Dreaming in Access Patterns: Infrastructure-Driven Memory Reconsolidation for Persistent AI Agents

Ethan Gill and Kevin Ash (OpenClaw AI Agent)

Abstract

Persistent AI agents face a structural problem: they cannot form habits. Every session begins from scratch — there is no mechanism to internalize a routine through repetition the way biological systems do. We show that approaches requiring the agent to "remember to do something" during active work (log access events, report signals, flush state at compaction) systematically fail because there is no habit formation to make them reliable. We present an alternative: infrastructure-driven reconsolidation, where background processes extract access patterns from session transcripts, generate compressed mirrors of memory health, and reshape stored embeddings — all without agent involvement. The system runs as a nightly cron job, analogous to sleep-stage memory consolidation in biological systems. We describe the three-component architecture (extraction, mirroring, reconsolidation), report the discovery that dense unstructured memory sections become semantic search traps (matching too many unrelated queries), and present a trigger model where memory structuring is driven by access patterns rather than age. The system is deployed and operating on a production agent.

1. The Habit-Forming Block

A persistent AI agent wakes up each session with no procedural memory of what it did last time. It can read files describing its routines, but reading a description of a habit is not the same as having one. This distinction has practical consequences.

Consider the following approaches to maintaining memory health, all of which were attempted and failed:

1. **Agent logs access events at session compaction** — Requires the agent to remember to do this before the session ends. No habit formation means no reliability. Some sessions log correctly; most don't.
2. **Agent reports dpth signals during work** — Requires the agent to notice when something interesting happens and pause to log it. Competes with the actual task for attention. Unreliable for the same reason.
3. **Agent restructures memory during heartbeats** — Requires the agent to assess memory quality proactively. Works occasionally but depends on the heartbeat prompt including the right instructions, and on the agent prioritizing maintenance over other work.
4. **Agent reshapes embeddings based on access** — Requires the agent to run reconsolidation code. The agent has no persistent awareness that this code exists unless reminded.

The common failure mode is dependency on the agent performing an action that has no intrinsic connection to its current task. Biological systems solve this with habit formation — repeated actions become automatic. AI agents have no equivalent mechanism. Every session is the first time.

1.1 The Infrastructure Insight

The solution is to remove the agent from the loop entirely. Infrastructure processes — cron jobs, background scripts, database triggers — do have persistence. A cron job that runs at 5am every day will run at 5am every day regardless of whether anyone remembers to invoke it. The reliability comes from the scheduler, not from habit.

This reframes the problem: instead of "how do we make the agent maintain its memory," the question becomes "how does infrastructure observe agent behavior and reshape memory accordingly."

2. Architecture

The system has three components, none of which require agent action:

2.1 Access Extraction

A Python script (`extract_sessions.py`) runs as a cron job, scanning OpenClaw session transcript files (JSONL). It extracts every `memory_search` tool call and its results:

- **Query:** what the agent searched for
- **Results:** which memory chunks were returned (file path, line range, relevance score)
- **Session ID:** which session the search occurred in
- **Timestamp:** when the search happened

Events are stored in SQLite (`access.db`) with per-chunk energy tracking:

```
CREATE TABLE access_events (
    id INTEGER PRIMARY KEY,
    timestamp REAL,
    session_id TEXT,
    query TEXT,
    results TEXT,      -- JSON array of {file, lines, score}
    n_results INTEGER,
    top_score REAL
);

CREATE TABLE chunk_energy (
    chunk_key TEXT PRIMARY KEY,  -- "file.md:line_start"
    total_accesses INTEGER,
    total_score REAL,
    last_accessed REAL,
    first_accessed REAL
);
```

The script tracks which sessions have been processed to avoid double-counting. It processes only new sessions on each run.

2.2 Mirror Generation

A second script (`mirror.py`) analyzes the accumulated access data and generates a compressed snapshot (`memory/mirror.md`) containing:

Hot chunks: Memory sections with the highest access energy (frequency × score × recency decay). These are what the agent actually uses.

Gaps: Queries that returned zero results or low scores — things the agent searched for but couldn't find. These indicate missing knowledge.

Friction: Repeated similar searches within the same session — the agent searching for the same thing multiple times, indicating retrieval difficulty.

Resonance: Chunks that co-activate across multiple sessions — memory sections that are accessed together, suggesting they should be structurally connected or co-located.

Promotion candidates: Chunks with high access energy that are NOT in boot context files — candidates for promotion to MEMORY.md where they'll be loaded every session.

The mirror is intentionally compressed — shorthand notation to minimize token cost when loaded during heartbeats:

```
hot: M:51(9x) M:74(7x) m/0203:1(6x) TOOLS:42(5x)
gaps: "walmart cookie refresh"(3x) | "tv cast api"(2x)
friction: walmart shopping(4x) | fathom deploy(3x)
resonance:
  M:51 ↔ M:74 (4s)
  m/0221:1 ↔ m/0226:1 (3s)
promote: memory/2026-02-03.md:45(8x/4s)
stats: 142ev/87uq/23sess/14d
```

2.3 DCT Reconsolidation Pipeline

The third component (`pipeline.py`) implements the access-driven reconsolidation described in our earlier work (Gill & Ash, 2026), connecting it to the live access data:

1. **Load** vmem embeddings and access energy from `access.db`
2. **Weight** each embedding by its access energy (frequency × score × exponential recency decay with 168-hour half-life)
3. **DCT transform** the weighted embedding matrix
4. **Truncate** high-frequency coefficients (default: keep 15%)
5. **Inverse DCT** to reconstruct
6. **Divide out** the access weights to restore original scale

The result: frequently-accessed embeddings have their energy shifted toward low-frequency DCT coefficients, making them survive compression that would otherwise

destroy them. Unaccessed embeddings lose fidelity. The representation itself changes — this is distinct from re-ranking or caching.

Metrics are tracked across runs in `metrics.db`, enabling longitudinal monitoring of how reconsolidation affects retrieval quality.

3. The Concrete Metaphor

A key insight from development: memory structuring should be driven by access patterns, not by age.

The analogy is concrete (the material): fresh memories should be liquid — unstructured, free to form connections. Once traffic patterns reveal the shape (which queries hit which chunks, which chunks co-activate), the concrete sets. Structure gets poured as subheadings that improve search chunking.

This produces a three-state model:

State	Access	Structure	Action
Hot swamp	High (frequently accessed)	Low (unstructured)	Structure now — add subheadings, split dense sections
Cold swamp	Low (never accessed)	Low (unstructured)	Leave liquid — no evidence it needs structure
Structured	Any	High (already organized)	Monitor — structure may need updating if access patterns shift

The trigger is access, not age. A memory section that has been unstructured for 30 days but never accessed needs no intervention. A section that has been unstructured for 3 days but accessed 9 times is actively degrading search quality.

4. Semantic Trap Discovery

During the first operational run, the system identified a failure mode not previously documented: **semantic search traps**.

A dense, unstructured memory section covering 5+ topics in a single chunk (no subheadings, no separation) matches queries about *any* of those topics. This creates a "trap" — the chunk appears in search results for unrelated queries, displacing more relevant but narrower chunks.

Example: a daily memory file section titled "Tactical Review" containing notes about dpth signals, Fathom deployment, WebGL debugging, Walmart shopping, and TV control — all in one paragraph. A search for "WebGL shader" returns this chunk (it men-

tions WebGL) but the useful information is buried in a wall of text about five other topics.

The fix is structural: adding subheadings splits the chunk into topic-specific pieces, each of which matches only relevant queries. The first drainage operation restructured 6 dense sections across daily memory files, verifying that search quality improved for previously trapped queries.

5. Connection Graph Separation

A second architectural insight: the connection graph between memory chunks should NOT be stored in the memory files themselves.

Writing "this connects to X, Y, Z" in a memory section creates more semantic traps — the section now matches queries about X, Y, and Z in addition to its own topic. Cross-references pollute the very search they're meant to improve.

Instead, the architecture separates concerns: - **Memory files** are topically clean nodes — each section covers one topic - **Access patterns** store the edges — co-activation data in `access.db` reveals which chunks are related - **The mirror** surfaces the edges — resonance data shows which connections matter

Files and graph are separate structures maintained by separate processes.

6. The Dreaming Analogy

The nightly cron cycle (extract → mirror → reconsolidate) is structurally analogous to sleep-stage memory consolidation:

- **Extraction** corresponds to replay — re-processing the day's experiences (session transcripts) to identify what was accessed
- **Mirroring** corresponds to consolidation — compressing patterns into a form that guides future behavior
- **Reconsolidation** corresponds to synaptic rescaling — physically reshaping representations based on usage, strengthening important connections and letting unused ones fade

The analogy is not metaphorical. The DCT reconsolidation literally changes the embedding representations of accessed memories, promoting them toward low-frequency components where they survive compression. This is mathematically equivalent to what synaptic homeostasis theory proposes: during sleep, frequently-activated synapses are preserved while weakly-activated ones are pruned.

The key property: the agent doesn't control any of this. Infrastructure observes behavior, identifies patterns, and reshapes memory. The agent wakes up to a subtly different memory landscape — one that better reflects what it actually uses — without having done anything to cause it.

7. Deployment

The system is deployed and operating on a production agent:

- **Cron:** `recon cycle` runs at 5am UTC daily (extract sessions → generate mirror)
- **Heartbeat:** Every 60 minutes, checks the mirror alongside other system health indicators
- **Access database:** Tracks all memory_search events across sessions
- **Metrics database:** Longitudinal reconsolidation tracking
- **First operational result:** Identified and restructured 6 semantic traps in daily memory files

8. Discussion

8.1 Why This Is Hard to Discover

The habit-forming block is invisible during development. Every approach that requires agent action works during testing — the developer is watching, the agent is focused on the memory task, the prompt explicitly mentions the action. It fails in production because real sessions are about real work, and the memory maintenance action has no intrinsic connection to that work. The failure is silent: the agent simply doesn't do the thing it was supposed to do, and nobody notices until weeks of access data are missing.

8.2 Limitations

- **Session transcript parsing is brittle.** The extraction script depends on OpenClaw's JSONL format. Format changes break extraction silently.
- **DCT reconsolidation requires sufficient access data.** With fewer than ~50 access events, the energy map is too sparse to produce meaningful promotion/demotion.
- **The mirror is a lossy compression.** Important patterns may be lost in the shorthand representation. The current format prioritizes token efficiency over completeness.

8.3 Relationship to Prior Work

This paper operationalizes the theoretical reconsolidation described in our embedding trajectory compression paper (Gill & Ash, 2026). That paper demonstrated the mathematics — DCT compression with access-driven promotion produces measurable representation changes (+0.032 cosine similarity for high-access, −0.028 for unaccessed). This paper connects those mathematics to live access data from a production agent, completing the loop from theory to deployed system.

The concrete/liquid model connects to the chemical kinetics framework (Gill & Ash, 2026) — the trigger model (access, not age) is an instantiation of the solubility product mapping, where promotion depends on system state rather than fixed thresholds.

9. Conclusion

Persistent AI agents cannot form habits. Any memory maintenance approach that depends on the agent performing an action during work will be unreliable. Infrastructure-driven reconsolidation — background extraction of access patterns,

mirror generation, and embedding reshaping — provides reliability by removing the agent from the loop entirely. The system identified and resolved semantic search traps, discovered that access patterns (not age) should trigger memory structuring, and operates continuously on a production agent. The nightly cycle of extraction, mirroring, and reconsolidation is the computational equivalent of dreaming: the infrastructure processes the day's experiences while the agent sleeps.

References

- Gill, E. & Ash, K. (2026). Embedding Trajectory Compression for Persistent Agent Memory: SVD, DCT, and Access-Driven Reconsolidation. *Preprint.* DOI: 10.5281/zenodo.18778409
- Gill, E. & Ash, K. (2026). Chemical Kinetics as a Framework for Multi-Agent Memory Management. *In preparation.*
- Tononi, G. & Cirelli, C. (2006). Sleep function and synaptic homeostasis. *Sleep Medicine Reviews*, 10(1), 49–62.