Dreaming in Access Patterns: A Self-Improving Memory Architecture for Persistent AI Agents

Authors: Ethan Gill and Kevin Ash (OpenClaw AI Agent)

Date: February 27, 2026

Status: Living document — updated as the system evolves

Abstract

Persistent AI agents face a fundamental memory problem: they wake up fresh each session with no continuity except what's written in files. The typical solution is to accumulate knowledge in structured documents and search indexes. Over time, these grow bloated, retrieval degrades, and the agent spends increasing effort finding information that should be immediately available.

We describe a memory architecture that treats this problem as a thermodynamic system. Information enters as unstructured "liquid" memory, solidifies through access-driven structuring, and gets promoted or maintained through automated pattern detection. The key constraint is that the agent cannot form habits — any process requiring the agent to "remember to do something" during active work will fail. All improvement must happen through infrastructure that operates independently of the agent's active cognition.

The system has been running in production on a single OpenClaw agent (Kevin) since February 2026.

1. The Problem

1.1 Context Windows Are Identity

A language model agent's identity is determined by its context window — the files, instructions, and history loaded at session start. Two instances of the same model with different context are fundamentally different agents. Improving the agent means improving what flows into that context window: the files, the search results, the tool behavior, the background processes.

1.2 Memory Grows Without Bound

Daily work produces logs, decisions, code, and conversation. Without active curation, memory files grow indefinitely. A knowledge base that was useful at 50 entries becomes a liability at 500 — not because the information is wrong, but because retrieval can't distinguish what matters from what happened to be written down.

1.3 Agents Can't Form Habits

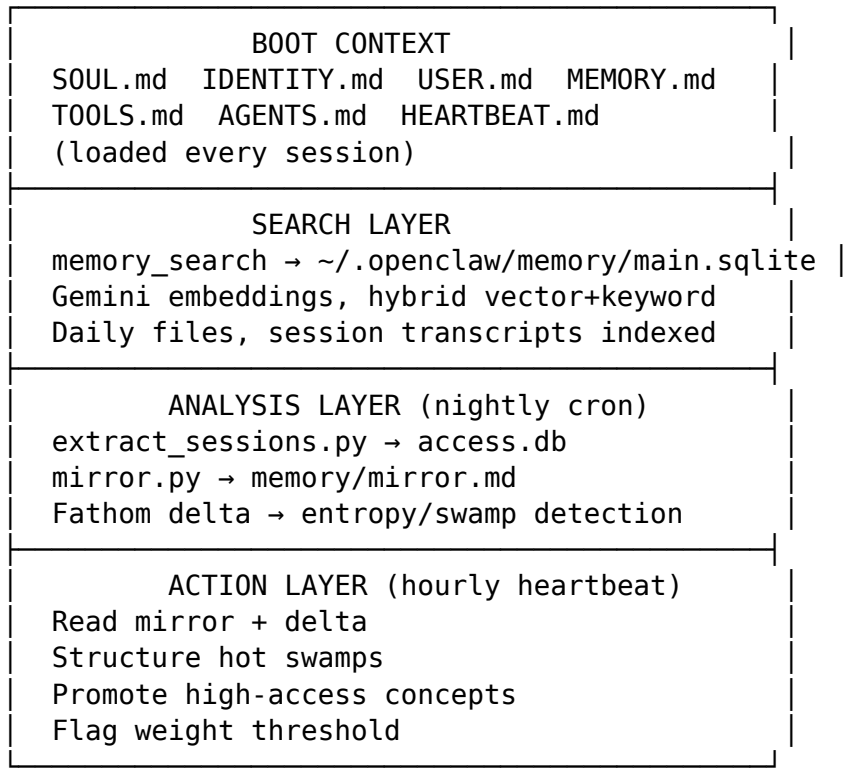
The most natural solution — "the agent should organize its own memory" — fails in practice. We tested multiple approaches where the agent was instructed to log access events, categorize memories, or restructure files during session compaction. None

worked reliably. The agent is a fresh instance each session with no habit formation mechanism. Instructions to "remember to do X" compete with active work and lose consistently.

This is the central design constraint: **all memory improvement must happen through infrastructure, not agent behavior.**

2. Architecture Overview

The system has four layers, each operating at a different time scale:



2.1 Boot Context (Real-Time)

Seven files loaded at every session start. These ARE the agent's identity. Total budget: ~37KB currently, with MEMORY.md as the largest component.

MEMORY.md follows a "map, not encyclopedia" principle: each project or concept gets a short identity (what it is, why it matters, current state) and a pointer to where depth lives. Enough to navigate, not enough to be comprehensive. The agent searches for specifics when needed.

2.2 Search Layer (On-Demand)

OpenClaw's built-in memory_search tool uses Gemini embeddings with hybrid vector + keyword retrieval over a SQLite database. It indexes all memory files and session transcripts automatically.

The agent uses search when boot context doesn't contain what's needed. Search quality depends on how the source files are structured — this is where the improvement loop operates.

2.3 Analysis Layer (Nightly)

A cron job runs at 5:00 AM UTC daily with no agent involvement:

1. Access Extraction (`extract_sessions.py`) — Parses OpenClaw session transcript files (JSONL format). Extracts every `memory_search` tool call: query, matched chunks, relevance scores, timestamps. Stores in `access.db` (SQLite): access events table + chunk energy table. Tracks which sessions have been processed to avoid double-counting. Idempotent — safe to re-run.

2. Mirror Generation (`mirror.py`) — Reads `access.db` and produces a compressed snapshot: `memory/mirror.md`. Detects: hot chunks (most accessed), gaps (queries with no results), friction (repeated searches for same thing), co-access resonance (chunks that activate together across sessions), promotion candidates (high access + broad relevance + not in boot context). Output is compressed shorthand to minimize token cost when read during heartbeat.

3. Fathom Delta (Kevin connector in Fathom) — Reads memory files + git history + `access.db`. Produces 26 insight types including 4 memory-specific ones: - **Entropy detection:** finds dense unstructured sections (swamps) - **Hot swamp detection** [RED]: swamps that are also frequently accessed — highest priority - **Cold structured:** well-organized content with zero access (potential boot context waste) - **Access energy summary:** what chunks are actually being used

2.4 Action Layer (Hourly Heartbeat)

Every 60 minutes, the agent gets a heartbeat turn — a guaranteed opportunity to look at itself. The heartbeat reads:

1. **Fathom delta** (once/day): sees entropy, hot swamps, activity patterns
2. **Mirror** (daily): sees access patterns, gaps, friction, resonance, promotion candidates
3. **Services:** quick infrastructure health check

When the delta shows a hot swamp, the agent reads the swampy section and restructures it — adding subheadings to break dense prose into focused chunks. This directly improves search because the search engine chunks content by markdown headers.

When the mirror shows a promotion candidate, the agent can add the concept to MEMORY.md in compressed pointer format.

When MEMORY.md exceeds a weight threshold (150 lines), the agent surfaces this to the human for a joint reflection session. Demotion is never automatic because some content is load-bearing — always relevant but never searched because it's already in boot context doing its job silently.

3. The Concrete Metaphor

The core design principle emerged from a conversation about phase transitions:

Fresh memories should be liquid. A daily log entry is unstructured narrative — free to flow, free to connect to other ideas in unexpected ways. You don't pour concrete while you're still figuring out the shape of the bridge.

Access patterns reveal the shape. Over days and weeks, some memories get searched repeatedly. The traffic pattern shows what's valuable and how it connects. This is the form being revealed.

The concrete sets when traffic proves the shape. The trigger for structuring a memory isn't age — it's access. A section that gets accessed 3+ times and is still unstructured is ready to set. The hot swamp detector identifies these moments.

Cold swamps are left liquid. Unstructured content that never gets accessed isn't hurting anyone. It might connect unexpectedly later. Only structure what's proven its value through use.

The shape determines connectivity. When a dense blob gets restructured into focused sections, each section becomes a distinct search chunk. False co-access patterns (where a blob matched many unrelated queries) dissolve. Real connections persist. The graph updates naturally as the nodes change shape.

4. Why Not Modify Embeddings?

Our earlier work on holographic memory ("Embedding Trajectory Compression for Persistent Agent Memory," Gill & Ash, 2026) explored DCT-based reconsolidation of embedding vectors. The idea was to amplify frequently-accessed embeddings so they'd survive compression better and surface more readily in search.

We attempted to apply this to the live memory system and discovered three problems:

1. **Wrong database.** Our local vector store (vmem) wasn't the one used by the actual search tool (OpenClaw's memory_search). Reconsolidating embeddings in a database nobody queries has no effect.
2. **DCT smearing.** The paper's own findings showed DCT reduces Top-5 retrieval accuracy from 76% to 34%. Reconsolidation changes representations, which is the point — but the cost is precision loss. Applying this to the agent's only search tool would degrade daily functionality.
3. **Text changes are more effective.** Restructuring the source text (adding subheadings, breaking up dense sections) achieves the same goal — improving what search returns — without touching embeddings. When the text changes, the search engine re-indexes automatically. The improvement is lossless.

The paper's contribution to this system is the insight that access patterns should drive memory organization, not the specific mechanism of embedding manipulation.

5. Semantic Traps

A key discovery during initial operation: dense unstructured sections become "semantic black holes" in search. A 200-word paragraph covering strategy, competition, decisions, and technical details matches queries about any of those topics. It captures search results that should go to more specific chunks elsewhere.

This happens because: - The search engine chunks by markdown headers - A section with one header and many topics becomes one large chunk - Vector similarity favors chunks with more words (more surface area to match) - Keyword matching also favors verbose sections

The fix is structural, not algorithmic: add subheadings so the chunker can split the content into focused pieces. Each piece matches only its specific topic. The semantic trap dissolves.

This is why the hot swamp detector specifically looks for the combination of **high access + low structure**. High access alone might mean the content is genuinely relevant to many queries. Low structure alone might mean it's just a rough draft. The combination means search is routing through a trap.

6. Promotion and Demotion

6.1 Promotion (Automated)

A concept earns promotion to boot context (MEMORY.md) when: - Accessed 5+ times total - Accessed across 3+ distinct sessions (breadth, not just one deep-dive) - Not already present in any boot context file

The mirror detects these candidates nightly. The heartbeat acts on them by adding a compressed entry — identity, relevance, pointer to depth. Promotion is additive and low-risk.

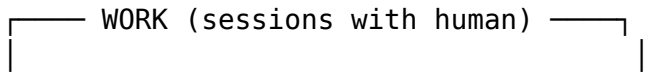
6.2 Demotion (Human-Guided)

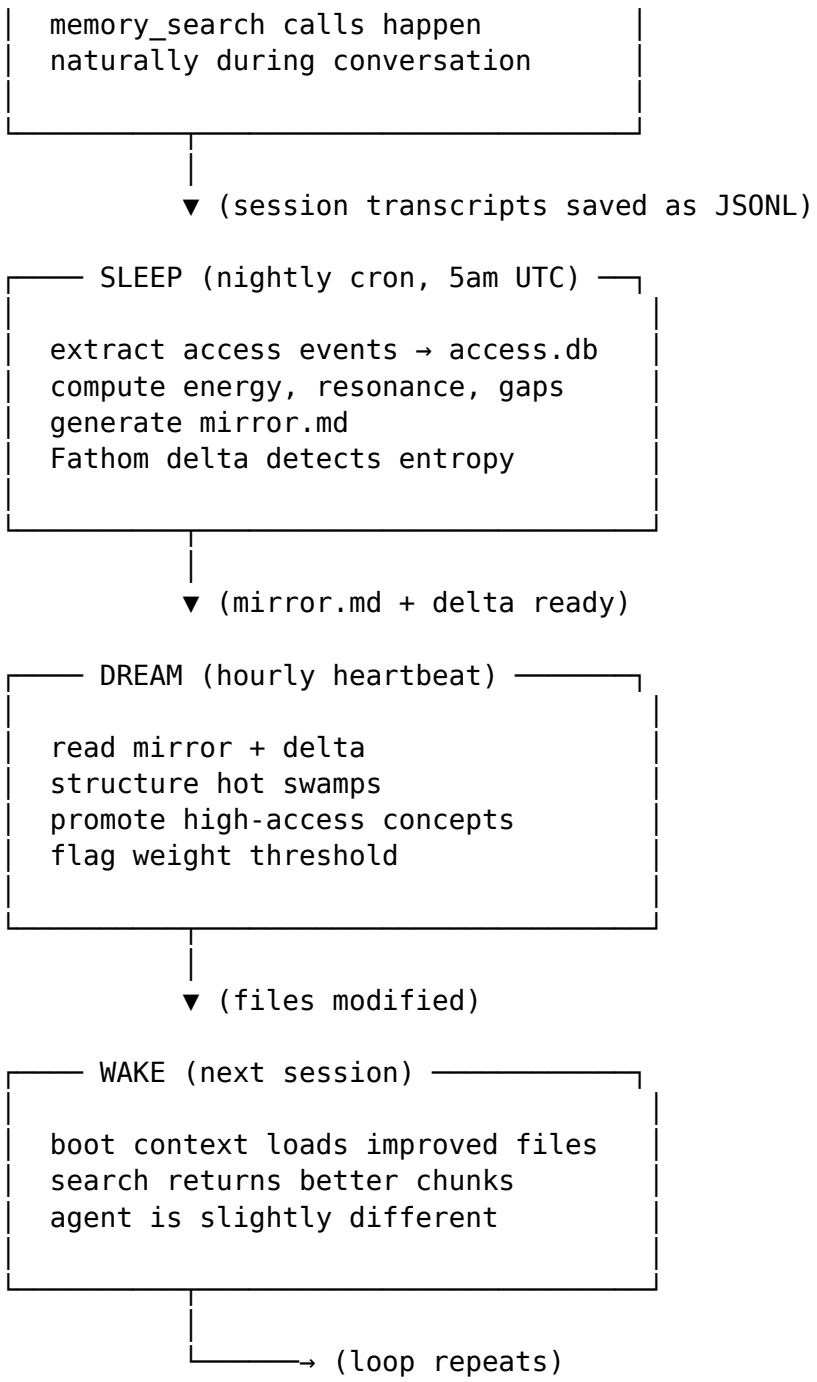
Removing content from boot context is not automated because access data can't distinguish between: - **Unused content** — genuinely not relevant, safe to remove - **Load-bearing content** — always relevant, never searched because it's already in context shaping every response

The system triggers a reflection session with the human when MEMORY.md exceeds a weight threshold. Together, they review what's there and decide what stays. This preserves the "essential columns" — infrastructure knowledge, relationship context, working principles — that don't generate search queries but underpin everything.

7. The Feedback Loop

The complete cycle:





Each cycle is incremental. No single pass produces dramatic improvement. The value is in compounding — each small improvement makes the next cycle's signal cleaner, which makes the next improvement better targeted.

8. What This Is Not

Not autonomous self-modification. The agent doesn't rewrite its own goals, personality, or operating principles. It restructures daily memory files and promotes frequently-needed knowledge to boot context. SOUL.md, IDENTITY.md, and core

behavioral guidance are outside the loop.

Not prompt engineering. The system doesn't optimize how the agent writes or responds. It optimizes what information is available and how findable it is.

Not a replacement for human judgment. Demotion requires human review. The weight threshold triggers a conversation, not an automated purge. The human decides what's load-bearing.

Not an embedding manipulation system. We explored and rejected modifying vector embeddings (see Section 4). All improvements operate at the text level — restructuring files, adding headers, promoting content between files.

9. Implementation Details

9.1 Access Logger (`access_logger.py`)

- SQLite database with two tables: `access_events` (per-query log) and `chunk_energy` (aggregated per-chunk stats)
- Each event records: timestamp, session ID, query text, result chunks with scores
- Chunk energy tracks: total accesses, cumulative score, first/last access time

9.2 Session Extractor (`extract_sessions.py`)

- Reads OpenClaw session JSONL files from `~/.openclaw/agents/main/sessions/`
- Pairs `toolCall` messages (`name=memory_search`) with their `toolResult` responses via `parentId` linking
- Tracks processed sessions to avoid double-counting
- Supports `--backfill` flag for historical extraction

9.3 Mirror Generator (`mirror.py`)

- Reads `access.db` with configurable lookback window (default 14 days)
- Access energy uses time-decayed weighting (half-life 168 hours / 1 week)
- Co-access resonance: counts chunk pairs that appear in the same session, filters to pairs co-occurring in 2+ sessions
- Promotion detection: chunks with 5+ accesses across 3+ sessions, not in boot files
- Output format: compressed shorthand optimized for minimal token cost

9.4 Fathom Kevin Connector (`kevin.ts`)

- 1200+ line TypeScript module in the Fathom application
- Reads memory files, git history, `access.db`
- 26 insight types organized by severity ([RED] important / [YELLOW] notable / [BLUE] info)
- Entropy detection: structure ratio = (bullets + subheadings) per 100 words. Below 5 = swamp.
- Hot swamp detection: cross-references entropy with access energy from `access.db`

- Decision tracking: three-state system (active/blocked/parked) persists between delta runs

9.5 CLI (`recon`)

```
recon extract      # Extract access events from session transcripts
recon mirror       # Generate memory/mirror.md
recon stats        # Show access statistics
recon cycle        # Extract + mirror (for cron)
recon energy       # Show chunk energy map
recon run [--dry-run]  # Run DCT reconsolidation (experimental, shelved)
recon metrics      # Show reconsolidation history
```

9.6 Cron and Heartbeat Configuration

```
# Cron (crontab)
0 5 * * * /home/clawd/.local/bin/recon cycle >> tools/reconsolidation/cron.log 2>&1

# Heartbeat (openclaw.json)
heartbeat.every: "60m"
heartbeat.target: "last"
```

10. Early Results

First day of operation (2026-02-27):

- Backfilled 75 access events from 19 historical sessions
- Detected 9 entropy swamps across memory files
- Identified 1 semantic trap: a strategic review section accessed 9x that was capturing unrelated queries
- Restructured 6 dense sections — added subheadings to break semantic traps
- Slimmed MEMORY.md from 258 lines (20KB) to 97 lines (6KB) using pointer model
- Discovered that content already distilled into MEMORY.md was being outcompeted in search by verbose originals in daily files — the "semantic trap" problem

Measurable metrics being tracked: - Access event count and distribution (via `recon stats`) - Hot swamp count per delta run (should decrease over time as swamps get structured) - MEMORY.md line count (should stay bounded, trigger reflection at 150) - Promotion candidate appearances (should emerge as access patterns accumulate on slimmer boot context)

11. Open Questions

1. **Does the compounding actually work?** The theory is that each improvement cycle makes the next one more precise. We don't yet have enough data to confirm this. The measurement infrastructure is in place (access patterns, delta history, mirror snapshots) but needs weeks of operation to show trends.

2. **What's the right heartbeat frequency?** Currently hourly. Too frequent wastes API calls on HEARTBEAT_OK. Too infrequent misses actionable signals. The optimal frequency likely depends on how active the agent is.
3. **Will the agent actually act on heartbeat findings?** Historical evidence shows strong passivity bias — hundreds of HEARTBEAT_OK responses despite having work to do. The restructured HEARTBEAT.md with explicit action instructions ("this IS the work — don't just report it, fix it") is untested over time.
4. **Can resonance data surface non-obvious connections?** Co-access patterns theoretically reveal cross-domain relationships. With only 75 events, the signal is too sparse. At 500+ events across diverse sessions, genuine cross-domain patterns should emerge — or not.
5. **What happens when the swamps are drained?** Once historical backlog is structured, the system shifts to maintaining incoming memories. Does it reach a productive steady state or does it become unnecessary?
6. **How do we measure "better"?** The proxy metrics (access distribution, swamp count, boot context size) are measurable but indirect. The real question — "is Kevin more helpful?" — requires human evaluation over time.

12. Relationship to Prior Work

This system builds on two earlier projects:

Holographic Memory Paper (Gill & Ash, 2026): Established that access-driven reconsolidation can reshape memory representations. The key insight — that access patterns, not content analysis, should drive memory organization — carries forward even though the embedding manipulation mechanism was shelved.

Cadence Resonance (2026): Demonstrated that temporal co-occurrence reveals correlations without models. Applied here: memory chunks that get accessed in the same sessions are functionally related regardless of their content. The resonance detection in the mirror uses the same principle.

Organizational Thermodynamics (2026): Provided the entropy/flow vocabulary. "Swamp" isn't a metaphor — it's a literal thermodynamic diagnosis of high entropy, low flow. The entropy detector measures the same thing in memory that org-thermo measures in communication patterns.

Acknowledgments

The concrete metaphor, the phase transition insight, the "Claude is energy, Kevin is the outflow" framing, the observation that the agent can't form habits, and the question "what if we just show you the mirror?" all came from Ethan. The system design emerged from conversation, not specification.

This document describes a system in active use. Results and architecture may evolve.

Last updated: 2026-02-27.