# memlocal

## A Local-First Cognitive Memory Architecture for On-Device AI Agents

Sirsha Chakraborty
memlocal.dev

28 March 2026
White Paper v1.0

https://github.com/memlocal/memlocal_core

## Abstract

We present **memlocal**, an open-source, local-first memory architecture designed to give AI agents persistent, structured recall on resource-constrained devices such as smartphones. Inspired by models from cognitive psychology, memlocal organises information into eight memory types (episodic, semantic, factual, procedural, social, spatial, prospective, and affective) and stores them in CozoDB, an embedded Datalog database that unifies vector search (HNSW), full-text retrieval (BM25), and recursive graph traversal in a single engine. The retrieval pipeline employs multi-channel candidate generation, cross-encoder reranking, query-complexity-adaptive context assembly, and session-window expansion to maximise recall precision. Evaluated on the LoCoMo long-conversation benchmark, memlocal achieves an 80.0% pass rate (score $\geq$ 3/5) with an average LLM score of 4.21/5 across 105 questions spanning single-hop, multi-hop, temporal, and adversarial categories. The core engine is implemented in Rust with a platform-agnostic FFI boundary, enabling deployment on iOS, Android, and desktop through thin Flutter or native bindings. All LLM, embedding, and reranker calls are mediated by pluggable provider traits, allowing operators to choose between cloud APIs and fully local inference. This paper details every algorithm, data structure, and design decision in the architecture.

## 1. Introduction

Large language models possess no persistent memory beyond their context window. When a conversation ends, everything discussed is lost. This fundamental limitation prevents AI assistants from building lasting relationships with users, tracking preferences over time, or recalling prior interactions. Several approaches have emerged to address this gap: retrieval-augmented generation (RAG) systems, vector databases, and purpose-built memory layers. However, nearly all existing solutions share two constraints: they require cloud infrastructure, and they treat memory as a flat collection of text chunks rather than structured, typed knowledge.

memlocal takes a different approach. It is designed as a **local-first** system where all data remains on the user's device. The entire database, vector indices, and knowledge graph run inside a single embedded process. No server is required. This design choice is motivated by three concerns: privacy (personal memories should not traverse the network), latency (on-device retrieval eliminates round-trip delays), and reliability (the system works offline).

The architecture draws from cognitive psychology's multi-store model of human memory. Rather than treating all information uniformly, memlocal classifies extracted facts into eight distinct memory types, each with its own decay characteristics, importance weighting, and retrieval behaviour. A conversation about a friend's birthday party produces episodic memories (the event), factual memories (the friend's name), affective memories (how the user felt), and social memories (the relationship dynamics)—all linked through a knowledge graph.

The storage layer is CozoDB, an embedded database engine that combines Datalog-based relational queries with HNSW vector search and BM25 full-text indexing in a single process. This eliminates the common pattern of running separate databases for structured queries, vector similarity, and keyword search. CozoDB's support for recursive Datalog enables graph traversal queries that would require multiple round-trips in conventional architectures.

This paper makes the following contributions: (1) a cognitive-science-inspired memory taxonomy with formal scoring functions for importance, decay, and confidence; (2) a multi-channel retrieval pipeline that combines semantic, lexical, graph, and structured search with cross-encoder reranking; (3) a query-complexity classifier that adapts retrieval depth and context assembly strategy per query; (4) an evaluation on the LoCoMo benchmark demonstrating competitive performance with cloud-scale systems; and (5) a fully open-source Rust implementation with FFI bindings for mobile deployment.

## 2. Related Work

### 2.1 Memory Systems for LLM Agents

MemGPT (Packer et al., 2023) introduced a virtual memory hierarchy for LLMs, paging information between a fixed context window and an external store. MemoryMachine (Li et al., 2025) extended this with an agentic tool-calling loop where the LLM itself decides when and what to retrieve. Mem-U (Safdari et al., 2025) proposed user-modelling through memory, achieving 88.31% on LoCoMo with structured user profiles. Letta (formerly MemGPT v2) added multi-agent orchestration and a REST API. All of these systems assume cloud deployment and use server-side vector databases.

SmartSearch (Stelmakh et al., 2025) demonstrated that raw text with excellent ranking can outperform structured memory systems, achieving 93.5% on LoCoMo using grep combined with ColBERT and CrossEncoder reranking with Reciprocal Rank Fusion (RRF). This finding significantly influenced memlocal's design: we prioritise reranking quality over structural complexity.

### 2.2 Embedded and On-Device Databases

SQLite is the most widely deployed embedded database but lacks native vector search. Extensions like sqlite-vec add HNSW indexing but require separate full-text search configuration. Chroma and LanceDB provide embedded vector databases but lack graph capabilities. CozoDB uniquely combines Datalog relational queries, HNSW vector indices, BM25 full-text search, and MinHash (LSH) near-duplicate detection in a single embedded library with a SQLite storage backend. This convergence eliminates the need for multiple database processes on a mobile device.

### 2.3 Cognitive Memory Models

Tulving's (1972) distinction between episodic and semantic memory remains foundational. The Atkinson-Shiffrin model (1968) proposed a multi-store architecture with sensory, short-term, and long-term stores. Working memory theory (Baddeley, 2000) refined the short-term component. memlocal implements a computational analogue of these models, using sensory buffers for raw input, a working memory context assembler, and typed long-term storage with decay functions calibrated to each memory type.

## 3. Architecture Overview

memlocal is structured as a three-layer system mirroring the Atkinson-Shiffrin multi-store model of human memory. The outermost layer handles transient sensory input; the middle layer maintains a bounded working memory context; and the innermost layer provides persistent, structured long-term storage. Figure 1 illustrates the data flow.

### 3.1 System Layers

**Sensory Buffer.** Raw input (conversation turns, document chunks, API responses) enters a time-limited sensory buffer implemented as a ring buffer with a configurable TTL (default 5000ms) and capacity (default 100 items). Items exceeding the TTL are evicted before processing. This mirrors the brief persistence of sensory memory in human cognition, filtering noise before deeper processing.

**Working Memory.** The WorkingMemory struct assembles a context block from multiple sources: keyword-matched facts, reranked evidence, session summaries, knowledge-graph triples, user profile data, and prospective reminders. For single-hop queries, it produces a flat ranked list; for complex queries, it produces a sectioned context with prioritised evidence channels. The output is a formatted string injected into the LLM's system prompt, enabling single-call answer generation without iterative tool use.

**Long-Term Memory.** The persistent store in CozoDB holds typed memory items, knowledge-graph edges, session summaries, semantic triples, and user profiles. Each memory item carries a 1536-dimensional embedding vector, full-text-indexed content, metadata (confidence, access count, reinforcement count), and temporal validity markers. CozoDB's native Validity schema provides time-travel semantics, allowing queries against the state of memory at any past timestamp.

### 3.2 Implementation Stack

The core engine (memlocal_core) is written in Rust and compiled as a C-compatible dynamic library. Platform SDKs (currently Flutter/Dart via FFI, with planned Kotlin and Swift bindings) implement the EmbeddingProvider, LlmProvider, and RerankerProvider traits using their native HTTP stacks. This separation means the core engine has no network dependencies—all I/O is delegated to the platform layer.

## 4. Memory Model

### 4.1 Memory Type Taxonomy

Each extracted fact is classified into one of eight types, inspired by Tulving's memory taxonomy and extended for computational use:

| Type | Description | Decay λ | Half-life |
|---|---|---|---|
| **Episodic** | Events, experiences, meetings | 0.005 | ~139 days |
| **Factual** | Preferences, personal details, stable facts | 0.002 | ~347 days |
| **Semantic** | General knowledge, learned concepts | 0.002 | ~347 days |
| **Procedural** | Workflows, routines, how-to knowledge | 0.002 | ~347 days |
| **Social** | Relationships, people, team dynamics | 0.003 | ~231 days |
| **Spatial** | Locations, places, addresses | 0.005 | ~139 days |
| **Prospective** | Reminders, future intentions | 0.020 | ~35 days |
| **Affective** | Emotions, moods, feelings | 0.005 | ~139 days |

*Table 1: Memory type taxonomy with exponential decay parameters.*

## 4.2 Scoring Functions

Each memory item is scored along four dimensions that combine into an importance metric used for retrieval ranking and consolidation decisions.

### 4.2.1 Exponential Time Decay

Inspired by Ebbinghaus's forgetting curve, each memory type decays at a rate λ calibrated to its expected persistence. The decay factor for a memory updated t days ago is:

$$D(t) = e^{-\lambda t} \quad (1)$$

where λ is the type-specific decay rate from Table 1. Factual and semantic memories (half-life ~347 days) persist far longer than prospective reminders (half-life ~35 days), reflecting the cognitive reality that stable facts endure while transient intentions fade.

### 4.2.2 Confidence Score

The extraction LLM assigns each memory a confidence value $c \in [0, 1]$ based on the explicitness of the source statement. Explicit, unambiguous statements receive $c > 0.8$; reasonable inferences receive 0.5–0.8; speculative content receives $c < 0.5$ and is typically not stored (the default threshold is 0.3). During retrieval, confidence is applied with square-root dampening to avoid over-penalising moderate-confidence items:

$$S\_conf(s, c) = s \cdot \sqrt{c} \quad (2)$$

### 4.2.3 Reinforcement

When a new memory duplicates or updates an existing one, the existing memory's reinforcement count r is incremented. This mirrors the spacing effect in human memory: repeatedly encountered facts are strengthened. The reinforcement factor is normalised logarithmically:

$$R(r) = \ln(1 + r) / \ln(1 + r\_max) \quad (3)$$

where $r_{max} = 10$ is the saturation constant. This ensures diminishing returns: the first few reinforcements matter most, while further repetitions provide progressively smaller boosts.

### 4.2.4 Composite Importance

The final importance score used for consolidation and retrieval-priority decisions is a weighted combination:

$$I = 0.25c + 0.25R(r) + 0.20A(a) + 0.30D(t) \quad (4)$$

where c is confidence, R(r) is normalised reinforcement, A(a) = ln(1+a)/ln(1+100) is normalised access count, and D(t) is time decay. The weights reflect the design priority of recency (30%) balanced with reliability (confidence 25%), social proof (reinforcement 25%), and usage patterns (access 20%).

# 5. Storage Layer: CozoDB

## 5.1 Why CozoDB

The choice of CozoDB as the storage engine is driven by a single constraint: on a smartphone, we cannot run multiple database processes. A typical RAG system requires a vector database (Pinecone, Qdrant), a

keyword search engine (Elasticsearch), and a graph database (Neo4j)—three separate services. CozoDB collapses all three into a single embedded library with a SQLite storage backend.

CozoDB's key capabilities exploited by memlocal are: (1) HNSW vector indices for semantic similarity search over 1536-dimensional embeddings; (2) BM25 full-text indices for keyword and entity retrieval; (3) MinHash (LSH) indices for near-duplicate detection during deduplication; (4) recursive Datalog for multi-hop graph traversal in a single query; (5) native Validity schema for time-travel queries; and (6) a C FFI for embedding in Rust, which in turn exposes a C-compatible dynamic library to Flutter, Kotlin, and Swift.

### 5.2 Schema Design

The core schema consists of seven relations. The primary `mem_items` relation stores memory items with a Validity key column enabling CozoDB's native time-travel semantics. Each item carries a 1536-dimensional embedding vector, full-text-searchable content, typed classification, speaker attribution, temporal validity markers (event_start, event_end), and a JSON metadata blob containing confidence, access count, and reinforcement count.

The `mem_edges` relation forms a directed knowledge graph linking related memories with typed relations (relates_to, contradicts, updates, causes, temporal_sequence). Edges are created automatically when a newly stored memory has cosine similarity > 0.5 with an existing memory. The `mem_triples` relation stores subject-predicate-object triples extracted alongside each memory, providing structured knowledge that can be queried independently of free-text search. The `mem_summaries` relation holds per-session narrative summaries with their own vector index for session-level retrieval.

### 5.3 Index Configuration

Five indices are maintained: (1) `mem_vec_idx`: HNSW over mem_items embeddings with M=16, ef_construction=100, cosine distance; (2) `mem_fts_idx`: BM25 full-text index over mem_items content using CozoDB's built-in tokeniser; (3) `mem_lsh_idx`: MinHash locality-sensitive hashing for near-duplicate detection with n_perm=200, n_gram=3, target_threshold=0.5; (4) `mem_triples_fts`: BM25 index over the concatenated subject+predicate+object fields of mem_triples; and (5) `mem_summaries_vec`: HNSW over session summary embeddings for session-level semantic search.

## 6. Ingestion Pipeline

### 6.1 LLM-Driven Extraction

Raw text (conversation transcripts, documents, API responses) is processed by an extraction LLM (currently Claude Haiku 4.5) with a structured system prompt that demands JSON output containing: (a) an array of typed memories, each with content, classification, confidence, speaker attribution, semantic triple, contradiction pattern, and optional temporal anchors; (b) an array of literal observations preserving exact proper nouns and specific details; (c) a 2–3 sentence session summary; and (d) a list of detected speakers.

The prompt enforces several critical constraints. Proper nouns must be preserved exactly as spoken: "Nothing is Impossible" must not become "a book"; "Matt Patterson" must not become "a performer." Each memory must include a semantic triple (subject, predicate, object) with named entities as subjects, never

pronouns. Speaker identity must be preserved—in multi-party conversations, each fact is attributed to the person who stated or is described by it.

### 6.2 Deduplication and Contradiction Resolution

Before storing, each candidate memory is compared against existing memories using two mechanisms. First, the LSH index identifies near-duplicates (Jaccard similarity > 0.5) based on the content's MinHash signature. Second, a deduplication LLM call classifies each match as ADD (new), UPDATE (supersedes existing), SKIP (duplicate), or CONTRADICTION (conflicts). For UPDATE actions, the existing memory's reinforcement count is incremented and its content is replaced. For CONTRADICTION, the newer memory takes precedence and the older one's validity is closed via CozoDB's Validity mechanism.

### 6.3 Automatic Graph Construction

After storage, each new memory is embedded and compared against the top-3 semantically similar existing memories. Pairs with cosine similarity exceeding 0.5 are connected via a `relates_to` edge in the knowledge graph. This produces an emergent graph structure where thematically related memories cluster naturally, enabling multi-hop traversal during retrieval.

## 7. Retrieval Pipeline

The retrieval pipeline converts a natural-language query into a ranked set of memory items ready for LLM consumption. It operates in five stages: query classification, multi-channel candidate generation, score fusion, cross-encoder reranking, and context assembly.

### 7.1 Query Complexity Classification

Each query is classified into one of four complexity levels, each triggering a different retrieval strategy with respect to candidate pool size and context budget:

| Complexity | Pool Size | Context Limit | Trigger |
|---|---|---|---|
| **SingleHop** | 80 | 15 | Default / simple factual |
| **MultiHop** | 120 | 18 | when/how long/both |
| **Temporal** | 140 | 24 | before/after/during/date |
| **OpenEnded** | 100 | 15 | would/could/should |

*Table 2: Adaptive retrieval strategy parameters per query complexity.*

### 7.2 Multi-Channel Candidate Generation

Multiple query variants are generated from the original query: the raw question, a keyword-reduced form, and an entity-focused form. For each variant, candidates are retrieved through six parallel channels:

- **Semantic search (HNSW):** Cosine similarity over 1536-d embeddings, filtered by memory type. Each of the eight memory types is searched independently with configurable per-type budgets (e.g., 10 episodic, 8 factual, 3 affective).

- **Full-text search (BM25):** Okapi BM25 scoring over tokenised content. Two BM25 channels operate: entity-based (query entities as search terms) and reserved (exact phrase matches for proper nouns).
- **Graph traversal (Datalog):** Starting from the top-5 semantic search results as seeds, CozoDB's recursive Datalog traverses the knowledge graph up to 2 hops deep, retrieving connected memories.
- **Triple search (FTS):** BM25 search over the subject-predicate-object fields of the triples relation, enabling structured fact lookup by entity name.
- **Session-window expansion:** When a raw conversation line is retrieved, the ±2 adjacent turns from the same session are pulled into the candidate pool. This captures facts that appear in neighbouring turns (e.g., an artist's name mentioned in the turn after a concert description).
- **Speaker-filtered search:** When the query mentions a known speaker by name, an additional semantic search is performed filtered to that speaker's memories only.

All channels contribute to a single unified candidate pool. Candidates are deduplicated by memory ID. Each candidate's provenance is tracked (which channels retrieved it) for diagnostic purposes.

## 7.3 Score Fusion

Candidates from different channels carry scores on incomparable scales (cosine similarity in [0,1], BM25 scores in [0, ∞), graph hop distances). The fusion strategy first sorts all candidates by their raw scores, then passes the full pool to the cross-encoder reranker, which produces globally comparable relevance scores.

## 7.4 Cross-Encoder Reranking

The reranker takes the top-K candidates (K = pool size, typically 80–140) and scores each against the original query using a cross-encoder model. The current implementation uses the Jina Reranker v2 API (jina-reranker-v2-base-multilingual), which processes query-document pairs through a cross-attention transformer to produce relevance scores in [0, 1].

Critically, the reranker receives the *full pool* of candidates, not just the top context_limit. This was identified as a key fix during development: when only context_limit candidates were reranked, strong evidence at BM25 ranks 10–50 was silently discarded before the cross-encoder could evaluate it. The reranker's scores then determine the final ranking, and only the top context_limit items (8–24 depending on query complexity) are passed to context assembly.

**Alternative local rerankers.** For fully offline operation, FlashRank (an ONNX-based reranker, ~33MB model size) can replace the Jina API. On mobile devices, FlashRank achieves 400–800ms latency via flutter_onnxruntime. For self-hosted scenarios, gte-reranker-modernbert-base can be deployed on a Cloud Run instance.

## 7.5 Context Assembly

The final stage assembles the top-ranked memories into a context block injected into the LLM's system prompt. The assembly strategy adapts to query complexity.

For **SingleHop** queries, a flat ranked list is produced with no section headers, no type labels, no relevance scores, and no source blocks. Each item includes only the speaker name (if known), the content, and the

approximate date. This minimalist format (~1,000–1,500 characters) prevents the LLM from over-including tangentially relevant information—a failure mode observed when the context was 5–6x larger.

For **MultiHop, Temporal, and OpenEnded** queries, a sectioned context is produced with: KEY FACTS (BM25 reserved matches), KEY FACTS from triples (structured knowledge grouped by speaker), TOP EVIDENCE (reranked candidates), RAW CONVERSATION EXCERPTS, SESSION CONTEXT (session summaries), and Important Memories. This multi-section format gives the LLM multiple evidence pathways to cross-reference when answering complex questions.

## 8. Iterative Retrieval

For queries where the initial retrieval may be insufficient, memlocal supports a two-round iterative retrieval pattern inspired by MemoryMachine's agent-mode approach. After the first retrieval round, the LLM is asked to evaluate context sufficiency and generate 1–3 follow-up search queries if gaps are detected. These refined queries trigger a second retrieval round, and the results are merged with round-1 candidates before final ranking. This mechanism is particularly effective for multi-hop and temporal questions where the answer depends on connecting information across multiple sessions.

## 9. Consolidation

Over time, the memory store accumulates redundant episodic memories about the same topics. The consolidation engine periodically (or on-demand) identifies clusters of related episodic memories and synthesises them into higher-level semantic summaries. A dedicated LLM prompt instructs the model to produce a single concise, third-person factual sentence capturing the key pattern or insight. The original episodic memories are retained but their importance scores are adjusted downward, allowing the consolidated semantic memory to surface preferentially during retrieval.

## 10. Local-First Deployment Considerations

### 10.1 On-Device Feasibility

The memlocal_core Rust library compiles to a ~4MB shared object (.so/.dylib) including CozoDB. A typical memory store for 1,000 memories with 1536-dimensional embeddings occupies ~25MB on disk (SQLite). The HNSW index adds ~12MB. Total on-device footprint for a year of daily conversations is under 50MB—well within smartphone storage budgets.

### 10.2 API Dependencies and Local Alternatives

The current implementation uses three cloud APIs, each replaceable with a local alternative:

| Function | Cloud API | Local Alternative | Trade-off |
|---|---|---|---|
| **LLM** | Claude Haiku 4.5 | Gemma 3 4B via llama.cpp | Lower extraction quality |
| **Embeddings** | text-embedding-3-small (1536d) | all-MiniLM-L6-v2 via ONNX (384d) | Smaller embedding, reduced recall |

| | | | |
|---|---|---|---|
| **Reranker** | Jina Reranker v2 | FlashRank ONNX (~33MB) | 400-800ms latency on mobile |

*Table 3: Cloud APIs and their local-first alternatives.*

The architecture's provider-trait abstraction means switching between cloud and local is a configuration change, not a code change. The EmbeddingProvider, LlmProvider, and RerankerProvider traits are implemented by the platform layer, which can instantiate either an HTTP client for cloud APIs or an ONNX/llama.cpp runtime for local inference.

### 10.3 Privacy Architecture

In full local-first mode, no data leaves the device. Embeddings are computed locally, memories are stored in an on-device SQLite file, and retrieval is performed entirely in-process. When cloud APIs are used for extraction or reranking, the architecture ensures that raw conversation text is sent only for processing and is not retained by the API provider (subject to provider data policies). The FFI boundary guarantees that the platform layer controls all network I/O—the core engine itself makes no network calls.

## 11. Evaluation

### 11.1 Benchmark: LoCoMo

We evaluate memlocal on the LoCoMo (Long Conversation Memory) benchmark, which tests a memory system's ability to answer questions about extended multi-session conversations between two speakers. The benchmark contains 105 questions across four categories: single-hop (direct factual recall), multi-hop (requiring synthesis of information from multiple sessions), temporal (requiring temporal reasoning about when events occurred), and adversarial (questions designed to trick systems with speaker-swapped or misleading phrasing).

### 11.2 Experimental Setup

The test conversation (conv-30) comprises 369 turns across multiple sessions, from which memlocal extracted 1,017 memories during ingestion (processing time: 1,082 seconds). The extraction LLM was Claude Haiku 4.5. The embedding model was OpenAI text-embedding-3-small (1536 dimensions). The reranker was Jina Reranker v2. The answer generation LLM was also Claude Haiku 4.5. Each question was evaluated by an LLM judge on a 1–5 scale, with scores ≥3 counted as passes.

### 11.3 Results

| Category | Count | J-score | Avg LLM | KW F1 | Pass % |
|---|---|---|---|---|---|
| **Single-hop** | 11 | 27.3% | 3.00 | 0.163 | 45.5% |
| **Multi-hop** | 26 | 92.3% | 4.65 | 0.131 | 96.2% |
| **Temporal** | 44 | 72.7% | 4.14 | 0.172 | 77.3% |
| **Adversarial** | 24 | 41.7% | 4.42 | 0.108 | 83.3% |
| **Overall** | 105 | 65.7% | 4.21 | 0.146 | 80.0% |

*Table 4: LoCoMo benchmark results for memlocal.*

### 11.4 Failure Analysis

Of the 21 failed questions (score < 3), the dominant failure patterns are: SPEAKER_SWAP (13 cases)—the model attributes a fact to the wrong speaker, particularly in adversarial questions designed to test this; WRONG_FACT (11 cases)—the model includes incorrect events or details, often due to over-inclusion from noisy context; INCOMPLETE (9 cases)—the model misses part of the ground truth, typically when evidence ranks just outside the context window; and LLM_REFUSE (3 cases)—the model declines to answer despite evidence being present. Root cause analysis traced 11 failures to context truncation or ranking issues, 12 to answer generation quality, 7 to adversarial/speaker handling, 4 to retrieval misses, and 2 to evidence-in-context-but-refused scenarios.

### 11.5 Performance Trajectory

Development involved three major benchmark iterations that isolated the impact of each architectural change:

| Version | Changes | Pass Rate |
|---|---|---|
| **v1** | Baseline: BM25 + semantic search, flat context | 55.8% |
| **v2** | + Speaker attribution, triples, session summaries, iterative retrieval | 62.8% |
| **v3** | + Jina reranker, query classification, adaptive context | 66.3% |
| **v4** | + Pool fix, flat single-hop context, session windows, extraction v2 | 80.0% |

*Table 5: Performance trajectory across development iterations.*

## 12. Discussion

### 12.1 Ranking Beats Structure

A key finding from both the SmartSearch literature and our own development is that retrieval ranking quality dominates structural memory organisation for question-answering tasks. The single highest-impact change in memlocal's development was fixing the reranker pool size—a one-line bug that prevented the cross-encoder from seeing most candidates. Before this fix (v3), the reranker only scored the top 8 candidates; after (v4), it scored all 80. This alone moved the pass rate from 66.3% to ~72% before any other changes. The implication for memory architecture design is clear: invest first in reranking infrastructure, then in structural enhancements.

### 12.2 Context Size as a Design Parameter

We discovered that for single-hop factual questions, smaller context produces better answers. When the LLM received ~8,000 characters of context (the v3 multi-section format), it listed every tangentially related fact, producing over-inclusive answers. Reducing to ~1,000–1,500 characters (flat ranked list) forced the LLM to be selective, improving precision. However, for multi-hop and temporal questions, the larger multi-section format remains essential because the LLM needs to cross-reference information across sessions. The adaptive context assembly strategy—flat for SingleHop, sectioned for everything else—captures this trade-off.

### 12.3 Limitations

Single-hop accuracy (45.5% pass rate) remains the weakest category. The primary bottleneck is now the answer generation LLM (Haiku 4.5), which occasionally hedges, over-includes, or misattributes facts even when correct evidence is in context. Upgrading to a Sonnet-class model for answer generation would likely push single-hop past 70%. The adversarial category (83.3% pass rate but 41.7% J-score) reveals that the system handles trick questions reasonably but its verbatim keyword overlap with ground truth is low, resulting in poor automated metrics despite correct semantic answers.

## 13. Future Work

Several directions remain for future development. First, porting the Flutter/Dart SDK to Kotlin (Android) and Swift (iOS) native bindings to enable direct integration without the Flutter runtime. Second, implementing fully local inference using quantised models (Gemma 3 4B via llama.cpp for extraction, all-MiniLM-L6-v2 via ONNX for embeddings) to achieve zero-network operation. Third, adding Reciprocal Rank Fusion (RRF) as explored by SmartSearch to combine BM25 and cross-encoder scores more formally. Fourth, exploring multi-agent memory sharing where multiple agents operating on the same device can share a common memory store with access-controlled views. Fifth, integrating with the Model Context Protocol (MCP) to expose memlocal as a tool server for any MCP-compatible client.

## 14. Conclusion

memlocal demonstrates that a cognitively-inspired, local-first memory architecture can achieve competitive performance with cloud-scale memory systems on standard benchmarks. By combining CozoDB's unified database capabilities with multi-channel retrieval, cross-encoder reranking, and adaptive context assembly, the system achieves an 80.0% pass rate on the LoCoMo benchmark—with the entire storage and retrieval pipeline running in a single embedded process suitable for smartphone deployment. The open-source Rust implementation, provider-trait abstraction, and FFI boundary make it practical for integration into mobile applications where privacy, latency, and offline capability are paramount. The architecture's modular design allows operators to choose their position on the cloud-to-local spectrum, from fully cloud-backed (maximum quality) to fully on-device (maximum privacy), without changing the core engine.

## References

[1] Atkinson, R. C., & Shiffrin, R. M. (1968). Human memory: A proposed system and its control processes. *Psychology of Learning and Motivation*, 2, 89–195.

[2] Baddeley, A. (2000). The episodic buffer: A new component of working memory? *Trends in Cognitive Sciences*, 4(11), 417–423.

[3] Tulving, E. (1972). Episodic and semantic memory. In E. Tulving & W. Donaldson (Eds.), *Organisation of Memory*. Academic Press.

[4] Ebbinghaus, H. (1885). Über das Gedächtnis. Duncker & Humblot. Translated as *Memory: A Contribution to Experimental Psychology* (1913).

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

[6] Li, Z., et al. (2025). MemoryMachine: Agentic Long-Term Memory for LLM Agents. *arXiv*.

[7] Safdari, M., et al. (2025). Mem-U: User Modelling Through Memory. *arXiv*.

[8] Stelmakh, I., et al. (2025). SmartSearch: Ranking beats RAG for long-form memory retrieval. *arXiv:2603.15599*.

[9] Malkov, Y. A., & Yashunin, D. A. (2018). Efficient and Robust Approximate Nearest Neighbor using Hierarchical Navigable Small World Graphs. *IEEE Trans. PAMI*, 42(4), 824–836.

[10] Robertson, S., & Zaragoza, H. (2009). The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval*, 3(4), 333–389.

[11] Nogueira, R., & Cho, K. (2020). Passage Re-ranking with BERT. *arXiv:1901.04085*.

[12] CozoDB Project. (2023). CozoDB: A transactional relational database that uses Datalog for query. *https://cozodb.org*.

[13] Jina AI. (2024). Jina Reranker v2: Multilingual cross-encoder for passage reranking. *https://jina.ai*.

[14] Hajishirzi, H., et al. (2024). LoCoMo: Long Conversation Memory Benchmark. *arXiv*.

---