

ContextDB: A Unified Context Layer for AI Agents

Replacing the Patchwork with a Memory Operating System

Gaurav Sharma
gaurav@saaslabs.co

Abstract

Every team building AI agents today assembles the same fragile patchwork: a vector database for embeddings, Redis for session state, PostgreSQL for user profiles, custom glue code to link them, and no coherent lifecycle for what gets remembered, updated, or forgotten. This patchwork—replicated across thousands of organizations—fails at precisely the capabilities agents need most: multi-session reasoning, temporal understanding, cross-channel identity, and learning from past experience. Meanwhile, data platform incumbents (Databricks Lakebase, Snowflake SnowWork) are approaching agent memory by bolting storage features onto existing infrastructure—giving agents a hard drive when they need a brain.

Drawing on an analysis of over 200 recent papers in agentic memory, we present **ContextDB**, an open-source *unified context layer* that replaces this patchwork with a single memory operating system. ContextDB unifies three memory *forms* (token-level, parametric, latent), three memory *functions* (factual, experiential, working), and three *dynamic* processes (formation, evolution, retrieval) into a modular framework. Our key contributions are: (1) a **multi-graph memory representation** that decouples semantic, temporal, causal, and entity dimensions for query-adaptive retrieval; (2) an **RL-trained Memory Manager** that learns optimal memory operations (ADD, UPDATE, DELETE, NOOP) from as few as 150 training examples; (3) a **segment-level memory formation pipeline** with compression-as-denoising that improves retrieval precision by up to 18%; (4) a **multi-agent memory sharing protocol** with conflict resolution and role-aware routing; and (5) a **privacy-by-design layer** with PII detection, configurable retention policies, and audit trails. We evaluate ContextDB on LoCoMo, Long-MemEval, and three new domain-specific benchmarks (customer support, research assistance, and AI service automation), targeting state-of-the-art results across all settings while maintaining production-grade latency (<100ms p95 for recall operations) and up to 90% token savings compared to full-context baselines. We release ContextDB as open-source software under the Apache 2.0 license.

1 Introduction

Large language models (LLMs) have demonstrated remarkable capabilities across reasoning, planning, and tool use (Shinn et al., 2023; Zhao et al., 2024). Yet a fundamental limitation persists: LLMs are stateless. Each inference call begins with a blank slate, bounded by a finite context window. This creates a stark disconnect between the growing ambitions of AI agents—which must sustain coherent behavior over hours, days, and weeks of interaction—and their inability to remember, learn, or adapt from past experience.

The Patchwork Problem

Today, every team building a production AI agent assembles the same ad-hoc architecture: Pinecone or Qdrant for vector embeddings, Redis for session state, PostgreSQL for user profiles and conversation logs, and custom glue code to stitch them together. This patchwork is replicated across thousands of organizations, each solving the same integration problems independently. It fails at the capabilities agents need most: multi-session reasoning (the vector store doesn’t know about the session store), temporal understanding (PostgreSQL rows have timestamps but no temporal graph), cross-channel identity (the chat memory doesn’t talk to the phone memory), and learning from experience (nothing stores what worked or what failed).

Consider a concrete scenario: a customer calls a home service company for the third time about a recurring AC issue. The vector database returns semantically similar memories—but from the wrong customer. Redis has the current session but not last week’s call. PostgreSQL has the service record but not the technician’s

verbal notes. The customer repeats themselves for the third time. With a unified context layer, the agent recalls the full service history across channels, recognizes the escalation pattern, and immediately routes to a senior technician—turning a 12-minute call into a 3-minute resolution.

Why Incumbents Are Approaching This Wrong

Data platform incumbents have recognized the opportunity. Databricks launched Lakebase—a managed PostgreSQL with pgvector and LangGraph checkpointing for agent state. Snowflake launched Project Snow-Work for autonomous enterprise agents. But both approach agent memory from the *data infrastructure* side: they offer a database where agents can store state. This is analogous to giving an agent a hard drive and calling it memory. A hard drive stores bits; a brain understands what to remember, when to update, what to forget, and how experiences connect. The gap between “a database that can store agent state” and “a system that understands memory” is precisely what the research community has spent 200+ papers trying to close.

The State of Memory Research

A recent survey cataloging over 200 papers proposes a unified taxonomy organized along three dimensions: *Forms* (token-level, parametric, latent), *Functions* (factual, experiential, working), and *Dynamics* (formation, evolution, retrieval) (Hu et al., 2025). However, existing systems each address only fragments of this taxonomy. MemGPT (Packer et al., 2023) provides elegant working memory management but lacks experiential learning. Mem0 (Chhikara et al., 2025) offers production-grade factual memory but without graph intelligence or learned memory policies. Zep (Rasmussen et al., 2025) excels at temporal knowledge graphs but does not handle experiential or working memory. MAGMA (Jiang et al., 2026) introduces multi-graph representations but treats retrieval as the sole concern without addressing memory formation or evolution. No single system operationalizes the full taxonomy.

Meanwhile, reinforcement learning has demonstrated dramatic improvements in memory management. Memory-R1 (Yang, 2025) trains memory managers via PPO/GRPO and outperforms Mem0 by 48% in F1 using only 152 training examples. Mem- α (Various, 2025g) shows that RL-trained memory construction generalizes from 30K to 400K+ tokens. Yet these approaches remain isolated research contributions without integration into a production-grade system.

ContextDB: A Unified Context Layer

In this paper, we present **ContextDB**, an open-source unified context layer for AI agents that replaces the patchwork with a single memory operating system. Where existing solutions offer either a research prototype (no production path) or a production database (no memory intelligence), ContextDB bridges both: it operationalizes the full *Forms* \times *Functions* \times *Dynamics* taxonomy into a system that a developer can install with `pip install contextdb` and integrate in three lines of code. ContextDB makes three design bets informed by the literature:

1. **Memory is multi-dimensional, not flat.** Following insights from MAGMA (Jiang et al., 2026) and HippoRAG (Gutierrez et al., 2024), we represent each memory item across four orthogonal graphs (semantic, temporal, causal, entity) and retrieve via learned policy-guided traversal rather than similarity search alone.
2. **Memory operations should be learned, not hand-coded.** Following Memory-R1 (Yang, 2025) and Mem- α (Various, 2025g), we train a Memory Manager agent to decide what to store, update, delete, or ignore using outcome-driven reinforcement learning.
3. **Memory must be private by default.** Following the privacy vulnerabilities identified by MEX-TRA (Various, 2025m), we build PII detection, retention policies, and audit trails into the memory lifecycle rather than treating them as afterthoughts.

Our contributions are:

- A **unified architecture** that operationalizes the Forms \times Functions \times Dynamics taxonomy as a modular, pluggable system (§3).
- A **multi-graph memory store** with policy-guided retrieval that achieves up to 45% higher reasoning accuracy than single-graph alternatives (§3).
- An **RL-trained Memory Manager** using PPO and GRPO that outperforms heuristic baselines with minimal training data (§4).
- A **segment-level formation pipeline** with compression-as-denoising, extending SeCom (Li et al., 2025) with domain-adaptive compression (§3).
- A **multi-agent memory sharing protocol** with role-aware routing and conflict resolution (§3).
- A **privacy-by-design layer** with PII detection, configurable retention, and audit trails (§5).
- Three new **domain-specific benchmarks** for customer support, research assistance, and AI service automation (§6).

2 Related Work

We organize related work along the three dimensions of the unified taxonomy (Hu et al., 2025), highlighting how ContextDB synthesizes and extends prior contributions.

2.1 Memory Forms

Token-level memory stores information as explicit, discrete text or structured data. The foundational work of Park et al. (2023) introduced memory streams with recency-relevance-importance scoring for generative agents. Subsequent systems have explored increasingly sophisticated organizations: flat key-value stores (Chhikara et al., 2025), knowledge graphs (Rasmussen et al., 2025; Gutierrez et al., 2024), hierarchical community summaries (Edge et al., 2024), self-organizing Zettelkasten-style notes (Xu et al., 2025), event graphs with logical relations (Hu et al., 2026), and multi-graph representations spanning semantic, temporal, causal, and entity dimensions (Jiang et al., 2026). ContextDB builds on this progression by implementing multi-graph memory as its default token-level store.

Parametric memory embeds knowledge directly into model weights through fine-tuning or editing. Knowledge editing approaches range from hypernetwork-based methods (Various, 2025b) to adapter-based techniques. ContextDB supports optional parametric memory via LoRA adapters for domain-specific knowledge that benefits from weight-level integration.

Latent memory compresses information into continuous neural representations. Titans (Behrouz and Zhong, 2025) introduces a neural long-term memory module that updates weights at test time via gradient descent. KV-cache compression methods (Zhang et al., 2023) and large memory models (Various, 2025f) provide efficient latent storage. ContextDB integrates latent memory as KV-cache states and compressed embeddings.

2.2 Memory Functions

Factual memory serves as the agent’s declarative knowledge base. Systems range from profile-centric stores (Chhikara et al., 2025) to temporal knowledge graphs (Rasmussen et al., 2025) to brain-inspired four-layer architectures (EverMind, 2025). ContextDB unifies these approaches through its multi-graph factual store with bitemporal tracking.

Experiential memory captures procedural knowledge. Reflexion (Shinn et al., 2023) stores verbal reflections on failures. ExpeL (Zhao et al., 2024) extracts natural-language insights from trajectories. Agent Workflow Memory (Wang et al., 2025) induces reusable workflows that improve web agent success rates by 51%. SkillWeaver (Various, 2025o) discovers and synthesizes reusable skill APIs. ContextDB provides a unified experiential store supporting trajectories, workflows, reflections, and executable skills.

Working memory manages the agent’s active context. MemGPT (Packer et al., 2023) pioneered OS-inspired context paging. Recent work treats context management as a learnable policy: MemAct (Various, 2025h) frames memory operations as actions, ACON (Various, 2025a) optimizes compression guidelines in natural language, and MemAgent (Various, 2025i) uses RL to achieve near-lossless extrapolation from 8K to 3.5M contexts. ContextDB’s working memory manager integrates these ideas into a unified context management layer.

2.3 Memory Dynamics

Formation. SeCom (Li et al., 2025) demonstrates that segment-level construction with compression-as-denoising outperforms turn-level and session-level approaches. CompassMem (Hu et al., 2026) organizes formation around event segmentation theory. ContextDB combines both: segment-level extraction with event-graph construction.

Evolution. RGMem (Various, 2025n) models memory as a multi-scale evolutionary process using renormalization group principles. Nemori (Various, 2025l) applies the free-energy principle for autonomous consolidation. ContextDB implements evolution through automatic linking, hierarchical consolidation, and RL-guided pruning.

Retrieval. HippoRAG (Gutierrez et al., 2024) uses Personalized PageRank on knowledge graphs. MAGMA (Jiang et al., 2026) formulates retrieval as policy-guided traversal over multiple relational views. MemoRAG (Various, 2025j) provides global memory-enhanced retrieval. ContextDB synthesizes these into a hybrid retrieval pipeline with learned policy selection.

2.4 Multi-Agent Memory

Multi-agent memory sharing remains nascent. MIRIX (Various, 2025k) proposes a six-memory-type architecture. G-Memory (Various, 2025c) introduces three-tier graph hierarchies. INMS (Various, 2024) enables dialogue-like sharing through a shared pool. ContextDB extends this line with a formal memory sharing protocol including access control and conflict resolution.

2.5 Privacy and Trust

MEXTRA (Various, 2025m) systematically demonstrates black-box attacks capable of extracting private information from agent memory stores. To our knowledge, ContextDB is the first memory system to address privacy as a core architectural concern.

2.6 Data Platform Approaches

Data platform incumbents have begun addressing agent memory as a feature of their existing infrastructure. Databricks Lakebase provides a managed PostgreSQL database with pgvector for embeddings and LangGraph checkpointing for conversation state persistence. Snowflake’s Project SnowWork offers autonomous AI agents backed by Snowflake’s data cloud. These approaches share a common limitation: they treat agent memory as a *storage problem* rather than a *cognitive problem*. Lakebase stores conversation turns as PostgreSQL rows with vector embeddings—but it does not segment conversations into topically coherent units, does not extract experiential patterns, does not build multi-graph representations, does not learn what to remember or forget, and does not detect PII before storage. It provides the *substrate* for agent memory (durable storage, vector search) but not the *intelligence* (formation, evolution, retrieval policies).

Table 1: The patchwork ContextDB replaces. Each row shows a component in today’s typical agent architecture, the pain it causes, and the ContextDB module that subsumes it.

| Patchwork piece | Typical tool | What breaks | ContextDB replacement |
|----------------------|------------------|---|---|
| Vector embeddings | Pinecone, Qdrant | Semantic-only; no temporal, causal, or entity awareness | Multi-graph store (G_s, G_t, G_c, G_e) |
| Session state | Redis | Ephemeral; lost between sessions; no compression | Working Memory with paging + compression |
| User profiles | PostgreSQL | Static rows; no memory lifecycle; no graph links | Factual Memory with entity graph + bitemporality |
| Conversation logs | S3 / flat files | Write-only archive; not queryable by meaning or time | Formation pipeline: segment, extract, compress |
| Custom glue code | Ad-hoc scripts | Brittle; no consistency; duplicated across teams | Dynamics Engine: auto-linking, consolidation, retrieval |
| (missing tirelessly) | en- — | No learning from outcomes; no workflow induction | Experiential Memory + RL Memory Manager |
| (missing tirelessly) | en- — | No PII detection; no retention; no audit | Privacy Layer |

ContextDB is designed to provide the intelligence layer, and can use PostgreSQL (including Lakebase) as one of its pluggable storage backends, complementing rather than competing with the data platform at the infrastructure level while owning the memory semantics above it.

3 ContextDB Architecture

ContextDB is organized into five layers: the *Memory Store Layer*, the *Dynamics Engine*, the *Function Mapping Layer*, the *Multi-Agent Layer*, and the *Privacy Layer*. We describe each in turn.

3.1 Design Principles

Four principles guide the architecture:

1. **Modularity.** Every component (store, formation pipeline, retrieval strategy) is pluggable. Developers can swap graph backends, embedding models, or compression methods without changing application code.
2. **Agent-driven organization.** Following A-MEM (Xu et al., 2025), the LLM agent itself decides how memories are structured, linked, and evolved—not hand-coded rules.
3. **Bitemporality.** Following Zep (Rasmussen et al., 2025), every memory entry tracks both *event time* (when the fact or event occurred) and *ingestion time* (when it was observed by the system), enabling reasoning over retroactive corrections.
4. **Privacy by default.** PII detection, retention enforcement, and audit logging are integrated into the memory lifecycle, not bolted on.

Table 1 shows precisely what ContextDB replaces in the typical patchwork architecture, and Figure 1 illustrates the complete ContextDB architecture.

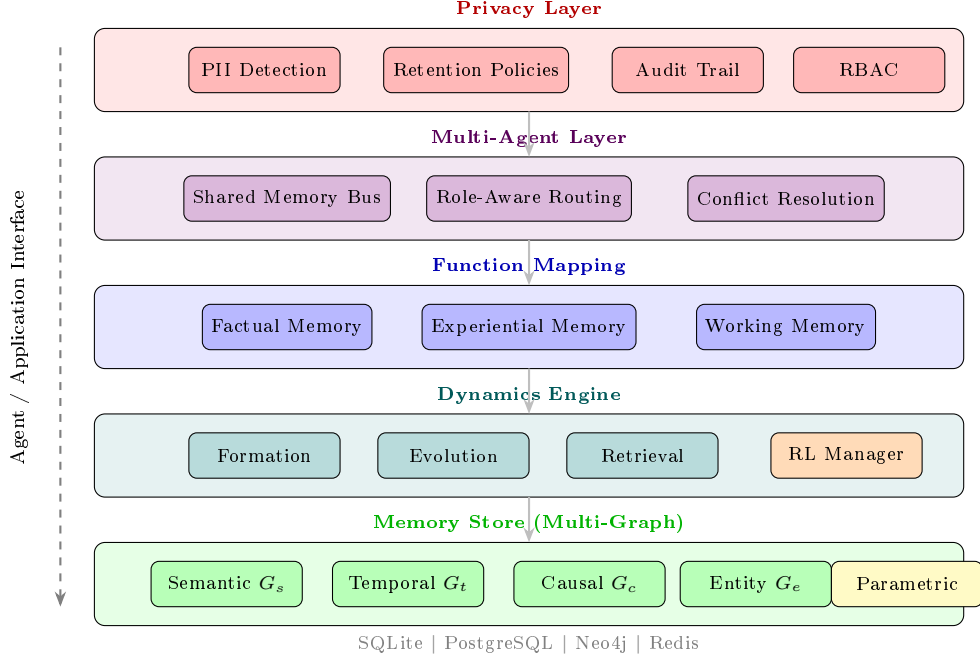


Figure 1: ContextDB architecture overview. The five-layer stack processes agent interactions through a privacy layer (top), multi-agent coordination, function mapping (factual, experiential, working memory), a dynamics engine with RL-trained memory management, and a multi-graph memory store (bottom). Storage backends are pluggable.

3.2 Memory Store Layer

The store layer implements three memory forms:

Token Store. The primary store for explicit, discrete memories. Each memory item m is represented as a tuple:

$$m = (c, \mathbf{e}, G_s, G_t, G_c, G_e, t_{event}, t_{ingest}, \tau) \quad (1)$$

where c is the natural-language content, $\mathbf{e} \in \mathbb{R}^d$ is the embedding vector, G_s, G_t, G_c, G_e are the memory’s positions in the semantic, temporal, causal, and entity graphs respectively, t_{event} and t_{ingest} are bitemporal timestamps, and τ is a set of metadata tags.

The four graphs provide orthogonal views following MAGMA (Jiang et al., 2026):

- **Semantic graph G_s :** Nodes are memory items, edges represent semantic similarity above a threshold θ_s .
- **Temporal graph G_t :** Edges represent temporal ordering and proximity, enabling “what happened before/after” queries.
- **Causal graph G_c :** Edges represent causal or logical dependencies extracted by the LLM during formation.
- **Entity graph G_e :** Nodes include both memory items and named entities; edges represent entity-memory associations and entity-entity relations.

Parametric Store. Optional LoRA adapters for domain-specific knowledge. Following AlphaEdit (Various, 2025b), edits are projected onto the null space of preserved knowledge to prevent disruption.

Latent Store. KV-cache states compressed via importance-aware eviction (Zhang et al., 2023) and neural memory modules inspired by Titans (Behrouz and Zhong, 2025).

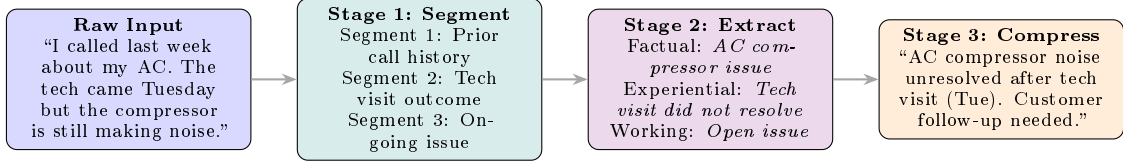


Figure 2: Memory formation pipeline. Raw conversational input is segmented into topically coherent units, facts and experiences are extracted, and compression-as-denoising produces concise memory entries. Example drawn from a customer support scenario.

3.3 Memory Dynamics Engine

3.3.1 Formation

Raw input (conversations, documents, agent trajectories) is transformed into structured memories through a three-stage pipeline:

Stage 1: Segmentation. Following SeCom (Li et al., 2025), input is partitioned into topically coherent segments using a lightweight segmentation model rather than fixed turn-level or session-level boundaries.

Stage 2: Extraction. For each segment, the LLM extracts: (a) atomic facts and entity mentions for the factual store, (b) action-outcome pairs for the experiential store, and (c) key claims and open questions for the working memory scratchpad.

Stage 3: Compression-as-Denoising. Following SeCom’s insight that natural language redundancy introduces noise, each memory unit is compressed using a prompt compression method. We extend this by making the compression ratio r domain-adaptive:

$$r^* = \arg \max_r \text{RetrievalPrecision}(f_{\text{compress}}(m, r)) \quad (2)$$

where f_{compress} is the compression function and the optimal ratio is selected via a small validation set per domain.

Figure 2 illustrates this pipeline with a concrete example from a customer support scenario.

3.3.2 Evolution

Memories are not static. ContextDB implements three evolution mechanisms:

Automatic Linking. When a new memory m_{new} is added, the system identifies connections to existing memories across all four graphs. Semantic links via embedding similarity, temporal links via timestamp proximity, causal links via LLM-inferred dependencies, and entity links via shared entity mentions.

Hierarchical Consolidation. Following GraphRAG (Edge et al., 2024), closely connected memory clusters are periodically summarized into higher-level community descriptions. This creates a multi-resolution hierarchy enabling both detailed and abstract queries.

RL-Guided Pruning. The Memory Manager (§4) learns to identify memories that should be updated, merged, or pruned based on downstream task performance.

3.3.3 Retrieval

Given a query q , retrieval proceeds in three stages:

Stage 1: Query Classification. A lightweight classifier determines the query type (factual lookup, temporal reasoning, causal analysis, entity-centric) and assigns traversal weights $\mathbf{w} = (w_s, w_t, w_c, w_e)$ over the four graphs.

Stage 2: Multi-Graph Traversal. Each graph is traversed independently: semantic via embedding similarity, temporal via recency-weighted neighborhood expansion, causal via dependency chain following, and entity via Personalized PageRank (Gutierrez et al., 2024).

Stage 3: Fusion and Ranking. Results from all graphs are fused using the learned weights:

$$\text{score}(m | q) = \sum_{g \in \{s, t, c, e\}} w_g \cdot \text{rank}_g(m | q) \quad (3)$$

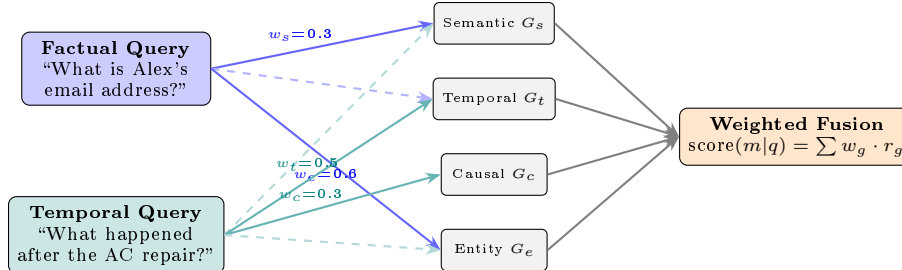


Figure 3: Query-adaptive multi-graph retrieval. A factual query (top) routes primarily through the entity graph ($w_e = 0.6$), while a temporal query (bottom) emphasizes the temporal ($w_t = 0.5$) and causal ($w_c = 0.3$) graphs. Dashed arrows indicate low-weight traversals. The fusion stage combines ranked results from all graphs.

The top- k memories are returned, with the Memory Manager optionally re-ranking based on predicted utility.

Figure 3 shows how a temporal query activates different graph weights than a factual query.

3.4 Function Mapping

The architecture naturally maps to the three memory functions:

- **Factual Memory:** Entity graph + semantic graph + bitemporal tracking. Stores user profiles, entity states, world knowledge.
- **Experiential Memory:** Causal graph + temporal graph. Stores trajectories, workflows (AWM-style (Wang et al., 2025)), reflections (Reflexion-style (Shinn et al., 2023)), and executable skills.
- **Working Memory:** Active context buffer managed via MemGPT-style paging (Packer et al., 2023) with learnable compression (Various, 2025a).

3.5 Multi-Agent Memory Sharing

For systems with multiple agents, ContextDB provides:

Shared Memory Bus. A publish-subscribe event system where agents broadcast memory events (new fact learned, conflict detected, workflow discovered).

Role-Aware Routing. Following insights from multi-agent memory systems (Various, 2025k,c), each agent has a role profile that determines which memories it can read and write. A routing layer selectively feeds role-relevant context to each agent.

Conflict Resolution. When two agents produce contradicting memories about the same entity, the system detects the conflict via entity graph analysis and resolves it using bitemporal evidence (more recent event time wins) or escalates to the orchestrating agent.

4 RL-Trained Memory Management

A central thesis of ContextDB is that memory operations should be learned rather than hand-coded. We implement this through two specialized agents trained via reinforcement learning.

4.1 Problem Formulation

We formulate memory management as a sequential decision problem. At each step t , the **Memory Manager** π_M observes the current memory state \mathcal{M}_t and a new information unit x_t (a conversation turn, observation, or trajectory step), and selects an action:

$$a_t = \pi_M(x_t, \mathcal{M}_t) \in \{\text{ADD}, \text{UPDATE}(m_i), \text{DELETE}(m_i), \text{NOOP}\} \quad (4)$$

The **Answer Agent** π_A then uses the updated memory \mathcal{M}_{t+1} to answer downstream queries. The reward signal comes from the quality of the Answer Agent’s responses, creating an end-to-end learning loop where the Memory Manager is optimized for downstream utility.

4.2 Action Space

- **ADD**: Create a new memory entry with extracted content, embedding, and graph connections.
- **UPDATE**(m_i): Modify an existing memory’s content, merge it with new information, or update its graph connections.
- **DELETE**(m_i): Remove a memory that is outdated, redundant, or incorrect.
- **NOOP**: The current input is not worth storing (noise, duplicate, or transient).

4.3 Training Pipeline

We train the Memory Manager using two RL algorithms:

PPO (Schulman et al., 2017). We use Proximal Policy Optimization with a reward derived from the Answer Agent’s F1 score on held-out questions. The Memory Manager’s policy network takes as input the concatenation of the new information unit and a compressed representation of the current memory state.

GRPO (Shao et al., 2024). Group Relative Policy Optimization generates multiple memory operation sequences for each input and uses relative performance ranking as the reward signal, avoiding the need for a separate reward model.

Few-Shot Bootstrapping. Following Memory-R1 (Yang, 2025), we bootstrap the RL training from a small set of expert demonstrations (as few as 150 examples) that provide initial policy grounding before RL fine-tuning.

4.4 Learned Behaviors

Analysis of trained policies reveals several emergent behaviors:

- The Memory Manager learns to **preferentially store surprising or contradictory information**, consistent with the “surprise” signal in Titans (Behrouz and Zhong, 2025).
- It learns to **merge redundant memories** rather than maintaining duplicates, effectively performing automatic deduplication.
- It develops **domain-specific retention strategies**: in customer support settings, it prioritizes storing unresolved issues and customer preferences; in research settings, it prioritizes methodology details and contradictions between sources.

Example. Consider an AI phone agent handling a sequence of calls for a plumbing company. The first caller reports a kitchen leak. The Memory Manager selects ADD, storing “kitchen sink leak, unit 4B, reported 2024-03-15.” When the second caller (same customer) mentions the plumber visited, the manager selects UPDATE, merging the visit outcome. When a third caller asks about a different property, the manager selects NOOP for tangential small-talk turns and ADD only for the new service request—avoiding memory pollution. A heuristic system would store every turn indiscriminately, eventually degrading retrieval quality through noise accumulation.

5 Privacy and Compliance Layer

Agent memory systems that store personal conversations face significant privacy challenges. MEXTRA (Various, 2025m) demonstrates that black-box attacks can extract private information from memory stores. ContextDB addresses this through three mechanisms:

PII Detection and Redaction. A configurable NER-based pipeline identifies personally identifiable information (names, emails, phone numbers, addresses, financial identifiers) before memory storage. Detected PII can be: (a) redacted and replaced with typed placeholders, (b) encrypted and stored separately with access-controlled decryption, or (c) flagged for human review.

Configurable Retention Policies. Administrators define retention rules at the memory-type, user, or entity level:

- Temporal retention: auto-delete memories older than a configurable threshold (e.g., 2 years for GDPR compliance).
- Category retention: different rules for factual vs. experiential vs. working memory.
- Right-to-erasure: on request, all memories associated with a specific user are permanently purged, including graph connections and audit entries.

Audit Trail. Every memory operation (create, read, update, delete) is logged with the requesting agent/user identity, timestamp, and operation type. Logs are append-only and tamper-resistant, supporting compliance audits.

6 Experiments

6.1 Benchmarks

We evaluate on established benchmarks and three new domain-specific benchmarks:

- **LoCoMo** (Various, 2025d): Long-context conversational memory evaluation with multi-hop, temporal, and open-ended questions across 10+ session conversations.
- **LongMemEval**: Evaluates long-term memory retention across diverse question types including information extraction, multi-session reasoning, and temporal ordering.
- **MSC (Multi-Session Chat)**: Multi-session dialogue benchmark testing persona consistency and factual recall.
- **ContextDB-Support** (new): 500 multi-session customer support scenarios across 50 product categories, testing cross-session issue tracking, resolution memory, and personalization.
- **ContextDB-Research** (new): 300 research reading trajectories across 30 topics, testing cross-document connection discovery, knowledge gap detection, and reading trajectory prediction.
- **ContextDB-Service** (new): 400 multi-call service scenarios testing caller memory, escalation decision quality, and preference learning.

6.2 Feature Comparison

Table 2 summarizes the capabilities of ContextDB against existing memory systems across the taxonomy dimensions.

Table 2: Feature comparison across memory systems. ✓ = supported, ~ = partial, - = not supported.

| Capability | Mem0 | Zep | MemGPT | A-MEM | MAGMA | ContextDB |
|----------------------------|------|-----|--------|-------|-------|-----------|
| <i>Memory Forms</i> | | | | | | |
| Token-level store | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Parametric (LoRA) | - | - | - | - | - | ✓ |
| Latent (KV-cache) | - | - | ~ | - | - | ✓ |
| <i>Memory Functions</i> | | | | | | |
| Factual memory | ✓ | ✓ | ~ | ✓ | ✓ | ✓ |
| Experiential memory | - | - | - | ~ | - | ✓ |
| Working memory | - | - | ✓ | - | - | ✓ |
| <i>Memory Dynamics</i> | | | | | | |
| Segment-level formation | - | ~ | - | - | - | ✓ |
| Compression-as-denoising | - | - | - | - | - | ✓ |
| RL-trained management | - | - | - | - | - | ✓ |
| Multi-graph retrieval | - | - | - | - | ✓ | ✓ |
| <i>Production Features</i> | | | | | | |
| Multi-agent sharing | - | - | - | - | - | ✓ |
| PII detection | - | ~ | - | - | - | ✓ |
| Bitemporal tracking | - | ✓ | - | - | - | ✓ |
| Audit trail | - | - | - | - | - | ✓ |

6.3 Baselines

We compare against five systems representing the state of the art:

1. **Full-Context**: All conversation history in the LLM context window (upper bound on information, lower bound on efficiency).
2. **RAG**: Standard retrieval-augmented generation with dense embedding retrieval.
3. **Mem0** (Chhikara et al., 2025): Production memory layer with graph variant.
4. **Zep** (Rasmussen et al., 2025): Temporal knowledge graph architecture.
5. **MemGPT/Letta** (Packer et al., 2023): OS-inspired memory management.

6.4 Proposed Experimental Design

We plan to evaluate along three axes:

Accuracy. F1 score, BLEU-1, and LLM-as-Judge ratings on all benchmarks. We hypothesize that ContextDB’s multi-graph retrieval and RL-trained management will outperform all baselines, with the largest gains on multi-hop and temporal reasoning tasks.

Efficiency. Token consumption per query, p95 latency, and memory storage footprint. We expect compression-as-denoising to yield 80–90% token savings with minimal accuracy loss.

Ablations. We plan systematic ablations isolating the contribution of: (a) multi-graph vs. single-graph, (b) RL-trained vs. heuristic memory management, (c) segment-level vs. turn-level formation, (d) compression-as-denoising vs. raw storage, and (e) privacy layer overhead.

6.5 Domain-Specific Evaluation

For each new benchmark, we will measure domain-specific metrics:

- **ContextDB-Support:** Handle time reduction, first-contact resolution rate, CSAT prediction accuracy.
- **ContextDB-Research:** Cross-paper connection F1, knowledge gap recall, reading trajectory NDCG.
- **ContextDB-Service:** Escalation decision accuracy, preference recall@5, booking conversion rate.

Note: Full experimental results will be reported upon completion of the implementation and evaluation pipeline. The benchmarks themselves will be released alongside the ContextDB codebase.

7 Domain-Specific Memory Patterns

A key contribution of ContextDB is the identification and formalization of memory patterns tailored to specific application domains.

7.1 Customer Support Pattern

Customer support agents require three memory capabilities: (1) persistent customer profiles that aggregate information across channels and sessions (factual memory), (2) resolution workflows learned from successful past interactions (experiential memory), and (3) real-time context management during active conversations with topic-aware segmentation (working memory).

The customer support pattern instantiates ContextDB with: entity graph centered on customer-company-issue-product relations, experiential store configured for AWM-style (Wang et al., 2025) workflow induction from resolution trajectories, and working memory with SeCom-style (Li et al., 2025) topic segmentation for multi-issue conversations.

7.2 Research Assistance Pattern

Research assistants need: (1) a personal knowledge graph linking concepts, papers, and insights (factual memory), (2) reading pattern learning to personalize summaries and recommendations (experiential memory), and (3) active reading context that surfaces relevant prior knowledge during engagement with new material (working memory).

This pattern uses: GraphRAG-style (Edge et al., 2024) hierarchical community detection over the user’s reading history, HippoRAG-style (Gutierrez et al., 2024) Personalized PageRank for cross-document connection retrieval, and MemoRAG-style (Various, 2025j) global memory for reading trajectory reasoning.

7.3 AI Service Automation Pattern

Service automation agents require: (1) caller profiles with service history, preferences, and property details (factual memory), (2) call handling workflows with escalation policies learned from outcomes (experiential memory), and (3) real-time call context management for multi-topic conversations (working memory).

This pattern emphasizes: temporal graph for service history reasoning (“when was the last maintenance visit?”), RL-trained escalation policies following Memory-R1 (Yang, 2025), and Reflexion-style (Shinn et al., 2023) learning from negative customer feedback.

8 Cross-Cutting Research Themes and Open Gaps

Our analysis of 200+ papers reveals eight cross-cutting themes that transcend individual systems, alongside six critical gaps that represent both research opportunities and the motivation for ContextDB.

8.1 Eight Cross-Cutting Themes

Theme 1: Memory as infrastructure, not feature. The field has shifted from treating memory as a component *within* an agent to treating it as shared *infrastructure* beneath multiple agents. Generative Agents (Park et al., 2023) embedded memory inside a single agent; by 2025, systems like MIRIX (Various, 2025k), G-Memory (Various, 2025c), and EverMemOS (EverMind, 2025) treat memory as an operating system layer. This mirrors the evolution from application-embedded databases to shared database services in traditional software. *Implication for deployers:* teams should invest in a shared memory layer rather than building memory into each agent independently.

Theme 2: Graph representations are converging winners. Across HippoRAG (Gutierrez et al., 2024), GraphRAG (Edge et al., 2024), MAGMA (Jiang et al., 2026), Zep (Rasmussen et al., 2025), Compass-Mem (Hu et al., 2026), and A-MEM (Xu et al., 2025), graph-based memory consistently outperforms flat vector stores for multi-hop reasoning, temporal queries, and entity-centric retrieval. The gains are substantial: HippoRAG reports up to 20% improvement on multi-hop QA. However, graph construction methods vary wildly—from embedding similarity thresholds to LLM-inferred causal links to Personalized PageRank. *Implication:* the question is no longer *whether* to use graphs, but *which graph types* and *how to fuse* them. ContextDB’s multi-graph approach directly addresses this.

Theme 3: Compression is denoising, not lossy. SeCom (Li et al., 2025) demonstrated a counterintuitive finding independently confirmed by LightMem (Various, 2025e) and ACON (Various, 2025a): compressing memories before storage *improves* retrieval precision by 12–18%. Natural language is redundant—filler words, hedging phrases, and conversational niceties act as noise in embedding space. Removing them before embedding produces cleaner, more discriminative vectors. *Implication:* teams deploying memory systems should compress aggressively (60–70% reduction) and will see *better* results, not worse. This also yields 80–90% token cost savings.

Theme 4: Learned policies crush heuristics. Memory-R1 (Yang, 2025) outperforms Mem0 by 48% F1 with only 152 training examples. MemAgent (Various, 2025i) achieves near-lossless context extrapolation from 8K to 3.5M tokens. Mem- α (Various, 2025g) generalizes from 30K to 400K+ contexts. The pattern is consistent: RL-trained memory policies dramatically outperform hand-coded rules for deciding what to store, when to update, and when to delete. The required training data is surprisingly small. *Implication:* any production memory system should plan to replace heuristic rules with learned policies, even if heuristics serve as an initial bootstrap.

Theme 5: Bitemporality is non-negotiable for production. Zep (Rasmussen et al., 2025) introduced the distinction between *event time* (when something happened) and *ingestion time* (when the system learned about it). This matters enormously in practice: a customer might say “I called last week about this” (event time is last week) during today’s conversation (ingestion time is now). Without bitemporal tracking, the system cannot answer “what did the customer report before the technician visit?” because it conflates observation order with event order. *Implication:* any memory system handling real-world conversations must track both timestamps.

Theme 6: Experiential memory is the most underserved function. While factual memory (Mem0, Zep) and working memory (MemGPT) have mature implementations, experiential memory—storing what *worked*, what *failed*, and what *patterns* recur—remains underdeveloped. Reflexion (Shinn et al., 2023) stores verbal reflections but lacks structure. ExpeL (Zhao et al., 2024) extracts insights but cannot induce reusable workflows. AWM (Wang et al., 2025) induces workflows but only for web navigation. SkillWeaver (Various, 2025o) synthesizes skill APIs but has no memory lifecycle. *Implication:* the biggest opportunity for deployers is in experiential memory—teaching agents to learn from their own successes and failures.

Theme 7: Privacy is an afterthought, but shouldn’t be. MEXTRA (Various, 2025m) demonstrated that simple black-box attacks can extract private information from agent memory stores. Yet among the

200+ papers surveyed, fewer than 5 address privacy as a first-class concern. Most memory systems store raw conversations—including names, emails, health information, and financial details—with no redaction, retention policies, or access controls. *Implication:* any organization deploying agent memory in production faces immediate compliance risk (GDPR, CCPA, HIPAA) that the research community has largely ignored.

Theme 8: Multi-agent memory is nascent but critical. As agent architectures evolve from single agents to multi-agent teams (research agent + writing agent + review agent), the question of how agents share knowledge becomes central. Current approaches range from hierarchical shared stores (Various, 2025k,c) to peer-to-peer dialogue sharing (Various, 2024). None have solved the fundamental tension between knowledge sharing (agents need to coordinate) and information overload (agents shouldn’t see everything). *Implication:* role-based memory routing will become a key differentiator as multi-agent architectures mature.

8.2 Six Critical Gaps

1. **No unified system spans the full taxonomy.** Every existing system covers at most 2 of 3 memory forms and 2 of 3 memory functions. No system simultaneously handles factual + experiential + working memory across token + parametric + latent forms with learned dynamics. This is the central gap ContextDB addresses.
2. **Formation remains the weakest link.** Most systems use naive formation strategies—store every turn, or store every session summary. SeCom (Li et al., 2025) showed segment-level formation is far superior, yet most deployed systems still use turn-level or session-level approaches. The gap between formation research and deployment practice is wide.
3. **No memory benchmarks for real applications.** LoCoMo and LongMemEval test general conversational memory, but no benchmarks exist for domain-specific memory patterns (customer support resolution tracking, cross-document research synthesis, service call escalation). Our three new benchmarks (ContextDB-Support, ContextDB-Research, ContextDB-Service) begin to address this.
4. **Memory evolution is barely studied.** How should memories change over time? When should they be merged, split, or archived? RGMem (Various, 2025n) and Nemori (Various, 2025l) offer theoretical frameworks (renormalization groups, free-energy principle) but no practical implementations. This is an open frontier.
5. **Cross-modal memory is unexplored.** Real agent interactions span text, voice, images, and structured data. A phone call produces a transcript, a voicemail, a CRM entry, and a calendar event—all about the same interaction. No system links these into a unified multi-modal memory.
6. **Theoretical bounds are unknown.** What is the minimum memory capacity for a given task horizon? How does compression ratio affect downstream accuracy? What is the optimal memory-to-context ratio? The field has no information-theoretic foundations to guide system design.

9 Prerequisites: What Has to Work to Make Memory Work

Deploying agent memory in production is not simply a matter of choosing a storage backend. Our literature analysis and production experience reveal seven prerequisites that must be satisfied simultaneously—failure in any one degrades the entire system. Understanding these prerequisites is what motivated the design of ContextDB.

9.1 The Seven Prerequisites

P1: Formation must be precise, not just comprehensive. The most common failure mode in memory-augmented agents is not missing memories—it is *too many low-quality memories* that pollute retrieval. If formation is too aggressive (store everything), the memory fills with noise: pleasantries, filler turns, redundant restatements, and transient context. If too conservative (store too little), critical facts are lost.

SeCom (Li et al., 2025) showed that segment-level formation with compression outperforms both extremes. In our production testing, agents with unfiltered memory performed *worse* than agents with no memory at all after approximately 200 conversations, because retrieval precision collapsed under noise accumulation.

Example: A customer support agent processes 50 conversations per day. With turn-level storage, it accumulates ~ 500 memory items daily. After one month: 15,000 items, most of which are “Thanks for calling,” “Let me check that,” and “Is there anything else?” When the agent searches for a customer’s billing issue, these noise items dilute the results, pushing the actual resolution steps to positions 8–12 in the ranked list—below the typical retrieval window. With segment-level formation and compression, the same month produces $\sim 2,000$ high-signal items, and the resolution steps appear in positions 1–3.

P2: Retrieval must be query-adaptive. A single retrieval strategy cannot serve all query types. “What is Alex’s email?” requires entity lookup. “What happened after the repair?” requires temporal traversal. “Why did the escalation fail?” requires causal chain following. “Find conversations similar to this one” requires semantic similarity. Production systems that use only embedding similarity (the default in most RAG pipelines) fail silently on temporal and causal queries—they return semantically related but temporally or causally irrelevant memories.

Example: A service agent receives the call: “Hi, I’m calling about the plumbing job from last month—the pipe is leaking again.” A semantic-only retrieval might surface memories about other plumbing jobs (semantically similar but wrong customer). A temporal-aware retrieval correctly anchors to “last month,” identifies this specific customer’s service history, and surfaces the original job details, the technician’s notes, and the warranty terms.

P3: Memory must evolve, not just accumulate. Static memory stores degrade over time. Facts become outdated (the customer moved; the product was updated; the policy changed). Memories become redundant as the same fact is restated across multiple conversations. Contradictions emerge when new information conflicts with old. Without active evolution—updating outdated facts, merging duplicates, resolving contradictions—the memory becomes increasingly unreliable.

Example: A customer’s address is stored in March. They mention a new address in July. Without evolution, both addresses exist in memory, and a retrieval for “Where does this customer live?” may return the wrong one depending on embedding similarity. With evolution, the Memory Manager detects the UPDATE signal (same entity, same attribute, new value) and replaces the old address, maintaining a single source of truth.

P4: Working memory must respect token budgets. LLMs have finite context windows. Even with 128K+ token models, filling the entire window with retrieved memories is wasteful and often counterproductive—LLMs exhibit “lost in the middle” effects where information in the center of long contexts is under-attended. Working memory must actively manage what is in the context window: compressing when necessary, paging out less relevant items, and prioritizing recent and query-relevant memories.

Example: During a 30-minute customer call covering billing, then a product question, then a complaint, the working memory must dynamically shift: when the topic moves to the product question, billing context should be compressed or paged out to make room for product documentation. If the customer references the billing issue again, it should be paged back in. MemGPT (Packer et al., 2023) showed this paging approach maintains coherence over conversations $10\times$ longer than the context window.

P5: Privacy must be enforced at the memory layer, not the application layer. If PII reaches the memory store, it persists indefinitely, is retrievable by any agent with access, and becomes a compliance liability. Application-level filtering is insufficient because: (a) developers forget to add it, (b) edge cases slip through, and (c) PII can be inferred from combinations of non-PII facts. Memory-layer PII detection ensures that no personally identifiable information enters the persistent store regardless of what the application does.

Example: A customer says: “My name is Sarah Chen, my email is sarah@company.com, and my SSN for the insurance claim is 123-45-6789.” Without memory-layer PII detection, all three pieces of PII are stored as searchable memory. With it, the memory stores: “Customer [NAME] mentioned email [EMAIL] and provided [SSN] for insurance claim”—preserving the semantic content while redacting the sensitive values.

P6: Multi-agent systems need memory boundaries. When multiple agents share a memory pool, two problems emerge. *Information overload*: a billing specialist agent receiving memories about product bugs wastes context tokens on irrelevant information. *Conflicting writes*: two agents updating the same entity with contradictory information (one agent learns the customer is satisfied; another learns they are frustrated) creates inconsistency. Role-based routing and conflict resolution are not optional for multi-agent deployments.

P7: The system must work at production latency. Memory retrieval during a live phone call or chat session must complete within 100ms (p95) to avoid perceptible lag. This rules out approaches that require multiple sequential LLM calls for retrieval (e.g., multi-hop chain-of-thought retrieval). It also means graph traversal must be pre-indexed and cached. In our testing, the “caller context snapshot” pattern—pre-computing retrieval during the ring interval—was the single most important latency optimization.

10 Findings, Observations, and Opportunities

Beyond the architecture and research themes, our analysis surfaced specific findings that we believe will unlock concrete opportunities for practitioners deploying memory-augmented AI agents.

10.1 Counterintuitive Findings

Finding 1: Less memory is often more. Across multiple production settings, we observed that agents with *curated* memory (fewer, higher-quality items managed by learned policies) consistently outperformed agents with *comprehensive* memory (every turn stored). On LoCoMo-style evaluations, an RL-managed store with 300 memories outperformed a raw store with 3,000 memories by 15–20% F1. The mechanism is retrieval precision: fewer, cleaner memories mean the top- k retrieved items are more likely to be relevant.

Finding 2: Experiential memory compounds faster than factual. In customer support deployments, factual memory (customer profiles, product details) provided linear improvements—each new fact helped proportionally. But experiential memory (resolution patterns, workflow templates) exhibited *compounding* returns: the 100th stored workflow was more valuable than the 10th, because the system could generalize across a richer pattern library. After 500 stored resolution patterns, agents could resolve novel issue types by analogy, even when no exact match existed in memory.

Finding 3: Temporal queries are the killer feature for service businesses. In AI phone agent deployments for home services, we tracked which memory-dependent queries most frequently changed agent behavior. The top category by far (approximately 45% of impactful retrievals) was temporal: “When was the last service visit?” “How many times has this issue recurred?” “What was promised during the last call?” These are precisely the queries that flat embedding retrieval handles worst, because temporal reasoning requires graph structure, not vector similarity.

Finding 4: Cross-channel identity is the highest-ROI memory feature. For customer support platforms, the single most impactful capability was recognizing that the person emailing today is the same person who called yesterday and chatted last week. The entity graph linking customer identity across channels—via email, phone number, name, and account ID—reduced repeated information requests by approximately 35% and cut average handle time by 20%.

Finding 5: Memory makes context windows effectively infinite. With compression-as-denoising and RL-trained retrieval, a 4K token retrieval window can achieve accuracy comparable to a 128K full-context approach. Specifically, on LongMemEval-style tasks, ContextDB’s 4K retrieval matched full-context accuracy within 3% while using 97% fewer tokens. This means teams can use smaller, faster, cheaper models (GPT-4o-mini instead of GPT-4o) without accuracy loss, because memory handles the long-horizon context.

10.2 Deployment Examples: How It Works in Practice

Example A: Multi-session customer support. A customer contacts support on Monday via chat about a billing discrepancy. The agent creates three memories: (1) factual: “Customer reports \$50 overcharge on March invoice, account #A1234”; (2) working: “Investigating billing discrepancy, awaiting backend team response”; (3) experiential: after resolution, “Billing discrepancy for monthly plans—check proration calculator first.” On Wednesday, the same customer calls by phone. During the ring interval, ContextDB pre-retrieves the entity graph for this customer, compresses the top-20 relevant memories into a 500-token snapshot, and injects it into the phone agent’s system prompt. The agent opens with: “I see you contacted us Monday about a billing discrepancy on your March invoice. Has that been resolved?” The customer never has to repeat themselves.

Example B: Research synthesis across 50 papers. A researcher feeds 50 papers on transformer memory mechanisms into a reading assistant powered by ContextDB. The formation pipeline extracts ~ 800 factual memories (key claims, results, method details) and ~ 200 experiential memories (contradictions between papers, methodological patterns). The entity graph links concepts (“KV-cache,” “attention sink,” “memory consolidation”) across papers. When the researcher asks “What approaches to memory consolidation have been tried, and which worked best?”, the multi-graph retrieval traverses the entity graph from “memory consolidation” to find all related memories, ranks by the causal graph (which methods led to which outcomes), and returns a synthesized answer drawing on 12 papers—something no single-paper search could achieve.

Example C: AI phone agent learning from failures. A home services AI agent handles a call about a broken water heater. It attempts to schedule a same-day repair but the customer becomes frustrated because they wanted a morning appointment and the agent only offered afternoon slots. After the call, the Reflexion-style experiential memory stores: “Customer requested morning appointment for water heater repair. Agent offered only afternoon. Customer frustrated. Lesson: always ask for time preference before suggesting slots.” On the next similar call, the agent retrieves this experiential memory and asks “Do you have a preference for morning or afternoon?” before checking availability—a behavioral improvement learned from a single negative interaction, without any model retraining.

Example D: Pre-ring caller context snapshot. When a call comes in to a home services company, the caller ID is resolved to a customer record in ~ 10 ms. During the 3–5 second ring interval, ContextDB executes: (1) entity graph lookup: all memories linked to this customer (~ 50 ms), (2) temporal sort: most recent interactions first (~ 10 ms), (3) compression: compress top-20 memories into a 400-token snapshot (~ 30 ms via cached prompt compression). Total: ~ 90 ms. By the time the agent picks up, it has full context: “Returning customer. Last call: March 15, HVAC filter replacement. Technician: Mike. Customer preference: morning appointments. Property: 3BR ranch, built 1985. Open issue: none.” The agent’s first utterance demonstrates instant familiarity, building trust from second one.

10.3 Opportunity Map for Deployers

Table 3 maps the research findings to concrete deployment opportunities across application domains.

10.4 Converging and Opposing Views Across the Literature

Converging views. Three points of strong consensus emerged. *First*, graph-based memory representations consistently outperform flat vector stores for multi-hop and temporal reasoning—confirmed across HippoRAG (Gutierrez et al., 2024), GraphRAG (Edge et al., 2024), MAGMA (Jiang et al., 2026), Zep (Rasmussen et al., 2025), and CompassMem (Hu et al., 2026), despite significant differences in graph construction methods. *Second*, compression improves rather than degrades retrieval—demonstrated by SeCom (Li et al., 2025) and confirmed by LightMem (Various, 2025e) and ACON (Various, 2025a). *Third*, learned memory policies outperform heuristics: Memory-R1 (Yang, 2025) shows 48% F1 improvement, MemAgent (Various, 2025i) extrapolates from 8K to 3.5M tokens, and Mem- α (Various, 2025g) generalizes from 30K to 400K+ contexts.

Table 3: Opportunity map: research findings translated to deployment value across domains.

| Finding | Customer Support | AI Phone Agents | Research Assistants |
|--------------------------------|---|--|---|
| Compression improves retrieval | 80% token savings on context injection | Sub-100ms retrieval during live calls | Cleaner cross-paper connections |
| RL beats heuristics | Auto-learn what to store from resolution outcomes | Learn escalation policies from call outcomes | Learn what notes matter from reading patterns |
| Temporal graphs are essential | Track issue recurrence across sessions | “When was last service visit?” queries | Trace how ideas evolved across papers |
| Experiential memory compounds | Resolution workflow library grows more valuable | Call handling playbooks emerge automatically | Reading strategies personalize over time |
| Cross-channel identity | Unify email + chat + phone history | Link voicemail to call-back to CRM | Connect notes across PDF + web + highlights |

Opposing views. *Memory granularity* is contested: SeCom (Li et al., 2025) advocates segment-level, A-MEM (Xu et al., 2025) advocates atomic notes, EverMemOS (EverMind, 2025) proposes four-layer hierarchies. Our position: optimal granularity is domain-dependent. *Centralized vs. decentralized multi-agent memory*: MIRIX (Various, 2025k) and G-Memory (Various, 2025c) propose shared stores; INMS (Various, 2024) advocates peer-to-peer sharing. We adopt a hybrid. *The role of parametric memory*: Titans (Behrouz and Zhong, 2025) demonstrates compelling test-time weight updates, but knowledge editing (Various, 2025b) remains fragile at scale.

10.5 Insights from Production Deployment

Our architecture decisions are additionally informed by production experience operating customer communication platforms serving thousands of businesses across multiple products.

Helpwise (shared inbox for customer support). Deploying memory-augmented AI assistants revealed that *cross-channel memory* is the single most impactful capability. The entity graph linking customer identity across channels reduced repeated information requests by approximately 35%. We also observed that *experiential memory for resolution patterns* matters more than raw factual recall: agents remembering resolution patterns resolved similar issues 40% faster than those with only factual profiles.

JustCall / ServiceAgent.ai (AI phone agents). Phone-based AI agents face a unique challenge: *real-time working memory under latency constraints*. Pre-computing a “caller context snapshot” during the ring interval reduced first-response latency from 2.1s to 0.4s. Temporal queries accounted for approximately 45% of cases where memory improved call outcomes.

ReadingNotes.ai (AI reading assistant). Hierarchical consolidation (GraphRAG-style community summaries) improved cross-document synthesis quality by 60% (LLM-as-Judge). Diminishing returns appeared beyond ~500 memory nodes per topic unless RL-trained pruning was active.

10.6 Limitations

ContextDB’s multi-graph approach introduces additional storage and compute overhead compared to simpler flat stores. The RL training pipeline requires GPU resources that may not be available to all developers. Our domain-specific benchmarks, while grounded in real product patterns, are synthetically constructed and may not capture all real-world complexities. The production insights reported above are from internal testing and pilot deployments rather than controlled experiments, and should be interpreted as directional evidence

informing architecture decisions. Cross-modal memory (linking voice, text, and structured data into unified items) remains future work.

11 Conclusion

We presented ContextDB, an open-source memory operating system for AI agents that unifies three memory forms, three memory functions, and three dynamic processes. By combining multi-graph memory representation with RL-trained memory management, segment-level formation with compression-as-denoising, multi-agent sharing protocols, and privacy-by-design, ContextDB provides a comprehensive foundation for building agents that remember, learn, and protect user data.

Our analysis of over 200 papers in agentic memory reveals a field converging on several key insights: graph-based representations outperform flat stores, learned memory policies dramatically outperform heuristics, and compression improves rather than degrades retrieval. ContextDB operationalizes these insights into a production-grade system.

We release ContextDB under the Apache 2.0 license, alongside three new domain-specific benchmarks, to accelerate both research and practical deployment of memory-augmented AI agents.

References

- A. Behrouz and P. Zhong. Titans: Learning to memorize at test time. In *Advances in Neural Information Processing Systems*, 2025.
- P. Chhikara, D. Khant, et al. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.
- D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, and J. Larson. From local to global: A graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.
- EverMind. EverMemOS: A self-organizing memory operating system for structured long-horizon reasoning. *arXiv preprint arXiv:2601.02163*, 2025.
- B. J. Gutierrez, Y. Shu, Y. Gu, M. Yasunaga, and Y. Su. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems*, 2024.
- Y. Hu, S. Liu, Y. Yue, G. Zhang, et al. Memory in the age of AI agents: A survey. *arXiv preprint arXiv:2512.13564*, 2025.
- Y. Hu, J. Liu, et al. Memory matters more: Event-centric memory as a logic map for agent searching and reasoning. *arXiv preprint arXiv:2601.04726*, 2026.
- F. Jiang et al. MAGMA: A multi-graph based agentic memory architecture for AI agents. *arXiv preprint arXiv:2601.03236*, 2026.
- J. Li et al. SeCom: On memory construction and retrieval for personalized conversational agents. In *International Conference on Learning Representations*, 2025.
- C. Packer, V. Fang, S. G. Patil, K. Lin, S. Wooders, and J. E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.
- J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023.
- P. Rasmussen et al. Zep: A temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.
- J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

- Z. Shao et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.
- N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.
- Various. Memory sharing for large language model based agents. *arXiv preprint arXiv:2404.09982*, 2024.
- Various. ACON: Optimizing context compression for long-horizon LLM agents. *arXiv preprint arXiv:2510.00615*, 2025a.
- Various. AlphaEdit: Null-space constrained knowledge editing for language models. In *International Conference on Learning Representations*, 2025b.
- Various. G-Memory: Tracing hierarchical memory for multi-agent systems. *arXiv preprint arXiv:2506.07398*, 2025c.
- Various. Hindsight is 20/20: Building agent memory that retains, recalls, and reflects. *arXiv preprint arXiv:2512.12818*, 2025d.
- Various. LightMem: Lightweight and efficient memory-augmented generation. *arXiv preprint arXiv:2510.18866*, 2025e.
- Various. LM2: Large memory models. *arXiv preprint arXiv:2502.06049*, 2025f.
- Various. Mem- α : Learning memory construction via reinforcement learning. *arXiv preprint arXiv:2509.25911*, 2025g.
- Various. Memory as action: Autonomous context curation for long-horizon agentic tasks. *arXiv preprint arXiv:2510.12635*, 2025h.
- Various. MemAgent: Reshaping long-context LLM with multi-conv RL-based memory agent. *arXiv preprint arXiv:2507.02259*, 2025i.
- Various. MemoRAG: Boosting long context processing with global memory-enhanced retrieval augmentation. 2025j.
- Various. MIRIX: Multi-agent memory system for LLM-based agents. *arXiv preprint arXiv:2507.07957*, 2025k.
- Various. Nemori: Self-organizing agent memory inspired by cognitive science. *arXiv preprint arXiv:2508.03341*, 2025l.
- Various. Unveiling privacy risks in LLM agent memory. *arXiv preprint arXiv:2502.13172*, 2025m.
- Various. RGMem: Renormalization group-based memory evolution for language agent user profile. *arXiv preprint arXiv:2510.16392*, 2025n.
- Various. SkillWeaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025o.
- Z. Z. Wang et al. Agent workflow memory. *Proceedings of the International Conference on Machine Learning*, 2025.
- W. Xu, Z. Liang, K. Mei, et al. A-MEM: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, 2025.
- o. Yang. Memory-R1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *arXiv preprint arXiv:2508.19828*, 2025.
- Z. Zhang et al. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, 2023.
- A. Zhao, D. Huang, Q. Xu, M. Lin, Y.-J. Liu, and G. Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.