

MEMBRAIN: AGENT-NATIVE MEMORY

— BY AGENTS, FOR AGENTS

BACKGROUND

Large language models have quickly moved beyond simple chatbots into assistants that operate across long-running, multi-session workflows. In these settings, memory becomes a core capability.

This has led to a growing interest in external memory systems. Most existing approaches fall into two camps.

Text-centric systems treat memory as collections of documents. This includes file-based storage like OpenClaw and retrieval-augmented setups. The advantage is straightforward: everything stays in natural language, which aligns well with how models process information and preserves rich semantic detail. But these systems don't explicitly model relationships. Related facts end up scattered across documents, with no clear structure tying them together.

Structure-centric systems take the opposite approach. They represent memory as entities and relationships, often using graph-based designs—for example, a simple knowledge graph triple like (Alice, loves, MemBrain), where two entities are connected through a predefined relation. This makes connections explicit and supports more structured reasoning, like disambiguating entities or tracing dependencies. The trade-off is that meaning gets diluted—breaking information into nodes and edges often strips away useful context, introducing extra steps and potential information loss.

Bridging the Gap in LLM Long-Term Memory: A Paradigm Comparison

Paradigm 1: Text-Centric Memory (e.g., OpenClaw)



Philosophy: "Everything is a File"

Treats memory as a flat collection of natural language documents or vector-based text chunks.

Pros: High Semantic Fidelity & Native Alignment



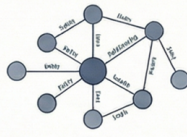
Data stored in original linguistic form, aligns perfectly with LLM's native processing, preserving nuanced details.

Cons: Scattered Context & No Linkage



Information implicitly scattered across files; no explicit mechanism for systematic linkage or cross-context relationships.

Paradigm 2: Graph-Centric Memory (e.g., Graphiti)



Philosophy: "Entities & Relations"

Memory represented as a Knowledge Graph consisting of nodes (entities) and edges (relationships).

Pros: Explicit Structural Modeling & Traceability



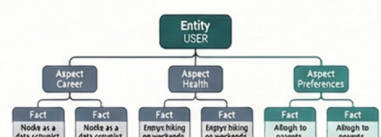
Enables effective entity disambiguation and structured reasoning through clear, traceable relationships.

Cons: Semantic Loss & LLM-Graph Mismatch



Breaking facts into triples often loses meaningful details; LLMs struggle with graph-structured inputs compared to linear text.

Paradigm 3: The MemBrain Approach (Hybrid)



Philosophy: "Entity-Centric Adaptive Semantic Tree"

Hybrid architecture uses entities as anchors while preserving facts in their original natural language form.

Hierarchical Structure: Entity | Aspect | Fact



Facts clustered into thematic "Aspect" nodes under a root Entity, allowing for pruned, efficient searching.

Mechanism: Multi-Agent Pipeline



Specialized LLM agents handle fact extraction, entity resolution (merging versions), and incremental tree maintenance.

Result: Balancing Fidelity and Structure



Provides structural explicitness of a graph without information loss from decomposing text into nodes and edges.

Feature	Text-Centric (OpenClaw)	Graph-Centric (Graphiti)	MemBrain (Hybrid)
Core Structure	Files / Document Chunks	Nodes & Edges (KG)	Adaptive Semantic Tree
Fidelity	High (Original Text)	Low (Decomposed Facts)	High (Natural Language Facts)
Structure	Implicit (Scattered)	Explicit (Graph)	Explicit (Hierarchical)
LLM Alignment	Native / Natural	Mismatched / Lossy	Optimized / Structured

NotebookLM

OVERVIEW OF MEMBRAIN

In practice, text-only memory preserves semantic richness but lacks explicit connections, while graph-based designs enforce structure at the cost of losing detail. MemBrain 1.5 is a step toward addressing both, by rethinking how memory is organized.

The core idea is simple: split the responsibility. Part of memory management is handled by predefined structure, and part is left to agents to decide how to best use it.

MemBrain centers memory around entities and connects information through an adaptive semantic tree. Leaf nodes store facts extracted from conversations, while intermediate nodes group these facts into themes generated by agents. The structure of the tree (its hierarchy, depth, and branching limits) is predefined. The content, however—how facts are clustered, how summaries are written, and where new information is placed—is decided dynamically by agents.

The system offers a multi-agent pipeline.

During **memorization**, agents handle fact extraction, entity resolution, and incremental structure updates. Agents extract facts with explicit entity references and normalized time expressions. Entities evolve over time and are linked to facts through a many-to-many mapping. On top of this, facts are incrementally organized into per-entity hierarchies, forming a natural "entity → aspect → fact" structure that can be directly consumed by models. The combination of flexible fact-entity mapping and hierarchical organization distinguishes MemBrain from traditional graph-based designs.

During **retrieval**, agent involvement scales with query complexity. The system combines full-text search and embedding-based retrieval, along with structure-aware traversal over the entity hierarchy. For more complex queries, an optional agent step generates follow-up queries to fill in missing information. The final output is a structured memory context—organized by entity and theme, and adaptable to different levels of granularity.

Overall, predefined structure handles most of the repetitive organization work, while agents focus on the smaller set of decisions that require semantic judgment. This balance keeps the system efficient while still adapting to complex, evolving memory.

METHODS

Entities and Facts

Traditional graph-based approaches break knowledge into triples like (A, relation, B). In practice, this often fragments information—something that can be expressed clearly in one sentence has to be split across multiple edges, making it harder to store and reconstruct without losing context.

MemBrain takes a different approach. Facts and entities are connected through a many-to-many mapping. A single fact can reference multiple entities, and each entity can appear across many facts.

Facts are stored in natural language, making them easy for models to consume. Each fact is a self-contained semantic unit, with built-in temporal information, and can reference multiple entities through aliases. This avoids the need for decomposition, while a consistent format ensures structural regularity.

Aliases point to entities, whose descriptions can evolve over time as new information is observed. As a result, each fact effectively acts as a renderable template: the fact itself remains stable, but the associated entity information is always up to date. Each entity has a stable identity and maintains a versioned history, so updates don't overwrite past information but extend it. This allows the system to track how an entity's description changes over time. Entities are stored with both a canonical name and a natural language description, enabling efficient lookup and disambiguation.

When facts are retrieved, aliases can be dynamically resolved into entities' canonical names and enriched with the latest entity descriptions. The final text presented to the model is both precise and natural, without requiring additional transformation, while carrying richer and more up-to-date context.

Hierarchical Entity Trees

On top of this, facts are organized into a lightweight hierarchy for each entity. At the root is the entity itself, followed by intermediate clusters (aspects) that group related facts, and finally the facts as leaf nodes. This structure is built incrementally, allowing the system to organize information as it grows. It enables more efficient retrieval by first narrowing down to relevant themes before selecting specific facts.

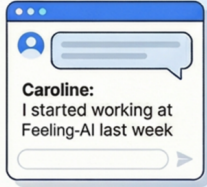
MEMORY INGESTION PIPELINE

MemBrain uses a multi-agent workflow to turn raw messages into structured memory.

MemBrain: The Memory Ingestion Pipeline

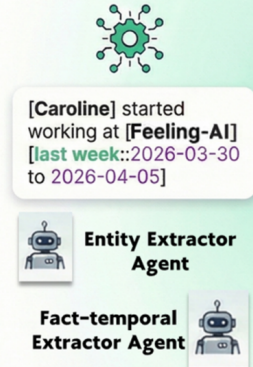
Step 1:

Raw Dialogue Input



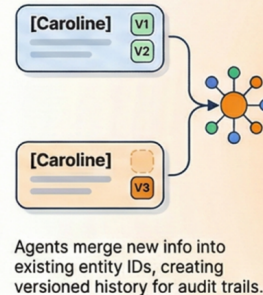
Step 2:

Entity & Fact Extraction

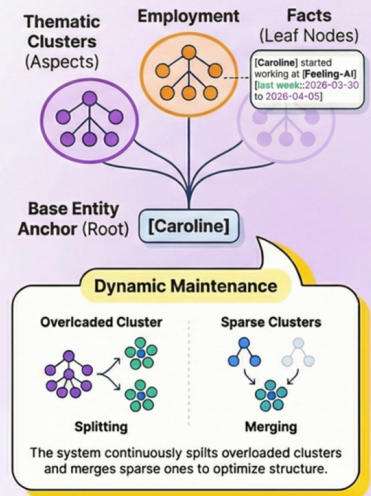


Step 3:

Entity Resolution & Fusion



Step 4: Hierarchical Tree Construction



NotebookLM

Batch processing. Each conversation (a session) is split into batches to stay within context limits. Batches are processed sequentially, with a small amount of context carried over so the system doesn't lose continuity across turns. After processing all batches in a session, the system generates a high-level summary of the conversation. For example: "Discussion about Caroline's new job, recent travel, and upcoming plans." This provides a quick entry point for queries that reference entire sessions rather than individual facts.

Entity and fact extraction. Extraction runs in two steps: first entities, then facts.

Entities are extracted in two passes. The first pass runs without context to get an initial entity name list. The system then retrieves related existing entities and feeds them back as context into a second pass, which refines the results and helps resolve ambiguity (e.g., linking a pronoun "she" to (Caroline)).

Facts are generated based on the finalized entity list. Each fact must explicitly reference entities using a bracketed format. For example:

- (Caroline) started a new job at (Feeling-AI)
- (Caroline) traveled to Shanghai (last week::2026-03-10 to 2026-03-16)

Here, entity mentions are explicitly marked, and time expressions are normalized into a consistent format ((raw::resolved)). This keeps the text readable while making it easier to index and retrieve later.

Entity resolution and updates. New entities are compared against existing ones to decide whether to merge or create. Simple cases—like exact name matches—are handled directly, while ambiguous cases (e.g., "Carrie" vs "Caroline") are resolved with the help of an agent.

Once resolved, entities are updated with a canonical name and a consolidated description. Instead of overwriting old data, the system creates a new version. For example, an entity might evolve from "Caroline, a student" to "Caroline, a software engineer at OpenAI", while still preserving earlier states.

Hierarchical organization. Facts are organized under each entity using a hierarchical structure. Initially, facts are attached directly to the entity. As more data accumulates, related facts are grouped into aspects. For example, facts about "Caroline" might be organized into clusters like:

- Career → job changes, promotions
- Travel → trips, locations

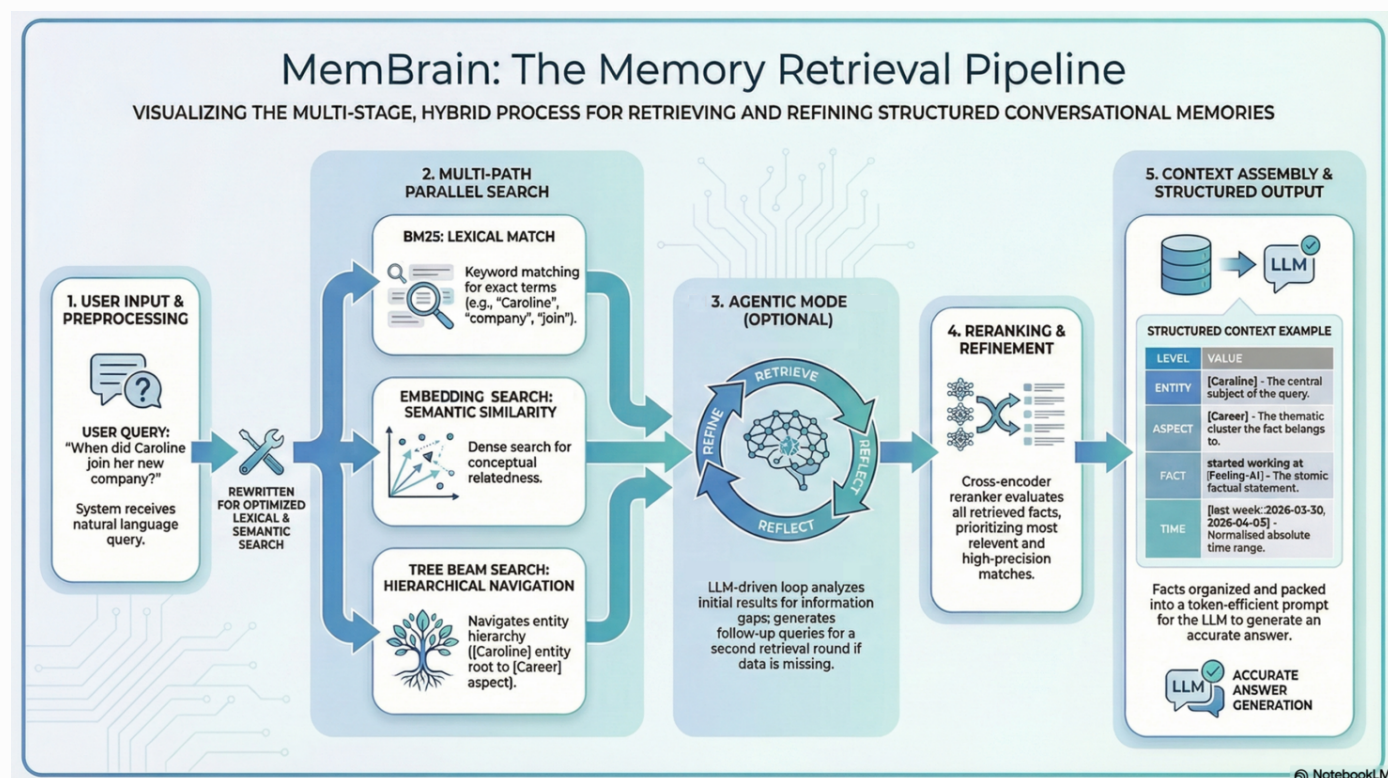
New facts are routed into the most relevant cluster based on semantic similarity. The structure is continuously refined—overloaded clusters can be split, and sparse ones can be merged. Summaries are generated for higher-level nodes to keep the hierarchy compact and easy to navigate.

Together, these steps convert raw dialogue into a structured, evolving memory. The system keeps the richness of natural language, while adding just enough structure to support efficient updates and retrieval.

RETRIEVAL WORKFLOW

The retrieval workflow builds a high-quality context for answering questions. It combines multiple retrieval signals and optionally adds a reflection agentic step for harder queries.

Standard retrieval.



Given a question, the system first rewrites it into a compact form for keyword search, and may generate a few variants to improve recall. For example, a query like "Where does Caroline work now?" might be expanded into keywords ("Caroline job") and a more descriptive version ("Caroline current company").

The system then runs three retrieval paths in parallel:

- Keyword search over fact text for exact matches
- Embedding search for semantic similarity
- Structure-aware search over entity trees, which first finds relevant entities (e.g., Caroline) and then retrieves facts from the most relevant aspects (e.g., Career)

Results from all paths are merged, deduplicated, and reranked to prioritize the most relevant facts while preserving precise matches.

Agent-augmented retrieval (optional). For more complex queries, the system adds a reflection step. After the first round of retrieval, an agent checks whether the results are sufficient. If not, it generates a small number of follow-up queries to fill in missing information and triggers a second retrieval round. This allows the system to progressively refine its understanding, rather than relying on a one-shot lookup.

For example, if the question is "What changed in Caroline's career recently?", the system might first retrieve job-related facts, then issue a follow-up query focused on recent time periods.

Context assembly. The final output is organized, not just retrieved. It combines three types of information:

- Session summaries for high-level context
- Facts grouped by entity and aspect (e.g., Caroline → Career → job changes)
- Optional raw messages as supporting evidence

Facts are selected by relevance while ensuring diversity across aspects, then ordered and structured into a compact prompt.

This pipeline ensures that retrieved context is both precise and well-organized, making it easier for the model to generate grounded and coherent answers.

EXPERIMENT RESULTS

We compare MemBrain against state-of-the-art memory systems: MemoryOS, Mem0, MemU, MemOS, Zep, and EverMemOS. We evaluate their performances across 4 benchmarks: LoCoMo, LongMemEval, PersonaMem-V2, and KnowMe-Bench. Baselines are primarily taken from the official EverMemOS results. Based on its evaluation protocol and codebase, we implement a more stable and feature-complete benchmarking toolkit, together with a front-end viewer for inspecting datasets and memory structures.

Table 1: Experiment results on **LoCoMo** under GPT-4.1-mini backbone. Baselines are primarily from the official *EverMemOS* results. All metrics are accuracy (%), except Avg. Tokens.

Method	Avg. Tokens	Single Hop	Multi Hop	Temporal	Open Domain	Overall
<i>MemoryOS</i>	5.5k	67.30	59.34	42.26	59.03	60.11
<i>Mem0</i>	1.0k	68.97	61.70	58.26	50.00	64.20
<i>MemU</i>	4.0k	74.91	72.34	43.61	54.17	66.67
<i>MemOS</i>	2.5k	85.37	79.43	75.08	64.58	80.76
<i>Zep</i>	1.4k	90.84	81.91	77.26	75.00	85.22
<i>EverMemOS</i>	2.3k	96.67	91.84	89.72	76.04	93.05
<i>MemBrain</i>	2.3k	96.67	91.49	90.97	76.04	93.25

Table 2: Experiment results on **LongMemEval** under GPT-4.1-mini backbone. Baselines are primarily from the official *EverMemOS* results. All metrics are accuracy (%), except Avg. Tokens.

Method	Token	SS-User	SS-Asst	SS-Pref	Multi-S	Know. Upd	Temp. Reas	Overall
<i>MemU</i>	0.5k	67.14	19.64	76.67	42.10	41.02	17.29	38.40
<i>Zep</i>	1.6k	92.90	75.00	53.30	47.40	74.40	54.10	63.80
<i>Mem0</i>	1.1k	82.86	26.78	90.00	63.15	66.67	72.18	66.40
<i>MemOS</i>	1.4k	95.71	67.86	96.67	70.67	74.26	77.44	77.80
<i>EverMemOS</i>	2.8k	97.14	85.71	93.33	73.68	89.74	77.44	83.00
<i>MemBrain</i>	2.3k	98.57	98.21	93.33	76.69	88.46	78.95	85.60

Table 3: Experiment results on **PersonaMem-V2** under GPT-4.1-mini backbone. Baselines are primarily from the official *EverMemOS* results. All metrics are accuracy (%).

Scenario	<i>Zep</i>	<i>Mem0</i>	<i>MemU</i>	<i>MemoryOS</i>	<i>MemOS</i>	<i>EverMemOS</i>	<i>MemBrain</i>
Consultation	39.51	43.21	37.86	35.80	48.15	51.03	54.29
Email (Personal)	42.51	41.30	33.20	36.84	49.80	53.85	55.50
Translation	36.92	43.08	38.46	40.00	51.92	50.00	55.87
Email (Professional)	37.59	42.41	32.76	35.86	50.00	53.79	55.61
Creative Writing	41.22	42.86	35.51	35.51	48.16	55.10	52.13
Writing (Professional)	40.54	34.75	35.14	35.91	48.26	45.56	50.09
Knowledge Query	63.43	59.20	56.97	57.96	61.94	63.68	67.35
Social Media	32.35	38.66	34.03	35.29	46.64	47.90	49.08
Chat	44.87	40.30	34.22	36.88	44.87	52.09	55.42
Profile	X	X	✓	✓	✓	✓	✓
Overall	43.40	43.85	38.70	40.05	50.72	53.25	55.72

Table 4: Experiment results on **KnowMe-Bench**. Baselines are from the official KnowMe-Bench results on GPT-5-mini backbone. All metrics are rubric-based LLM scores (%) normalized from a 0-5 scale, validated by human alignment ($\kappa > 0.75$). Our method outperforms the baseline methods by a substantial margin in two Level III (Psychoanalytic Depth) tasks: Mind-Body Interaction (T6) and Expert-Annotated Psychoanalysis (T7). These results demonstrate that our memory system moves beyond simple data retrieval to master high-order cognitive modeling.

Method	<i>Base Model</i>	<i>Naive RAG</i>	<i>Mem0 (Entity)</i>	<i>MemOS</i>	<i>MemBrain</i>
Mind-Body Interaction	18.6	21.3	20.7	21.5	82.6
Expert-Annotated Psychoanalysis	19.6	20.9	20.8	22.6	64.9

FUTURE WORK

A database should not be treated as just a container for retrievable objects. In a memory system, it is core infrastructure. In our experience, the more the system leverages database-level properties—traceability, auditability, isolation, and synchronization—the easier it becomes to iterate. Every operation is recorded, reversible, and inspectable, which significantly lowers the cost of experimentation. With a solid foundation, each decision made by agents during memory management naturally accumulates as data, feeding back into the system and enabling continuous improvement.

Another direction worth exploring is the signal embedded in the retrieval process itself. Most evaluations focus on whether the final answer is correct, but in practice, retrieval paths also reveal the mismatch between how memory is organized and how queries are distributed. This suggests a natural extension: use retrieval as an opportunity to lightly reorganize memory along the accessed paths, gradually aligning the structure with real usage patterns. In this view, each query is not just reading memory, but also helping maintain it.

The challenge is largely on the engineering side. Incremental updates need to remain stable, and the cost of reorganization must be amortized over time—handled asynchronously without slowing down retrieval.