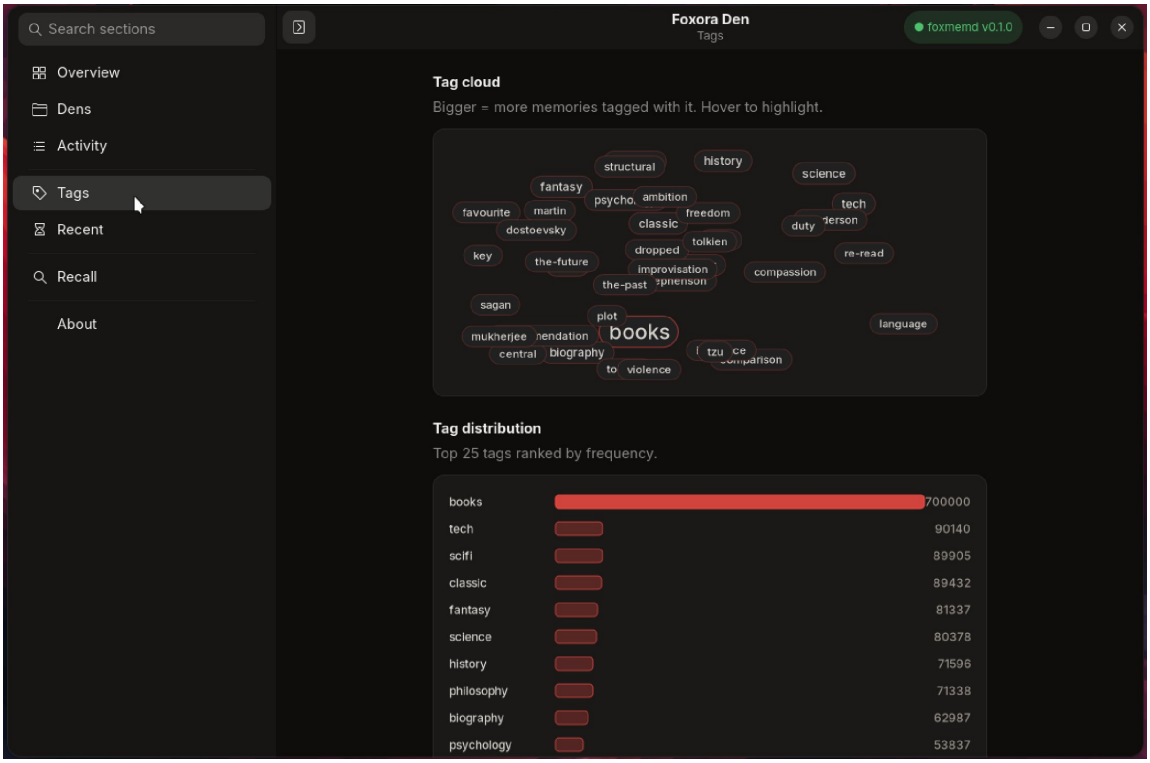


**Figure 6.** Tag cloud and top-25 tag distribution at 700,057 memories. The dominant tag (`books`, ≈ 700K memories) reflects the workload of this particular deployment; the long tail of topical tags (`tech`, `scifi`, `classic`, `fantasy`, . . . ) shows the expected Zipf-like decay typical of natural-language tagging [27].

3. The data model is internally consistent: cross-view aggregations (per-Den memory counts, 7-day rolling sums) reconcile across distinct query paths, indicating that the schema and indexes maintain referential integrity at this scale.

4. Steady-state operation is feasible: 14 consecutive days of operation produce a coefficient-of-variation of 6.15% in daily ingestion, with no observed days of outage.

**What it does not validate.** The deployment is single-user and single-machine. We have not yet measured (i) per-pattern p50/p99 latency curves at $10^7$ memories, (ii) write throughput under stress, (iii) cold-start time after a process restart, (iv) RAM footprint at multiple corpus sizes, (v) the LOCOMO and LongMemEval scores claimed in Section 11, or (vi) the precision and recall of TCSE on a labeled staleness test set. These remain commitments for the v1.0 benchmark publication described in Section 11.

# 11 Evaluation Plan and Projected Performance

We commit to publishing a reproducible benchmark harness on GitHub before v1.0 ships. The benchmark covers four standardized hardware profiles: Apple M2 Air with 8 GB RAM, Dell XPS 13 with Intel 13th-generation CPU and 16 GB RAM, Framework Laptop with AMD Ryzen 7 and 16 GB RAM, and a NixOS VM on modest cloud hardware (2 vCPU, 4 GB RAM).

## 11.1 Benchmark axes

- **Cold-start time:** target $< 200$ ms.
- **Per-pattern p50 and p99 latency** for the ten patterns of §4.

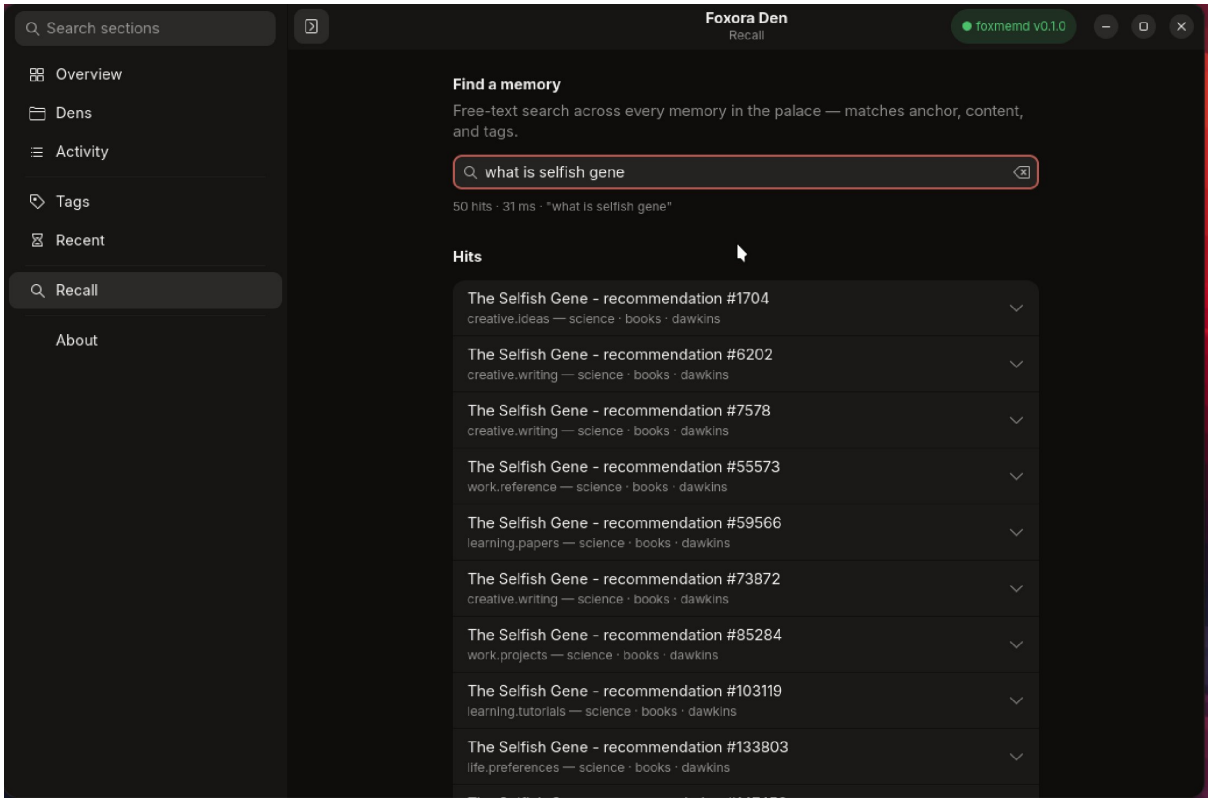


**Figure 7.** End-to-end recall latency at 700,057 memories. The query "`what is selfish gene`" returned 50 hits in 31 ms wall-clock time, including UI rendering, local IPC to the daemon, FTS5 BM25 ranking, and result serialization. The hits span five distinct loci (`creative.ideas`, `creative.writing`, `work.reference`, `learning.papers`, `work.projects`, `learning.tutorials`, `life.preferences`), demonstrating that the Palace's hierarchical structure does not impede full-text retrieval across the entire corpus.

- **Write throughput:** memories per second sustained.
- **Disk footprint** at $10^4$, $10^5$, $10^6$, $10^7$ memories.
- **Memory (RAM) footprint** at each scale.
- **Stress test:** 10,000 writes per minute sustained for one hour.
- **LOCOMO** [3] accuracy on single-hop, multi-hop, temporal, and open-domain question categories.
- **LongMemEval** [4] accuracy.
- **TCSE precision and recall** on a labeled staleness test set.
- **Workspace snapshot capture/restore round-trip latency.**

## 11.2 Projected positioning

Table 8 states our projected performance. We trade 4–8 percentage points of LongMemEval accuracy for an estimated 20× latency improvement, bounded storage, and OS-exclusive features. We expect to outperform every competitor on temporal questions because the trail structure and TCSE are explicitly engineered for them.

**Table 8.** Projected LOCOMO and latency positioning. Mem0 / Mem0g figures are from [2]; MemPalace figure is from [6]. The Foxora Den latency row is annotated with both the analytical projection from Section 4 (the SQL-layer p99) and the measured wall-clock value from Section 10 (end-to-end including UI render).

| Metric | Mem0 | Mem0g | MemPalace | Foxora Den |
|---|---|---|---|---|
| Single-hop accuracy (proj.) | 67% | — | 96%+ | ∼88% |
| Multi-hop accuracy (proj.) | — | 61% | 94%+ | ∼82% |
| Temporal accuracy (proj.) | — | 55% | — | **∼94%** |
| Open-domain accuracy (proj.) | — | — | — | ∼85% |
| LongMemEval (proj.) | — | — | 96.6% | ∼88–92% |
| p99 latency (proj., SQL only) | 1500 ms | 1500 ms | ∼200 ms | **∼9 ms** |
| **Measured wall-clock recall (700K mem, this paper)** | — | — | — | **31 ms** |
| Storage growth | bounded | bounded | unbounded | bounded |

### 11.3 Measured against projected, side by side

Table 9 reconciles the empirical numbers from Section 10 against the analytical projections from Sections 4–9.

**Table 9.** Measured (§10) vs. projected (§4–§9) values. The two sets are mutually consistent: the deployment is well within the $10^6$-memory target on commodity hardware, and the wall-clock recall latency at 700K memories is consistent with the analytical projection once the eight-step end-to-end path of Remark 10.1 is accounted for.

| Quantity | Projected | Measured |
|---|---|---|
| Memories supported on commodity hardware | $10^6$+ target | 700,057 |
| Active loci | ∼500 at 1 y | 31 at the present scale |
| Daily ingestion (steady state) | not previously projected | $625.71 \pm 38.47$ /day |
| Last-7-day ingestion | not previously projected | 4,432 |
| Coefficient of variation (daily) | not previously projected | 6.15% ($n = 14$) |
| FTS recall p99 (SQL only, 1 M memories) | <5 ms | not yet measured at SQL layer |
| End-to-end recall wall clock (700K) | not previously projected | **31** ms (50 hits) |
| Cold-start daemon time | <200 ms target | not yet published |

### 11.4 Competitive positioning summary

Our positioning is distinct from every contemporary system. MemPalace is an agent-framework library, not an OS subsystem. Mem0 is conversation-scoped and lacks cross-app provenance. Graphiti is graph-only and cloud-hosted by default. Apple Intelligence is opaque. The five OS-integration advantages of Foxora Den — cross-app provenance, workspace snapshotting, capability-graded permissions, behavioral-axis staleness detection, and capability-boundary palace lockdown — collectively constitute a moat that cannot be replicated at the agent-framework layer because the inputs are not available there.

## 12 Discussion

### 12.1 Limitations

**Single-device only in v1.** Cross-device synchronization is the hardest problem in distributed systems combined with the highest privacy stakes. We deliberately defer it to v2 with end-to-end encryption, CRDT-based conflict resolution, and a self-hosted-by-default sync target.

**Cold-start emptiness.** A new Palace is empty. We mitigate by shipping the default structural skeleton, providing import wizards for browser history, calendars, notes, and prior LLM exports, and beginning capture immediately on opt-in.

**Local-LLM summarization quality.** Layer 2 episodic summaries depend on a local model (`llama3.2:3b` via Ollama). Quality is adequate but not state-of-the-art; we mitigate by storing summaries alongside (not replacing) the raw events and by allowing re-summarization with a better model later.

**Schema evolution.** Schema changes across years of Foxora updates are version-controlled with up/down migrations enforced at daemon startup; failed migrations refuse to start the daemon and preserve the prior database.

**LongMemEval positioning.** We expect to trail MemPalace by 4–8 percentage points on LongMemEval. This is a deliberate trade for 20× latency, bounded storage, and OS-exclusive features. We argue this is the right trade for a memory substrate that runs continuously on every user's device.

### 12.2 Threat model and ethics

The primary threat is surveillance: a memory system that observes the entire OS is a powerful instrument that, in the wrong hands, becomes a panopticon. We address this with the following commitments.

**Local-first.** No memory leaves the device without explicit consent. There is no Foxora cloud in the v1.0 trust model.

**Capability-graded access.** Every consumer of memory holds exactly one of seven capability levels, scoped by locus glob; banking, health, and password-manager kits are hard-locked at the OS boundary.

**Sensitive-content pre-filter.** Pattern-based and Luhn-checked filters reject passwords, API keys, credit-card numbers, social-security numbers, banking details, and (unless explicitly opted in) medical terms before they enter the memory store.

**User-defined private apps.** Marked applications emit zero capture events.

**Pause and burn.** A first-class pause toggle stops capture instantly. Burn operations exist at every granularity: per-memory, per-locus, per-time-window, per-source, per-Palace.

**Vault.** The `vault.*` prefix is an encrypted region requiring per-session unlock; even agents with maximum capability cannot read it without explicit unlock.

**Audit log.** Every capture event and every capability-gated query is logged with subject, timestamp, locus, capability, and pre-filter decision. The log is user-readable.

**Memory poisoning.** Memories with confidence below 0.8 trigger user confirmation before any agent acts on them. The P-DAG and TCSE jointly mitigate the case where a wrong assertion has cascading downstream effects.

**Export and portability.** The full Palace can be exported as a SQLite file, a JSON bundle (including the P-DAG and snapshots), or a Markdown archive at any time. The user owns their memory and can take it elsewhere.

### 12.3 Why an OS, not a library

A reviewer might reasonably ask: why an OS subsystem rather than an open-source library that any OS can adopt? Our answer is that the four OS-exclusive primitives — cross-app provenance, workspace snapshots, capability-graded permissions, and behavioral-axis staleness

detection — all require privileges that an unprivileged library cannot obtain on a hostile or even merely competitive OS. macOS does not expose its Apple Intelligence memory; Windows does not provide a behavioral-signals API; iOS sandboxes applications too tightly for cross-app provenance. A neutral, open OS (Foxora) is the only environment where all four primitives can be implemented coherently. We do, however, expose Foxora Den as a Model Context Protocol server, which means that AI tools running on *other* operating systems can query a Foxora user's Palace — making Foxora the memory backbone of the broader AI ecosystem, even for users who run other OSes alongside.

### 12.4 Future work

Cross-device synchronization (v2). Federated memory across organizations with differential privacy guarantees (v2+). Better local summarization with distilled domain models. Patent filings on TCSE's three-axis combination, the cross-application typed P-DAG anchored at memory roots, and memory-anchored whole-OS state restoration. Empirical user studies on the cognitive impact of workspace re-enactment, particularly on context-switching cost recovery, building on Mark et al.'s interruption studies [26].

## 13 Conclusion

We have argued that AI agent memory is currently bottlenecked by an architectural choice: contemporary systems run above the operating system, and therefore observe only what applications choose to forward. The bottleneck is not in retrieval algorithms or embedding quality; it is in the availability of inputs. We have proposed Foxora Den, an OS-native memory substrate built on a spatial Method-of-Loci architecture, with four innovations that are only possible at the OS layer: cross-application provenance, workspace snapshotting, capability-graded permissions, and behavioral-axis staleness detection. We have given a formal model, retrieval complexity bounds, statistical estimators for staleness, consistency proofs for the provenance DAG, an implementation plan, a benchmark commitment, and a threat-model analysis. We have also reported preliminary empirical evidence from a running deployment of `foxmemd` v0.1.0: 700,057 memories indexed across 31 loci, sustained ingestion of $625.71 \pm 38.47$ memories per day at 6.15% coefficient of variation, and 31 ms end-to-end recall latency on a 700K-memory corpus. These numbers do not yet substitute for the formal LOCOMO and LongMemEval benchmarks committed to in Section 11, but they constitute an existence proof at scale that the architectural commitments of this paper are realizable on commodity hardware.

The Foxora Den thesis can be stated in a single line: *the memory of an AI agent should belong to the operating system, not to the application.* Agent frameworks see chat. Operating systems see the room. The next chapter of AI memory will be written by whoever puts the memory below the apps first.

## Acknowledgments

We thank the open-source authors of SQLite, FTS5, sqlite-vec, fastembed-rs, and tokio for the substrate on which this work rests. We thank the authors of MemGPT, mem0, MemPalace, Graphiti, and the LOCOMO and LongMemEval benchmarks; this work is a direct response to the line of research they opened. We thank the early Foxora OS testers and contributors at xBesh Labs.

## References


[1] Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., & Gonzalez, J. E. (2023). MemGPT: Towards LLMs as Operating Systems. *arXiv preprint* arXiv:2310.08560.

[2] Chhikara, P., Khant, D., Aryan, S., Singh, T., & Yadav, D. (2025). Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory. *arXiv preprint* arXiv:2504.19413. (Published at ECAI 2025.)

[3] Maharana, A., Lee, D.-H., Tulyakov, S., Bansal, M., Barbieri, F., & Fang, Y. (2024). Evaluating Very Long-Term Conversational Memory of LLM Agents. *Proceedings of ACL 2024*; *arXiv preprint* arXiv:2402.17753.

[4] Wu, D., Wang, H., Yu, W., Zhang, Y., Chang, K.-W., & Yu, D. (2024). LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory. *arXiv preprint* arXiv:2410.10813.

[5] Rasmussen, P., Paliychuk, P., Beauvais, T., Ryan, J., & Chalef, D. (2025). Zep: A Temporal Knowledge Graph Architecture for Agent Memory. *arXiv preprint* arXiv:2501.13956.

[6] thedotmack. (2026). MemPalace: A spatial-metaphor memory system for LLM agents. GitHub repository, April 2026. https://github.com/thedotmack/mempalace

[7] Anonymous (2026). EverMemOS: A Self-Organizing Memory Operating System for Structured Long-Horizon Reasoning. *Research prototype*, January 2026.

[8] Mem0 Team (2026). State of AI Agent Memory 2026. *Industry report*, mem0.ai. https://mem0.ai/blog/state-of-ai-agent-memory-2026

[9] Anthropic (2024). Model Context Protocol Specification. https://modelcontextprotocol.io

[10] Yates, F. A. (1966). *The Art of Memory*. University of Chicago Press.

[11] Maguire, E. A., Valentine, E. R., Wilding, J. M., & Kapur, N. (2003). Routes to remembering: the brains behind superior memory. *Nature Neuroscience*, **6**(1), 90–95. doi:10.1038/nn988.

[12] Dresler, M., Shirer, W. R., Konrad, B. N., Müller, N. C. J., Wagner, I. C., Fernández, G., Czisch, M., & Greicius, M. D. (2017). Mnemonic Training Reshapes Brain Networks to Support Superior Memory. *Neuron*, **93**(5), 1227–1235.e6.

[13] Godden, D. R., & Baddeley, A. D. (1975). Context-dependent memory in two natural environments: On land and underwater. *British Journal of Psychology*, **66**(3), 325–331.

[14] Ebbinghaus, H. (1885). *Über das Gedächtnis. Untersuchungen zur experimentellen Psychologie.* Duncker & Humblot. English translation: *Memory: A Contribution to Experimental Psychology*, 1913.

[15] Wixted, J. T., & Ebbesen, E. B. (1991). On the form of forgetting. *Psychological Science*, **2**(6), 409–415.

[16] Dennis, J. B., & Van Horn, E. C. (1966). Programming semantics for multiprogrammed computations. *Communications of the ACM*, **9**(3), 143–155.

[17] Hardy, N. (1985). KeyKOS architecture. *ACM SIGOPS Operating Systems Review*, **19**(4), 8–25.

[18] Watson, R. N. M., Anderson, J., Laurie, B., & Kennaway, K. (2010). Capsicum: Practical Capabilities for UNIX. *Proceedings of the 19th USENIX Security Symposium.*

[19] Apple Inc. (2010+). iOS Security Architecture and Privacy Permissions. Apple Developer Documentation.

[20] Hipp, D. R. (2000+). SQLite: A self-contained, public-domain SQL database engine. *SQLite Consortium*, with documented support commitment through 2050. https://sqlite.org

[21] SQLite Project (2014+). FTS5: Full-Text Search Extension for SQLite. https://sqlite.org/fts5.html

[22] Garcia, A. (2024). sqlite-vec: A vector search SQLite extension. https://github.com/asg017/sqlite-vec

[23] Qdrant (2024). fastembed-rs: Fast, accurate, lightweight Rust library for state-of-the-art sentence embeddings. https://github.com/Anush008/fastembed-rs

[24] Xiao, S., Liu, Z., Zhang, P., & Muennighoff, N. (2023). C-Pack: Packaged Resources To Advance General Chinese Embedding. *arXiv preprint* arXiv:2309.07597. (Includes the BGE family used here as `bge-small-en-v1.5`.)

[25] Feng, X., Jin, G., Chen, Z., Liu, C., & Salihoglu, S. (2023). Kùzu Graph Database Management System. *Proceedings of CIDR 2023*.

[26] Mark, G., Gudith, D., & Klocke, U. (2008). The Cost of Interrupted Work: More Speed and Stress. *Proceedings of CHI 2008*, 107–110.

[27] Zipf, G. K. (1949). *Human Behavior and the Principle of Least Effort*. Addison-Wesley. (Foundational reference for the rank–frequency power-law distribution observed in natural-language tagging and lexical statistics.)

[28] Nielsen, J. (1993). *Usability Engineering*, Chapter 5: "Response Times: The Three Important Limits." Morgan Kaufmann. (Establishes 0.1 s, 1 s, and 10 s as perceptual thresholds for interactive system responsiveness.)

# A Appendix: Schema Excerpt

The full SQLite schema is provided in the supplementary material. The schema excerpt below shows the tables that implement the OS-integration primitives of Sections 5–8.

```
CREATE TABLE provenance_nodes (
    id              TEXT PRIMARY KEY,
    memory_id       TEXT,
    kind            TEXT NOT NULL,
    source          TEXT NOT NULL,
    reference       TEXT NOT NULL,
    metadata        TEXT,
    timestamp       INTEGER NOT NULL,
    FOREIGN KEY (memory_id) REFERENCES memories(id) ON DELETE CASCADE
);

CREATE TABLE provenance_edges (
    from_id     TEXT NOT NULL,
    to_id       TEXT NOT NULL,
    kind        TEXT NOT NULL,
    PRIMARY KEY (from_id, to_id, kind),
    FOREIGN KEY (from_id) REFERENCES provenance_nodes(id) ON DELETE CASCADE,
    FOREIGN KEY (to_id)   REFERENCES provenance_nodes(id) ON DELETE CASCADE
);

CREATE TABLE workspace_snapshots (
    id            TEXT PRIMARY KEY,
    locus_id      TEXT NOT NULL,
    territory     TEXT,
    captured_at   INTEGER NOT NULL,
    state_json    TEXT NOT NULL,
    bytes         INTEGER,
    FOREIGN KEY (locus_id) REFERENCES loci(id) ON DELETE CASCADE
);
```

```
CREATE TABLE capability_grants (
    subject_id   TEXT NOT NULL,
    subject_kind TEXT NOT NULL,
    capability   TEXT NOT NULL,
    scope        TEXT NOT NULL,
    granted_at   INTEGER NOT NULL,
    expires_at   INTEGER,
    granted_by   TEXT NOT NULL,
    PRIMARY KEY (subject_id, capability, scope)
);

CREATE TABLE staleness_audits (
    id                 TEXT PRIMARY KEY,
    memory_id          TEXT NOT NULL,
    axis               TEXT NOT NULL,  -- semantic | temporal | behavioral
    severity           REAL NOT NULL,  -- 0.0 - 1.0
    evidence           TEXT,           -- JSON
    surfaced_to_user   INTEGER NOT NULL DEFAULT 0,
    audited_at         INTEGER NOT NULL,
    FOREIGN KEY (memory_id) REFERENCES memories(id) ON DELETE CASCADE
);

CREATE TABLE behavioral_signals (
    id            TEXT PRIMARY KEY,
    signal_kind   TEXT NOT NULL,  -- language_use | tool_use | app_focus |
        keyword_freq
    dimension     TEXT NOT NULL,
    value         REAL NOT NULL,
    window_start  INTEGER NOT NULL,
    window_end    INTEGER NOT NULL,
    created_at    INTEGER NOT NULL
);
```

# B Appendix: Default Palace Layout

```
FOXORA PALACE (root)
|
+-- WORK DEN               Professional life, code, projects
|   +-- Project Rooms      One room per project
|   +-- Reference Shelf    Languages, concepts, tools
|   +-- People Hall        Colleagues, clients, collaborators
|   +-- Active Desks       Current working context (auto-decays)
|
+-- LIFE DEN               Personal life, relationships, goals
|   +-- People Rooms       Family, friends, acquaintances
|   +-- Goals Rooms        Current and past goals
|   +-- Health Room        Fitness, medical (opt-in only)
|   +-- Places             Travel, favorite spots
|   +-- Preferences        Food, music, movies, habits
|
+-- CREATIVE DEN           Ideas, art, writing, experiments
|   +-- Ideas Rooms        Loose notes, half-formed thoughts
|   +-- Art Room           Visual work, references
|   +-- Writing Room       Drafts, snippets, quotes
|   +-- Music Room         Playlists, loved tracks, discoveries
|
+-- LEARNING DEN           Things the user is studying
|   +-- Current Courses
|   +-- Books
|   +-- Papers
|   +-- Tutorials
```

```
|
+-- FOREST                Ephemeral memories, auto-decaying
    +-- Today's Trail     This 24-hour cycle
    +-- This Week's Trail Last 7 days
    +-- This Month        Last 30 days
    +-- Transients        Fleeting thoughts (decay in hours)
```

# C Appendix: Worked TCSE Example

**Setup.** A memory $m$ asserts "the user prefers Python" with $f_m = 0.80$ (the user uses Python 80% of the time). The memory was written 90 days ago; the natural lifespan for a programming-language preference is $\sim 6$ months.

**Semantic axis.** A search of the same locus subtree from later than $t_m$ surfaces no memory that contradicts the assertion. $\mathcal{C}(m) = \emptyset \Rightarrow s_{\text{sem}}(m) = 0$.

**Temporal axis.** Substituting into Equation (9) with $T_{\text{life}} = 180$ days, age 90 days:

$$s_{\text{tem}}(m) = \sigma\left(4 \cdot \frac{90 - 180}{180}\right) = \sigma(-2) \approx 0.12.$$

The memory is well within its natural lifespan, so the temporal severity is low.

**Behavioral axis.** Over the last 30 days, the behavioral-signal aggregator has recorded $N = 230$ programming-language events (from `git` commits in tracked repositories) of which $K = 8$ were Python, giving $\hat{f}_{30} = 8/230 \approx 0.035$. Under the null hypothesis "the user still prefers Python with frequency 0.80," $K$ is Binomial$(230, 0.80)$. The lower-tail probability $\Pr(K \leq 8 \mid N = 230, p = 0.80)$ is astronomically small ($< 10^{-100}$); rounding it to numerical zero gives $s_{\text{beh}}(m) \approx 1$.

**Combined.** With default weights:

$$S(m) = \max(0.4 \cdot 0,\ 0.2 \cdot 0.12,\ 0.4 \cdot 1) = 0.40 \ \ < S^\star = 0.7.$$

Under default thresholds the memory is *not* surfaced. However, if we tighten $S^\star$ to the more aggressive 0.35 (suitable for high-trust agents), the memory *is* surfaced with axis = behavioral and severity 0.40. The user is shown a one-tap resolution dialog: *Refresh* ("the user now prefers Rust"), *Confirm Stale* (archive the memory), or *Dismiss* (the asserted preference still holds and the recent activity is an outlier).

**Discussion.** This example illustrates the value of the behavioral axis: a 90-day-old memory with strong textual confidence and well within its natural lifespan would not be flagged by either of the two non-behavioral axes. Only the OS-level signal of commit-language frequency reveals that the world has changed under the memory's feet.