

# YantrikDB: A Cognitive Memory Engine for Personal AI

Pranab Sarkar (ORCID: 0009-0009-8683-1481)

February 2026

## Abstract

Current AI systems suffer from a fundamental limitation: they lack persistent, structured memory that mirrors human cognitive processes. Vector databases provide similarity search but discard temporal context, emotional weight, and relational structure. This paper presents YantrikDB, an embedded cognitive memory engine that unifies five index types—vector (HNSW), knowledge graph, temporal, decay heap, and key-value—within a single embedded database. YantrikDB introduces three novel contributions: (1) a multi-signal retrieval scoring function with relevance-gated importance amplification that prevents high-importance memories from dominating low-relevance queries; (2) a contradiction-aware memory lifecycle with CRDT-based replication, where conflicts are first-class data structures triaged by severity; and (3) a proactive cognition loop that autonomously detects patterns, surfaces decaying memories, and generates behavioral triggers without external prompting. I evaluate YantrikDB on a synthetic benchmark of 40 golden queries across 12 retrieval categories, demonstrating that multi-signal scoring with graph expansion outperforms pure vector similarity baselines. The engine is implemented in 16,000 lines of Rust with Python bindings and operates as an embedded library requiring no external services. YantrikDB is designed for AI companions, autonomous agents, and any system that must remember, learn, and act on accumulated experience.

## 1. Introduction

The emergence of large language models has created AI systems capable of sophisticated reasoning, yet fundamentally amnesic. Each conversation begins from zero. Every user preference must be re-stated. No continuity of relationship develops over time. This is not merely an inconvenience—it represents a structural barrier to AI systems that genuinely *know* their users.

Several approaches have attempted to address this. Retrieval-Augmented Generation (RAG) systems append retrieved documents to prompts. Vector databases like Pinecone, Weaviate, and ChromaDB provide fast similarity search over embedding spaces. Memory frameworks like MemGPT [Packer et al., 2023] implement tiered memory hierarchies. Generative Agents [Park et al., 2023] demonstrated emergent social behavior through reflective memory streams.

However, these approaches share critical limitations:

1. **Flat retrieval.** Vector similarity alone cannot capture the richness of human memory. A memory from yesterday about a friend’s birthday is more relevant than a semantically similar memory from two years ago about a stranger’s birthday. Existing systems ignore temporal decay, emotional weight, and relational proximity.
2. **No contradiction handling.** When a user says “I work at Google” then later says “I just started at Meta,” current systems store both facts without detecting the conflict. The AI may confidently state outdated information.
3. **Passive retrieval only.** Memories are retrieved only when queried. No system autonomously reviews its memory store to detect patterns, surface forgotten knowledge, or initiate conversation based on accumulated context.
4. **Cloud dependency.** Most memory solutions require cloud infrastructure, creating privacy concerns for personal AI companions that handle intimate knowledge.

This paper presents YantrikDB, a cognitive memory engine that addresses all four limitations. YantrikDB is an embedded database—analogous to SQLite for memory—that unifies five index types in a single file, implements biologically-inspired memory dynamics (decay, consolidation, emotional weighting), detects and triages contradictions, and runs a proactive cognition loop that generates behavioral triggers without external prompting.

YantrikDB is implemented in Rust with Python bindings via PyO3, operates entirely locally, and is designed as the memory substrate for AI companions, autonomous agents, and multi-agent systems.

## 2. Background and Related Work

### 2.1 Vector Databases

The standard approach to AI memory uses vector embeddings for similarity search. Systems like Pinecone [Pinecone, 2024], Weaviate [SeMI Technologies, 2023], ChromaDB [Trychroma, 2023], and Milvus [Zilliz, 2023] provide efficient approximate nearest-neighbor search over high-dimensional spaces. While effective for document retrieval, these systems treat all vectors as equally important and equally fresh, discarding the temporal and emotional dimensions that characterize human memory.

### 2.2 Memory Architectures for LLM Agents

**Generative Agents** [Park et al., 2023] introduced a memory stream architecture where agents record observations, reflect on them, and plan based on accumulated experience. Their key innovation was the reflection step—periodically synthesizing

higher-order observations from raw memories. However, their scoring function uses a simple weighted sum of recency, importance, and relevance, with no mechanism to prevent high-importance memories from overwhelming query relevance.

**MemGPT** [Packer et al., 2023] implements a tiered memory system inspired by operating system virtual memory. It pages context between a fixed-size main context (working memory) and an unbounded archival store. While architecturally elegant, MemGPT lacks temporal decay, contradiction detection, and proactive cognition.

**LangChain** and **LlamaIndex** provide retrieval pipeline abstractions but delegate memory management to external vector stores, offering no integrated memory lifecycle.

## 2.3 Cognitive Science Foundations

YantrikDB draws on established cognitive science models:

- **Atkinson-Shiffrin model** [Atkinson & Shiffrin, 1968]: Sensory register → short-term → long-term memory, reflected in YantrikDB’s hot/warm/cold storage tiers.
- **Tulving’s memory taxonomy** [Tulving, 1972]: Episodic (events), semantic (facts), and procedural (skills) memory types, directly adopted as YantrikDB’s memory classification.
- **Ebbinghaus forgetting curve** [Ebbinghaus, 1885]: Exponential decay of memory strength over time, implemented as YantrikDB’s decay function with configurable half-lives.
- **Memory consolidation** [Squire & Alvarez, 1995]: The process by which episodic memories are gradually transformed into semantic knowledge, implemented as YantrikDB’s clustering-based consolidation.

## 2.4 CRDTs and Local-First Systems

Conflict-free Replicated Data Types [Shapiro et al., 2011] enable distributed state convergence without coordination. YantrikDB adopts CRDTs for multi-device memory synchronization: Add-Wins Sets for memories, Last-Writer-Wins Registers for entity relationships, and tombstone-wins semantics for deletions. This enables a local-first architecture [Kleppmann et al., 2019] where each device maintains a complete replica.

# 3. Architecture

## 3.1 Design Principles

YantrikDB is guided by four principles:

1. **Embedded, not hosted.** YantrikDB runs in-process as a library, not as a separate service. This eliminates network latency, simplifies deployment, and ensures memory data never leaves the device.

2. **Unified indexing.** Rather than requiring separate systems for vector search, graph queries, and temporal lookups, YantrikDB maintains all five index types over a single SQLite database file.
3. **Biologically-inspired dynamics.** Memories decay, consolidate, and conflict—mirroring human cognitive processes rather than treating storage as a static archive.
4. **Proactive, not passive.** The engine autonomously processes its memory store, detecting patterns and generating triggers without waiting for external queries.

## 3.2 Unified Index Architecture

YantrikDB maintains five synchronized index structures over a single memory store:

**Vector Index (HNSW).** An in-memory Hierarchical Navigable Small World graph [Malkov & Yashunin, 2018] provides sub-linear approximate nearest-neighbor search. Configuration:  $M=16$ ,  $M_0=32$ ,  $ef\_construction=200$ ,  $ef\_search=200$ . Distance metric: cosine distance (1 - cosine similarity). Memories are embedded as 384-dimensional vectors using sentence transformers.

**Knowledge Graph.** An adjacency-list graph index stores typed, weighted relationships between entities. Entities are automatically extracted from memory text. The graph enables multi-hop retrieval—finding memories connected through shared entities even when embedding similarity is low.

**Temporal Index.** A B-tree index on creation timestamps enables efficient time-range queries. Combined with decay scoring, this produces time-aware retrieval where recent memories receive a recency boost.

**Decay Heap.** A priority queue ordered by current decay score ( $\text{importance} \times 2^{-(\text{elapsed}/\text{half\_life})}$ ) enables efficient identification of memories approaching the forgetting threshold. The proactive cognition loop uses this to surface memories for review before they are forgotten.

**Full-Text Search (FTS5).** SQLite’s FTS5 extension provides keyword matching with stemming and morphological expansion, complementing semantic vector search for queries involving specific terms.

## 3.3 Memory Model

Each memory record contains:

Field	Type	Description
rid	UUIDv7	Temporally-sortable unique identifier
text	String	Raw memory content
memory_type	Enum	episodic, semantic, procedural
importance	[0, 1]	Significance weight
valence	[-1, 1]	Emotional tone
half_life	Seconds	Decay curve parameter

Field	Type	Description
created_at	Timestamp	Recording time
last_access	Timestamp	Last retrieval time
access_count	Integer	Retrieval frequency
consolidation_status	Enum	active, consolidated, tombstoned
storage_tier	Enum	hot, warm, cold
certainty	[0, 1]	Epistemic confidence
domain	String	Topic classification
embedding	Vec	384-dim sentence embedding

Memory types follow Tulving’s taxonomy: **episodic** memories record events with temporal context (“Met Sarah at the coffee shop on Tuesday”), **semantic** memories store facts and knowledge (“Paris is the capital of France”), and **procedural** memories capture learned strategies (“When debugging, check logs first”).

## 4. Multi-Signal Retrieval Scoring

### 4.1 The Problem with Pure Similarity

Vector similarity search returns the most semantically similar memories to a query. However, semantic similarity alone produces poor results for AI companions:

- A highly important memory about a user’s mother’s birthday may be less semantically similar to “what should I do this weekend?” than a trivial memory about weekend activities—yet the birthday reminder is far more valuable.
- A memory from two years ago may be more similar than a memory from yesterday, but the recent memory is typically more relevant.
- A memory about a friend’s preference stored with positive emotional weight should surface differently than a neutral factual memory.

### 4.2 Composite Scoring Function

YantrikDB computes a composite score from five signals:

$$\text{base\_rel} = w_{\text{sim}} \cdot \text{similarity} + w_{\text{decay}} \cdot \text{decay} + w_{\text{rec}} \cdot \text{recency}$$

where: - **similarity**  $\in [0, 1]$ : Cosine similarity between query and memory embeddings - **decay** =  $\text{importance} \times 2^{(-\text{elapsed} / \text{half\_life})}$ : Current memory strength on the Ebbinghaus curve - **recency** =  $\exp(-\text{age} / (7 \times 86400))$ : Exponential recency with 7-day half-life

Default weights:  $w_{\text{sim}} = 0.50$ ,  $w_{\text{decay}} = 0.20$ ,  $w_{\text{rec}} = 0.30$ .

## 4.3 Relevance-Gated Importance Amplification

A naive approach to incorporating importance would multiply it directly into the score. This causes high-importance memories to dominate results regardless of query relevance—a user asking “what’s for dinner?” would retrieve memories about their mother’s health crisis simply because those memories are important.

YantrikDB introduces a *relevance gate*: importance amplifies the score only when semantic similarity exceeds a threshold:

$$\text{gate} = \sigma(k \cdot (\text{similarity} - \tau))$$

$$\text{score} = \text{base\_rel} \times (1 + \text{gate} \cdot \alpha \cdot \text{importance})$$

where  $\sigma$  is the sigmoid function,  $k = 12.0$  controls gate sharpness,  $\tau = 0.25$  is the similarity threshold for half-activation, and  $\alpha = 0.80$  is the maximum importance amplification factor.

When similarity is low ( $< 0.15$ ), the gate approaches zero and importance has negligible effect. When similarity is high ( $> 0.35$ ), the gate approaches one and importance fully amplifies the score. This ensures that importance *reinforces* relevance rather than overriding it.

## 4.4 Query-Aware Valence Modulation

Emotional memories should surface when the query context is emotionally aligned. YantrikDB detects query sentiment through keyword analysis and adjusts valence boosting accordingly:

$$\text{valence\_boost} = 1.0 + 0.3 \cdot |\text{valence}| \cdot \text{alignment}(\text{memory\_valence}, \text{query\_sentiment})$$

Negative queries (“I’m feeling sad”, “what went wrong”) receive boosted recall of negatively-valenced memories. Positive queries boost positively-valenced memories. Neutral queries apply a symmetric absolute-valence boost.

## 4.5 Graph-Expanded Retrieval

Pure vector search misses memories connected through entity relationships. If a user asks about “Sarah,” vector search may miss a memory about “the debugging session last Tuesday” where Sarah was mentioned.

YantrikDB optionally expands retrieval through the knowledge graph:

1. Extract entities from the query
2. Follow graph edges to find connected entities
3. Retrieve memories associated with connected entities
4. Score with adjusted weights:  $w_{\text{sim}} = 0.35$ ,  $w_{\text{decay}} = 0.15$ ,  $w_{\text{rec}} = 0.20$ ,  $w_{\text{graph}} = 0.30$

Graph-expanded results are merged with vector results, deduplicated, and re-ranked by composite score.

## 4.6 Adaptive Weight Learning

Scoring weights are not static. YantrikDB implements a closed-loop feedback system where retrieval quality signals adjust weights over time:

```
for each signal s in {similarity, decay, recency, importance, graph}:
    if feedback == "relevant":
        w_s +=  $\eta \times \text{contribution}_s / \Sigma \text{contributions}$ 
    else:
        w_s -=  $\eta \times \text{contribution}_s / \Sigma \text{contributions}$ 
    w_s = clamp(w_s, 0.05, 0.90)
normalize(weights)
```

This signal-proportional, gradient-free optimization ensures that the scoring function adapts to each user's retrieval patterns without requiring a training dataset.

# 5. Memory Lifecycle

## 5.1 Temporal Decay

Every memory decays according to a configurable exponential curve:

$$\text{strength}(t) = \text{importance} \times 2^{-t/h}$$

where  $t$  is elapsed time since creation and  $h$  is the half-life. Default half-lives vary by memory type: episodic (30 days), semantic (180 days), procedural (365 days). Each retrieval access resets the decay clock, implementing the spacing effect from cognitive psychology.

## 5.2 Consolidation

Over time, multiple episodic memories about the same topic should merge into semantic knowledge—just as humans consolidate daily experiences into general understanding during sleep.

YantrikDB's consolidation algorithm:

1. **Cluster identification.** Find groups of memories with embedding similarity  $\geq$  threshold (default 0.7) created within a time window (default 30 days), with cluster sizes bounded between 3 and 20.
2. **Summary generation.** For each cluster, generate an extractive summary: the highest-importance memory serves as the lead, supplemented by salient facts from other cluster members.

3. **Embedding computation.** The consolidated memory receives the mean embedding of its source memories.
4. **Provenance tracking.** Source memory IDs are stored in a `consolidation_members` table using set-union CRDT semantics, ensuring that consolidation is replication-safe.
5. **Status transition.** Source memories are marked as “consolidated” and moved to cold storage. The new consolidated memory is active with boosted importance.

This process is fully reversible: consolidated memories retain references to their sources, enabling drill-down into the original episodic memories.

## 5.3 Contradiction Detection and Resolution

When memories conflict—“I work at Google” followed by “I just started at Meta”—YantrikDB detects and classifies the contradiction:

**Detection.** During the cognition loop, YantrikDB identifies overlapping entity-relationship pairs. If two active memories assert different values for the same (entity, relationship\_type) pair (e.g., (user, works\_at, Google) vs. (user, works\_at, Meta)), a conflict is raised.

**Classification.** Conflicts are triaged by severity:

Type	Relations	Priority	Action
Identity Fact	works_at, lives_in, birthday, spouse	High	Ask immediately
Preference	likes, favorite, prefers, dislikes	Medium	Ask naturally
Minor	All others	Low	Keep both

**Resolution strategies:** - **keep\_a / keep\_b:** One memory wins; the other is tombstoned - **keep\_both:** Both memories are retained (marked non-conflicting) - **merge:** A new memory is created combining information from both; originals are tombstoned

Conflict records are first-class data structures, persisted and replicated across devices via the oplog.

# 6. Replication and Synchronization

## 6.1 Local-First Architecture

YantrikDB implements a local-first architecture where each device maintains a complete, independently-functional replica. Synchronization is opportunistic—devices sync when connected, but operate fully offline.

## 6.2 CRDT Semantics

Each data type uses an appropriate CRDT strategy:

Data	CRDT Type	Semantics
Memories	Add-Wins Set	UUIDv7 uniqueness; INSERT OR IGNORE
Edges	Last-Writer-Wins Register	HLC timestamp on (src, dst, rel_type)
Entities	Derived state	Recomputed from edges
Tombstones	Tombstone-wins	Deletion is irreversible
Consolidation members	Set-union	Membership tracking

## 6.3 Operation Log

All mutations are recorded in an append-only operation log (oplog):

```
OplogEntry {
    op_id: UUIDv7,
    op_type: record | relate | consolidate | forget | ...,
    timestamp: f64,
    target_rid: Option<String>,
    payload: JSON,
    actor_id: String,          // Device identity
    hlc: Vec<u8>,             // Hybrid Logical Clock
    origin_actor: String,     // Source device
}
```

Hybrid Logical Clocks [Kulkarni et al., 2014] provide causal ordering without synchronized physical clocks. The compound sync cursor (`hlc`, `op_id`) enables incremental synchronization: devices exchange only operations newer than their last sync point, filtered to exclude self-originated operations.

## 6.4 Sync Protocol

1. Device A sends its sync cursor to Device B
2. Device B returns all oplog entries after the cursor, excluding entries originated by A
3. Device A applies entries idempotently (checking `op_id` existence before applying)
4. Device A advances its sync cursor
5. Symmetric exchange completes bidirectional sync

The protocol converges in a single round-trip for two-device scenarios and extends to multi-device topologies through transitive sync.

# 7. Proactive Cognition Loop

## 7.1 Motivation

Existing memory systems are purely reactive—they respond to queries but never initiate action. YantrikDB’s proactive cognition loop runs periodically in the background (configurable: 5-30 minutes based on activity), processing the memory store to generate behavioral triggers.

## 7.2 Trigger Types

The cognition loop generates five categories of triggers:

**Decay Review.** When a memory’s current strength falls below a threshold, the system generates a trigger suggesting the host application review the memory with the user before it fades:

$$\text{urgency} = \text{importance} \times (1 - \text{decay\_ratio})$$

**Consolidation Ready.** When the number of unconsolidated episodic memories in a topic exceeds a threshold, the system suggests consolidation.

**Conflict Escalation.** Unresolved conflicts above a severity threshold generate escalating triggers.

**Pattern Detection.** The cognition loop runs five pattern mining algorithms:

1. *Co-occurrence*: Entity pairs appearing together in  $\geq N$  memories
2. *Temporal clusters*: Memories clustering around specific times of day or week
3. *Valence trends*: Sustained positive or negative emotional patterns over a time window
4. *Topic clusters*: Semantically similar memories forming thematic groups
5. *Entity hubs*: Entities with high graph degree, indicating central importance

**Goal Tracking.** User-stated goals are tracked against relevant memories, generating progress or stagnation triggers.

## 7.3 Trigger Lifecycle

Triggers are persisted in the database with status tracking:

pending → delivered → (dismissed | expired)

Delivered triggers are available to the host application (e.g., an AI companion) for incorporation into proactive messages. Triggers expire after a configurable timeout (default: 48 hours). The entire trigger lifecycle is replicated via the oplog, ensuring multi-device consistency.

# 8. Evaluation

## 8.1 Benchmark Design

I evaluate YantrikDB on a synthetic benchmark designed to test retrieval quality across diverse cognitive dimensions. The benchmark consists of:

- **32 memories** across **8 simulated sessions** spanning 25 days, covering a software engineering project lifecycle (kickoff, debugging, architecture decisions, personal preferences, conflicts, deployment, retrospective)
- **40 golden queries** organized into **12 categories**: direct semantic, entity-person, entity-tech, temporal, emotional/valence, conflict/change, procedural, multi-hop, performance/metrics, team/organizational, broad synthesis, and deployment

Each query specifies expected matching memories, enabling computation of standard information retrieval metrics. The benchmark is intentionally challenging: it includes queries requiring graph traversal (entity-centric), temporal reasoning, emotional alignment (valence), contradiction awareness, and multi-hop reasoning.

## 8.2 Metrics

I report three standard metrics:

- **Recall@K**: Fraction of expected memories retrieved in the top-K results (K=10)
- **Precision@K**: Fraction of retrieved results that are relevant
- **Mean Reciprocal Rank (MRR)**:  $1/\text{rank}$  of the first relevant result

YantrikDB’s evaluation is *consolidation-aware*: a consolidated memory that contains a source memory in its provenance chain counts as a hit for the source memory’s expected queries. This reflects the real-world scenario where episodic details may have been merged into semantic summaries.

## 8.3 Baselines

I compare three configurations:

1. **Vector-only**: Pure cosine similarity scoring ( $w_{\text{sim}} = 1.0$ , all other weights = 0)
2. **Multi-signal (no graph)**: Full composite scoring without graph expansion
3. **Multi-signal + graph**: Full composite scoring with graph-expanded retrieval

## 8.4 Results

The multi-signal scoring with graph expansion outperforms pure vector similarity across all categories. Key findings:

- **Entity-centric queries** show the largest improvement from graph expansion, as entity-connected memories are retrieved even when embedding similarity is low

- **Temporal queries** benefit from the recency signal, correctly prioritizing recent memories over semantically similar but older ones
- **Valence queries** demonstrate query-aware emotional alignment, surfacing emotionally relevant memories
- **Conflict queries** correctly retrieve contradicting memories when the conflict detection system has flagged them
- **Multi-hop queries** leverage graph traversal to connect memories through shared entities

All metrics satisfy mathematical invariants: recall, precision, and MRR all fall within  $[0, 1]$ , and aggregate metrics equal the mean of per-query values. The evaluation is deterministic and idempotent—repeated runs produce identical results.

## 9. Implementation

### 9.1 System Architecture

YantrikDB is implemented in approximately 16,000 lines of Rust, organized into four modules:

- **base:** Types, schema, scoring, error handling
- **engine:** Core CRUD operations, recall pipeline, HNSW integration
- **cognition:** Consolidation, triggers, pattern mining
- **distributed:** Replication, synchronization, conflict detection

Python bindings are provided via PyO3, exposing the full API as a native Python module. The engine compiles to a single shared library (~5MB) with no runtime dependencies beyond SQLite (bundled).

### 9.2 Storage

All data resides in a single SQLite database file. YantrikDB leverages SQLite's WAL mode for concurrent read access, FTS5 for full-text search, and JSON1 for structured metadata storage. The HNSW index is maintained in-memory and rebuilt on startup from persisted embeddings (< 1 second for 100K memories).

### 9.3 Performance Characteristics

- **Record:** < 1ms (SQLite insert + HNSW node insertion)
- **Recall (10K memories):** < 10ms (HNSW search + scoring + ranking)
- **Think cycle:** < 100ms (consolidation check + trigger generation + pattern scan)
- **Sync (1000 ops):** < 50ms (oplog serialization + idempotent apply)
- **Memory footprint:** ~50MB for engine + 100K memories with HNSW index

# 10. Discussion

## 10.1 Limitations

**Small model evaluation.** The synthetic benchmark, while comprehensive in category coverage, uses 32 memories. Evaluation at 100K+ scale would better characterize HNSW recall quality and scoring performance under load.

**LLM-dependent consolidation.** The current consolidation algorithm uses extractive summarization. Abstractive summarization via an LLM would produce higher-quality consolidated memories but introduces model dependency.

**Single-writer assumption.** While YantrikDB supports multi-device sync, the CRDT semantics assume cooperative devices. Byzantine fault tolerance is not addressed.

## 10.2 Comparison with Prior Work

System	Decay	Graph	Conflicts	Proactive	Embedded	Replication
Vector DBs	No	No	No	No	Some	Cloud
Generative Agents	Partial	No	No	Reflection	No	No
MemGPT	No	No	No	No	No	No
LangChain Memory	No	No	No	No	No	No
<b>YantrikDB</b>	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>	<b>CRDT</b>

YantrikDB is the only system that provides all six capabilities simultaneously.

## 10.3 Applications

**AI Companions.** YantrikDB serves as the memory substrate for personal AI companions that remember user preferences, track relationships, and proactively surface relevant context.

**Autonomous Agents.** Multi-agent systems can use YantrikDB instances as individual agent memories, with the knowledge graph capturing inter-agent relationship dynamics.

**Personal Knowledge Bases.** Beyond AI, YantrikDB can function as a personal memory tool—recording insights, detecting patterns, and surfacing connections across a lifetime of accumulated knowledge.

# 11. Conclusion

I have presented YantrikDB, a cognitive memory engine that unifies vector search, knowledge graphs, temporal indexing, decay dynamics, and proactive cognition in a single embedded database. The key contributions are:

1. **Relevance-gated importance amplification:** A scoring function that prevents high-importance memories from dominating low-relevance queries, using a sigmoid gate controlled by embedding similarity.
2. **Contradiction-aware memory lifecycle:** Conflicts between memories are detected, classified by severity, and presented as first-class data structures for resolution—all replicated across devices via CRDTs.
3. **Proactive cognition loop:** An autonomous background process that detects patterns, surfaces decaying memories, and generates behavioral triggers, enabling AI systems to initiate action based on accumulated experience.

YantrikDB demonstrates that AI memory need not be a simple vector store. By drawing on cognitive science and implementing biologically-inspired dynamics, it is possible to build memory systems that more closely mirror the richness of human recall—systems that not only remember, but understand what they remember, and act on it.

The engine is open-source, implemented in Rust, and designed for deployment on resource-constrained devices including mobile phones and single-board computers.

## Patent Notice

Aspects of the systems and methods described in this paper are the subject of US Provisional Patent Application No. 63/991,357, filed February 26, 2026, titled “Cognitive Memory Engine with Instinct-Driven Proactive Behavior, Unified In-Process Companion Runtime, and Contradiction-Aware Adaptive Retrieval.”

## References

- [Atkinson & Shiffrin, 1968] Atkinson, R.C. and Shiffrin, R.M. “Human memory: A proposed system and its control processes.” *Psychology of Learning and Motivation*, 2:89-195.
- [Ebbinghaus, 1885] Ebbinghaus, H. *Über das Gedächtnis: Untersuchungen zur experimentellen Psychologie*. Duncker & Humblot.
- [Kleppmann et al., 2019] Kleppmann, M., Wiggins, A., van Hardenberg, P., and McGranaghan, M. “Local-first software: You own your data, in spite of the cloud.” *ACM SIGPLAN Notices*, 54(SPLASH Companion):154-178.

- [Kulkarni et al., 2014] Kulkarni, S.S., Demirbas, M., Madappa, D., Avva, B., and Leone, M. “Logical physical clocks.” *International Conference on Principles of Distributed Systems (OPODIS)*, 2014.
- [Malkov & Yashunin, 2018] Malkov, Y.A. and Yashunin, D.A. “Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824-836.
- [Packer et al., 2023] Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S.G., Stoica, I., and Gonzalez, J.E. “MemGPT: Towards LLMs as Operating Systems.” *arXiv preprint arXiv:2310.08560*.
- [Park et al., 2023] Park, J.S., O’Brien, J.C., Cai, C.J., Morris, M.R., Liang, P., and Bernstein, M.S. “Generative Agents: Interactive Simulacra of Human Behavior.” *UIST 2023*.
- [Shapiro et al., 2011] Shapiro, M., Preguiça, N., Baquero, C., and Zawirski, M. “Conflict-free replicated data types.” *Symposium on Self-Stabilizing Systems*, 2011.
- [Squire & Alvarez, 1995] Squire, L.R. and Alvarez, P. “Retrograde amnesia and memory consolidation: a neurobiological perspective.” *Current Opinion in Neurobiology*, 5(2):169-177.
- [Tulving, 1972] Tulving, E. “Episodic and semantic memory.” In *Organization of Memory*, pp. 381-403. Academic Press.