Hydra DB: Beyond Flat Embeddings for Production AI Agents

Soham Ratnaparkhi, Nishkarsh Srivastava, Aadil Garg, Pratham Garg

Engineering, Hydra DB, San Francisco, California, United States of America.

Contributing authors: soham@hydradb.com; nishkarsh@hydradb.com; aadil@hydradb.com; pratham@hydradb.com;

Abstract

As Large Language Models transition from ephemeral chat interfaces to persistent agentic systems, two foundational problems remain unsolved. First, agents have no continuity across sessions: every interaction begins from zero, with no awareness of prior decisions, no history of how facts have changed, and no understanding of the user they are serving. Second, enterprise knowledge bases remain trapped in flat vector indexes based on fragmented chunks that reduce all knowledge to similarity scores. This results in collapsed relationships which are essential for accurate reasoning. We present **Hydra DB**, a context engine for AI agents that models knowledge as versioned, relational, time-aware state. Agents ingest operational history and knowledge bases; Hydra DB resolves entities, versions every transition, and preserves the full relational record of how context has evolved. This enables agents to answer not just what is currently true, but what was true, when it changed, and why. At its core, Hydra DB combines a *Sliding Window Inference Pipeline* for contextually self-contained ingestion with a *Git-style* versioned knowledge graph that makes every state transition a first-class, addressable record. We evaluate on the **LongMemEval-s** benchmark, where Hydra DB achieves a state-of-the-art (SOTA) accuracy of **90.79%**, significantly outperforming all established baselines and demonstrating robust, model-agnostic performance across Gemini 3.0 Pro, GPT-5.2, and GPT-5 Mini.

Keywords: agent context engine, versioned knowledge graph, temporal reasoning, knowledge base retrieval, LongMemEval-s, SOTA

1 Introduction

The current trajectory of Artificial Intelligence is defined by a shift from stateless, single-turn interactions to autonomous, multi-turn agentic sessions. While the industry has seen a rapid expansion in LLM context windows, this "brute force" scaling introduces extremely high computational costs[1] and remains susceptible to the "lost-in-the-middle" phenomenon[2], where information density degrades over long horizons. LLMs either forget or become extremely slow and costly after the length of a conversation increases beyond a threshold. Beyond these issues, long-running interactions suffer from *context rot* [3], a gradual degradation in the usefulness of earlier context as conversations grow. As irrelevant, stale, or weakly related information accumulates, models struggle to distinguish salient facts from noise, leading to diminished recall, incorrect reasoning, and unstable behavior even before hard context limits are reached. More fundamentally, extending the context window solves the wrong problem. An agent with a longer context-window is still stateless; it simply forgets more slowly and yet is often confused about the user and their environment. Every new session starts from zero: no accumulated context, no record of prior decisions, no awareness of how the user or their environment has changed.

Existing memory-augmented systems typically address this via Retrieval-Augmented Generation (RAG). However, standard RAG implementations are architecturally predisposed to failure in real-world environments due to two primary factors:

- **Semantic Fragmentation**: Naive chunking strategies isolate facts [4]. When an entity is introduced in one segment and its attributes are updated in another, the system often fails to synthesize a unified identity, leading to "hallucinations of omission."
- **Temporal Stagnation**: Most RAG approaches [5] treat their index as a flat, chronology-agnostic substrate. They cannot distinguish between obsolete facts and current truths (e.g., "Where did I live in 2023?" vs. "Where do I live now?"), resulting in stale context injection. This failure compounds as the knowledge base grows: a policy updated six months ago and the current version live in the same flat index, indistinguishable by any similarity metric.

Beyond these structural limitations, standard RAG systems fail to "understand" sentiments. When a user states, "I absolutely hate React; it's a nightmare to debug," typical RAG may preserve the factual statement but loses the intensity and emotional understanding critical for personalized agent responses.

We introduce **Hydra DB**, the context engine for AI agents. Where vector stores alone reduce knowledge to similarity scores, Hydra DB goes further by modeling context as a versioned relational record that captures entities, relationships, decisions, and the complete timeline of how each has evolved. Vector search remains part of the pipeline, but it is one signal among many: agents query against a hybrid substrate that understands not just what is similar, but what is actually connected, when it changed, and why. At the core of Hydra DB is a Dual-Engine Preprocessing Pipeline that transforms raw input,into an enriched, relational knowledge substrate. Our architecture introduces a Sliding Window Inference Pipeline, which ensures every ingested

record is self-contained with resolved entities and contextual bridges. Simultaneously, we map these records into a Git-style versioned temporal graph, tracking the evolution of facts and decisions over time without the data loss associated with traditional overwriting.

This report covers our evaluation on LongMemEval-s. We tested across all five context capabilities: information extraction, multi-session reasoning, knowledge updates, temporal reasoning, and knowing when to say "I don't know." LongMemEval evaluates Hydra DB across five distinct cognitive abilities essential for production-grade AI agents by utilizing 500 question–conversation stacks, with an average length per stack exceeding 115k tokens.

"LongMemEval"[6] serves as the ultimate stress test for long-context retrieval, pushing retrieval frameworks beyond simple "needle-in-a-haystack" lookups (common in LoCoMo[7]) to assess their ability to synthesize information across massive, multi-turn conversation histories. The conversations in LoCoMo average around 16,000–26,000 tokens, as opposed to 115,000 tokens in LongMemEval. LoCoMo fails to test long-horizon retrieval under pressure; therefore, we benchmarked Hydra DB on LongMemEval[8].

Hydra DB sets a new benchmark with **90.79%** overall accuracy, outperforming the previous current best by +5 points. The most significant gains appear in single-session-user & single-session-assistant (**100% accuracy**), temporal reasoning (**90.97%**), preference understanding (**+80%**), and knowledge updates (**97.4%**). More about our evaluations in Section 3.2.1.

2 Methodology

Standard RAG [5] approaches treats operational history, knowledge base entries, structured records as a flat collection of text chunks, indexed by an embedding model and retrieved using cosine similarity alone. This fails in production because agent context is multi-dimensional: it is simultaneously semantic (what something means), relational (how it connects to everything else), and chronological (when it was true). Flattening all three into a similarity score produces a system that actively degrades as the knowledge base grows.

To bridge this gap, Hydra DB introduces a **Composite Context protocol**. We replace the flat index of traditional systems with a composite substrate that fuses a **Git-Style Temporal Graph for relational integrity** with a **High-Dimensional Vector Substrate** for semantic breadth.

2.1 Ontological Structure vs. the Flat Index Problem

A fundamental architectural assumption in standard RAG systems is that semantic proximity implies informational relevance. This assumption is false in production. Two facts can be semantically distant yet causally linked (a user's career switch and their subsequent relocation), or semantically close yet factually orthogonal ("I love Python" and "I used to love Python"). Vector stores, by design, cannot distinguish between these cases. They reduce all knowledge to a *flat index* — a high-dimensional soup of embeddings where the only retrieval primitive is cosine similarity. This imposes a

Table 1: Benchmark Categories and Question Distribution for Hydra DB Evaluation

Benchmark Category	What It Determines	Why We Test This	# Questions
Single-Session Information Extraction	Ability to recall explicit facts stated once within large, noisy conversations.	Ensures Hydra DB can reliably extract precise evidence required for production-grade agents, regardless of contextual noise.	70
Single-Session Preference Extraction	Ability to identify and retain user preferences expressed within a single session.	Validates correct personalization based on explicit preference signals without cross-session reasoning.	30
Single-Session Assistant Recall	Ability to recall facts introduced by the assistant itself within a session.	Ensures assistant-generated information remains accessible for downstream reasoning and follow-up queries.	56
Multi-Session Reasoning	Ability to combine facts distributed across multiple sessions.	Evaluates whether the system can link semantically related facts across temporally and contextually disconnected sources, simulating human-like associative memory.	133
Temporal Reasoning	Ability to preserve and reason over the chronology of stored information.	Validates time-aware indexing to prevent obsolete or incorrect responses in time-sensitive queries (e.g., residence history, financial states).	133
Knowledge Updates	Ability to overwrite outdated facts when user preferences or states change.	Tests dynamic personalization, ensuring responses always reflect the most recent user intent or profile updates.	78
Abstention (Safety)	Ability to identify insufficient information and refuse to answer.	Prevents hallucinations and enforces trust by ensuring the system does not fabricate answers when evidence is missing.	–
Total (Answerable)			**500**

fundamental ceiling on the kinds of queries an agent can answer. Microsoft Research has demonstrated that this baseline approach "struggles to connect the dots" when answers require traversing disparate pieces of information, and fails altogether on questions requiring holistic understanding across large corpora [9].

To illustrate the failure mode: a user's dietary preference, their reason for changing it, their family context, and their medical history may each reside in separate chunks with no embedding-level proximity to one another. Vector retrieval treats these as independent facts. But for a production-grade agent tasked with meal planning or health monitoring, the *connection* between these facts is the answer — not any individual fact in isolation.

Hydra DB addresses this by treating all ingested knowledge as **unified relational context** rather than just a flat vector index. The question shifts from "what is similar?" to "what is actually connected, when did that connection form, and has it changed?" This distinction has three concrete consequences for agent behavior.

2.1.1 Structured Relational Index over a Flat Embedding Space

Where vector databases index chunks as independent floating-point vectors, Hydra DB indexes knowledge as a typed graph of entities and relationships. Each entity — a person, project, system, preference, or decision — is a first-class node. Each relationship carries a semantic type (`WORKS_AT`, `PREFERS`, `CAUSED_BY`, `BLOCKED_BY`), a natural language context string, and temporal metadata. This structure enables *deterministic*, multi-hop traversal that is impossible in a flat index.

Consider the query: *"Why is the authentication service behaving differently since last month?"* Vector retrieval may surface recent logs mentioning the service. But the graph can traverse: `auth-service` $\xrightarrow{\text{DEPENDS\_ON}}$ `user-db` $\xrightarrow{\text{MODIFIED\_BY}}$ `migration-v2` $\xrightarrow{\text{AUTHORED\_BY}}$ `alice` $\xrightarrow{\text{CAUSED\_BY}}$ `schema-change-ticket`, recovering the full causal chain without any of these hops being co-located in embedding space.

This is the core architectural advantage of an ontology based context index: the graph makes *distant but causally connected* facts retrievable, while preserving the semantic retrieval capabilities of the vector substrate for cases where entity identity is ambiguous.

2.1.2 Graph-Derived Conclusions as Universal Context Signals

A persistent blind spot in RAG-based retrieval systems is the loss of *reasoning context* around state changes. When a user updates a fact — switching jobs, changing a preference, revising a plan — standard systems record the new state and discard the old one, or at best store both without linkage. The downstream agent then cannot answer: *"Why did I switch from React to Vue?"* or *"What was I optimizing for when I made that architectural decision?"* — because the *decision trace* was never encoded.

Hydra DB's append-only graph architecture natively encodes decision traces as part of every state transition. When a new edge e_k is committed to $\mathcal{E}(u, v)$, the accompanying $\mathcal{C}_{\text{meta}}$ field preserves the reasoning context, sentiment, and situational factors surrounding that transition (see Section 2.2). This means the graph does not merely record that a user *changed* their preference — it records *why they changed it*, what alternatives were considered, and what outcome they were optimizing for.

Critically, because these decision traces are encoded as structured graph relationships rather than buried in free-form text chunks, Hydra DB can *synthesize conclusions* from the topology of the graph itself — independent of any single retrieved chunk. A pattern of edges such as `user` $\xrightarrow{\text{REJECTED}}$ `cloud-vendor-A`, `user` $\xrightarrow{\text{REJECTED}}$ `cloud-vendor-B`, `user` $\xrightarrow{\text{OPTIMIZES\_FOR}}$ `data-sovereignty` allows the system to infer a vendor preference that was never explicitly stated. These graph-derived conclusions propagate as enriched context signals across the entire retrieval pipeline — improving not just the specific query that triggered them, but every downstream interaction

that touches related entities. The result is a context engine that compounds in value with use: the more the graph is traversed, the more latent structure it surfaces, and the more coherent the agent's responses become across all domains.

2.1.3 Context Accumulation Across Sessions and Knowledge Sources

Vector databases treat each session as stateless with respect to preference learning. Any personalization is contingent on retrieving the specific chunk that mentioned a preference — which fails when preferences are implicit, distributed across sessions, or expressed indirectly. Hydra DB's graph accumulates user preferences and outcomes as structured, typed relationships that persist and compound across the full interaction history.

For example, a user who repeatedly accepts recommendations involving open-source tooling, declines SaaS suggestions, and expresses cost-sensitivity across unrelated sessions generates a preference subgraph that encodes these patterns as explicit edges: `user` $\xrightarrow{\text{PREFERS}}$ `open-source`, `user` $\xrightarrow{\text{AVOIDS}}$ `SaaS`, `user` $\xrightarrow{\text{OPTIMIZES\_FOR}}$ `cost-efficiency`. These nodes accumulate confidence over time and serve as structured priors for downstream reasoning.

This accumulation spans both operational history and the knowledge base. A preference expressed in conversation and a constraint documented in the knowledge base contribute to the same entity subgraph - resolved, typed, and queryable as a single unified record.

Critically, outcome signals - whether a recommendation was acted upon, whether a plan succeeded, whether a decision was later reversed - are encoded as edge-level annotations on the preference subgraph. This transforms the retrieval layer from a passive record of what was said into an active model of what the user values, enabling agents to move from *retrieving stated preferences* to *reasoning over demonstrated outcomes*.

2.2 Temporally-Aware Context Graph

We employ a Git-Style Versioned Knowledge Graph to store entities and relationships with immutable contextual history. Standard RAG systems suffer from the "State Confusion Problem." For example, suppose a user says in 2022: "I live in New York because I work at startup XYZ, which is headquartered in NYC." Later, in 2024, they say: "I live in London because I now work at Meta; I switched because of the work culture, and I moved to be closer to my parents." In a traditional vector store, these statements collide: the system either retrieves both without knowing which is current, or it overwrites the earlier fact. As a result, the downstream agent loses the timeline, reasoning (what changed, when it changed, and why) and the associated decision tree. Hydra DB implements a Temporal-State Multigraph. We treat the Knowledge Graph $\mathcal{G}$ as an immutable, append-only context ledger for cognition, similar to a Git repository's commit history where every transition, across any source, is a versioned, addressable commit.

2.2.1 Challenge: The Destructive Update Problem

Many contemporary retrieval systems attempt to manage state through an Iterative Resolution Loop. In this pattern, the system performs a vector search for every incoming chunk to find "similar" existing facts, then asks an LLM to decide whether to update or delete the old record. We rejected this architecture for two reasons:

- **Instability via False Positives**: Semantic similarity does not imply factual redundancy. "I love Python" and "I used to love Python" are semantically close but chronologically distinct. Relying on an LLM to "resolve" this often leads to a False Positive Delete, where historical context is dangerously purged based on a probabilistic hallucination.
- **The $O(N)$ Latency Trap**: Triggering a retrieval-and-reasoning step for every ingested chunk (N lookups for N inputs) introduces prohibitive latency and cost at scale; hence it is not scalable.

Instead of nondeterministic overwrites, Hydra DB adopts a Git-Style Append-Only Log. We treat context as an immutable ledger. If a user moves from "NYC" to "London" we do not "update" the node/edge; we commit a new edge with a fresh temporal information. This guarantees zero data loss and enables our system to answer queries ("What places I visited last year?" or "From where and why did I make a professional career switch?") - a capability impossible in systems that destructively resolve conflicts. Another benefit of this approach is that it provides downstream AI agents with a highly detailed, temporally aware decision tree to answer *why a particular decision was taken*.

2.2.2 Solution

We define a relationship R between two entities u and v not as a single static edge, but as a time-ordered sequence of state changes, analogous to versioned *commits*.

Let $\mathcal{E}(u, v)$ denote the set of all edges connecting entities u and v. Each edge $e_k \in \mathcal{E}(u, v)$ is represented as a tuple:

$$e_k = (r_k, t_{\text{commit}}, t_{\text{valid}}, \mathcal{C}_{\text{meta}}). \tag{1}$$

Here:

- r_k denotes the semantic relation (e.g., *located_in*, *prefers*).
- t_{commit} represents the ingestion timestamp corresponding to the interaction time.
- t_{valid} denotes the extracted temporal validity, reflecting real-world time (e.g., "in 1999").
- $\mathcal{C}_{\text{meta}}$ contains auxiliary metadata, including contextual information, the relation, etc.

When a knowledge update occurs (e.g., a user changes their diet from *Omnivore* to *Vegan*), the system does not mutate or overwrite the previous edge e_{k-1}. Instead, a new edge e_k is appended to $\mathcal{E}(u, v)$. This strictly monotonic growth enables *Differential*

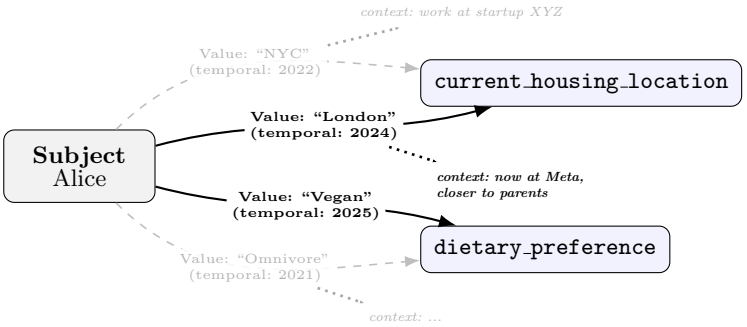

Fig. 1: Temporal-State Graph Topology with Contextual Metadata

Reasoning over relational states. We formalize the current relational state as:

$$\Delta\text{State}(u, v) = \text{SortByTime}(\mathcal{E}(u, v)) \,\big|\, t \leq t_{\text{now}}. \tag{2}$$

2.3 Sliding Window Inference Pipeline

Standard document chunking often creates *blind segments*, where a chunk c_i loses dependencies on neighboring chunks c_{i-k} or c_{i+k}. To address this limitation, we introduce a *Sliding Window Inference Pipeline*.

2.3.1 Challenge: The Orphaned Pronoun Paradox

During early development, we observed that standard chunking (e.g., recursive character splitting) rendered nearly 40% of chunks semantically invisible. If a user said, "I hate that framework," and a couple of chunks back, the framework being discussed is "React," the isolated chunk (revealing the user's feelings) becomes largely useless. Standard vector search could never map "that framework" to "React." We initially attempted to solve this with larger overlap windows, but this merely increased downstream token costs without solving the core problem. Hence, we experimented with the following solution.

2.3.2 Solution

Let D denote a conversational session represented as a sequence of tokens. We partition D into a sequence of base segments:

$$S = \{s_1, s_2, \ldots, s_n\}. \tag{3}$$

For each segment s_i, we construct a context window W_i that includes a lookback horizon h_{prev} and a lookahead horizon h_{next}:

$$W_i = [s_{i-h_{\text{prev}}}, \ldots, s_i, \ldots, s_{i+h_{\text{next}}}]. \tag{4}$$

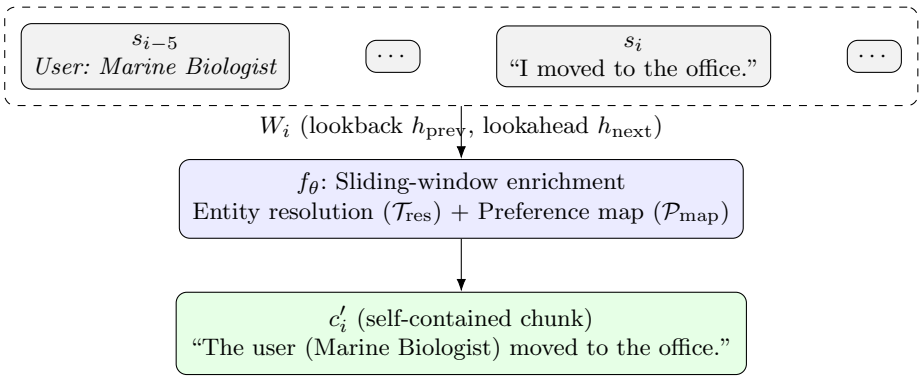

Fig. 2: Sliding Window Inference Pipeline: enriching a segment s_i using a broader context window W_i to produce a self-contained chunk c'_i with resolved entities.

The enrichment process is modeled as a transformation function f_θ, parameterized by a lightweight language model, which maps a raw segment s_i and its context window W_i to an enriched chunk c'_i:

$$c'_i = f_\theta(s_i \mid W_i) = \{\mathcal{T}_{\text{res}}, \mathcal{P}_{\text{map}}, s_i\}. \tag{5}$$

Here:

- $\mathcal{T}_{\text{res}}$ denotes *Entity Resolution*, where implicit references (e.g., pronouns such as "he" or underspecified mentions like "the project") are replaced with explicit, uniquely identifiable entities inferred from W_i.
- $\mathcal{P}_{\text{map}}$ represents *Preference Mapping*, which extracts persistent user constraints and conclusions from sequences that may influence future interactions.

This transformation ensures that each enriched chunk c'_i is semantically self-contained. For example, if s_i contains the statement "I moved to the office," and an earlier segment s_{i-5} establishes the user as a *Marine Biologist*, the enriched chunk explicitly embeds this information as: "The user (Marine Biologist) moved to the office."

2.4 Experimental Framework: Bio-Mimetic Context Consolidation

A fundamental limitation of current RAG systems is the assumption that "more data is better." In reality, unbounded record growth — particularly in deployments ingesting large knowledge bases — leads to retrieval latency and semantic drift, where outdated records surface ahead of current context. To address this, we are currently experimenting with a Bio-Mimetic Decay Engine that mimics the process of synaptic pruning and consolidation. Our hypothesis is that the database should not be a static archive but a lifecycle-managed context store, governed by a Retention Score ($\mathcal{R}$).

2.4.1 The Time-Frequency Decay Function

We are experimenting with a decay mechanism inspired by the Ebbinghaus Forgetting Curve, augmented with reinforcement learning principles. Rather than modeling retention solely as a function of age, the proposed formulation incorporates the *utility* of a record based on its usage frequency.

The retention score of a context record m at time t is defined as:

$$\mathcal{R}(m,t) = \underbrace{I_{\text{salience}}}_{\text{Initial Importance}} \cdot \underbrace{e^{-\lambda \Delta t}}_{\text{Temporal Decay}} + \underbrace{\sigma \sum_{i=1}^{n} \frac{1}{t - t_{\text{access}_i}}}_{\text{Reinforcement Boost}}. \tag{6}$$

Here:

- I_{salience} denotes the amalgamation of initial importance assigned to a record during preprocessing and its retrieval frequency, where high-impact facts (e.g., *medical allergies*) receive higher scores than low-impact facts (e.g., *coffee orders*).
- Δt represents the elapsed time since the record was first created.
- λ controls the rate of exponential decay.
- σ is a reinforcement scaling factor.
- t_{access_i} denotes the timestamp of the i-th successful retrieval of record m.

The reinforcement term ensures that each successful retrieval strengthens the retention trace, effectively resetting and elevating its decay curve. This behavior mirrors reinforcement-based retention, preventing high-signal records from being evicted regardless of chronological age.

2.4.2 Tiered Storage Architecture

In this approach, records that are candidates for eviction move through a tiered storage process, progressing into lower-priority storage levels and becoming less likely to be retrieved, until they are ultimately evicted. A record is demoted one tier when its Retention Score ($\mathcal{R}$) falls below a configured threshold.

2.5 The High-Dimensional Vector Substrate

While the Graph (Section 2.2) maintains structural & relational integrity, the core semantic recall relies on a heavily optimized vector substrate. Hydra DB utilizes a Multi-Field Hybrid Schema within a self-hosted vectorstore. For every ingested record — whether a conversation turn or a knowledge base entry — we index three representations: Raw Content ($\mathbf{v}_{content}$), Sparse Keywords ($\mathbf{v}_{sparse}$), and Latent Context ($\mathbf{v}_{latent}$). Crucially, $\mathbf{v}_{latent}$ is not an arbitrary projection; it is the direct vectorization of context from Section 2.3. This ensures that the dependencies and resolved entities captured during ingestion are physically embedded into the search space.

2.5.1 Challenge: The Vocabulary Mismatch Gap

A persistent failure mode in RAG is the disconnect between user intent and system logs. A user might ask, "Why is the app behaving strangely?" (Abstract Intent). The relevant record might contain "Error 503: Service Unavailable" (Technical Fact). Standard embedding models place these two strings far apart in vector space because they share no lexical or immediate semantic overlap.

2.5.2 Our Solution

We realized that standard embedding models are "literal-minded." To bridge this gap, we introduced Latent Semantic Bridging. By embedding the contextual aware output which contains the contextual implications of a chunk rather than just its raw text - we effectively "pre-compute" the answer. This allows an abstract query to latch onto the meaning of the event (captured in $\mathbf{v}_{latent}$) even if the description of the event (captured in $\mathbf{v}_{content}$) is technically obscure.

2.6 Recall Pipeline

Hydra DB employs a *Multi-Stage Pipeline* that combines hybrid semantic search with a versioned context graph. Records are stored as enriched chunks carrying entity, relationship, and temporal metadata, enabling retrieval to leverage both unstructured semantic similarity and structured relational context via the contextual Graph.

Given a user query, Hydra DB constructs relevant context by progressively searching through candidate objects using the vectors and graph. This design enables Hydra DB to retrieve not only semantically similar content, but also temporally and relationally relevant records that are difficult to recover using vector similarity alone.

2.6.1 Adaptive Query Expansion (Multi-Query)

Hydra DB treats the user query q as a semantic seed rather than a fixed string. We apply an LLM-based projection function $\Phi(q)$ to generate a set of N semantically diverse reformulations:

$$Q' = \{q_1, q_2, \ldots, q_N\}$$

Each q_i captures a distinct interpretation of the user's intent, including paraphrases, temporal concretizations, and domain-specific restatements. For example, the query *"What did I do last week?"* may be expanded into:

- "Projects worked on in the last 7 days"
- "Commits pushed during the previous week"
- "Meetings or tasks completed last week"

All expanded queries are executed in parallel against the retrieval backend. This ensures high recall even when relevant records differ significantly in surface phrasing from the original query, a common failure mode in single-query vector retrieval.

2.6.2 Weighted Hybrid Search: The Retrieval Equation

Unlike systems that rely solely on cosine similarity, Hydra DB performs **Weighted Rank Fusion** at the database level. For a query q and candidate chunk c, the initial retrieval score is computed as a linear combination of three complementary signal paths:

$$S_{\text{retrieval}}(q, c) = \underbrace{x \cdot \text{sim}(q, \mathbf{v}_{\text{content}})}_{\text{Primary Dense}} + \underbrace{y \cdot \text{sim}(q, \mathbf{v}_{\text{inferred}})}_{\text{Secondary Dense}} + \underbrace{\alpha \cdot \text{BM25}(q, \mathbf{v}_{\text{sparse}})}_{\text{Sparse Signal}} \quad (7)$$

where:

- $\mathbf{v}_{\text{content}}$ is the dense embedding of the literal chunk content.
- $\mathbf{v}_{\text{inferred}}$ represents inferred or latent semantics extracted during ingestion.
- $\mathbf{v}_{\text{sparse}}$ corresponds to sparse lexical features used by BM25.

The Primary Dense signal captures direct semantic similarity, while the Secondary Dense signal (weighted at x) captures implicit meaning that may not be explicitly stated. The Sparse Signal, weighted by α, ensures that rare but critical tokens (e.g., project IDs, issue numbers, or usernames) strongly influence retrieval, preventing semantic drift toward loosely related but incorrect records.

2.6.3 Graph-Augmented Retrieval: Entity-Based Search

In parallel with hybrid vector retrieval, Hydra DB executes a graph-based retrieval pass over its versioned context graph. Entities E are extracted from the query q, and the system performs exact entity name matching followed by bounded, variable-length path traversal:

$$P_{\text{graph}} = \text{Path}(E_{\text{start}} \xrightarrow{*1..n} E_{\text{end}}) \quad (8)$$

For each retrieved path, we construct structured graph context by concatenating node names, relation context, and temporal metadata:

$$\text{context}_{\text{graph}}(p) = \text{concat}(\text{node}_{\text{name}}, \text{relation}_{\text{context}}, \text{temporal}_{\text{details}}) \quad (9)$$

These graph contexts are then reranked using a cross-encoder applied to $(q, \text{context}_{\text{graph}}(p))$, producing a relevance score:

$$S_{\text{graph}}(p) = S_{\text{semantic}}(q, \text{context}_{\text{graph}}(p)) \quad (10)$$

where the cross-encoder evaluates the combination of entity names, relational context, and temporal information against the query. This ensures that graph retrieval captures relational dependencies and temporal sequences that may not be present within any single text chunk (e.g., "Project A is blocked by Issue B").

2.6.4 Chunk-Level Graph Expansion

Beyond query-anchored entity search, Hydra DB performs a second-stage **chunk-level expansion** to avoid the need for post-hoc entity extraction from vector results. During

the ingestion phase, each chunk c is pre-linked to its referenced entities. At retrieval time, for each chunk c retrieved by the vector store, the system directly accesses its pre-indexed entity references $E(c)$ and explores their adjacent graph neighborhoods up to depth n:

$$\mathcal{N}(c) = \bigcup_{e \in E(c)} \text{Path}(e \xrightarrow{*1..n}) \tag{11}$$

Each expanded path is converted to structured context using the same concatenation approach:

$$\text{context}_{\text{expansion}}(p) = \text{concat}(\text{node}_{\text{name}}, \text{relation}_{\text{context}}, \text{temporal}_{\text{details}}) \tag{12}$$

These expansion contexts are then reranked independently:

$$S_{\text{expansion}}(p) = S_{\text{semantic}}(q, \text{context}_{\text{expansion}}(p)) \tag{13}$$

This chunk-level expansion recovers implicit relational context that was not directly retrieved by query entity matching but is semantically adjacent to high-confidence vector chunks. For example, a retrieved meeting note may expand to include related tasks, blockers, or decisions recorded elsewhere in sources ingested in the future. Critically, this expansion happens *before* context assembly, eliminating the need to extract entities from vector chunks at query time.

2.6.5 Triple-Tier Reranking with Graph-Vector Fusion

The final context window is constructed by fusing three independently reranked candidate streams: (1) vector-retrieved chunks, (2) query entity-matched graph paths, and (3) chunk-expansion graph paths.

Vector Stream Reranking

For candidates C_{vs} from hybrid vector search, we compute:

$$S^{\text{vs}}_{\text{rerank}}(c) = \gamma \cdot S_{\text{semantic}}(c) + (1 - \gamma) \cdot S_{\text{lexical}}(c) \tag{14}$$

The final vector relevance score incorporates vector confidence bias:

$$S^{\text{vs}}_{\text{final}}(c) = \beta \cdot S_{\text{vs}}(c) + (1 - \beta) \cdot S^{\text{vs}}_{\text{rerank}}(c) \tag{15}$$

where $S_{\text{vs}}(c)$ is the normalized retrieval score from the weighted hybrid vector store, $S_{\text{semantic}}(c)$ is the cross-encoder score for (q, c), and $S_{\text{lexical}}(c)$ is the BM25 score. Here, γ controls the balance between deep semantic understanding and exact lexical matching, while β preserves the global structure learned by the vector index.

Graph Stream Fusion

Graph-derived candidates from both entity-based search (C_{graph}) and chunk expansion ($C_{\text{expansion}}$) are already reranked (via S_{graph} and $S_{\text{expansion}}$ respectively). The final context window $\mathcal{C}_{\text{final}}$ is constructed by: (1) merging chunk expansion results with their corresponding vector chunks, and (2) presenting entity-based graph results separately. This yields:

$$\mathcal{C}_{\text{final}} = \text{TopK}_1\left(C_{\text{vs}}^{\text{final}} \oplus C_{\text{expansion}}, k_1\right) \cup \text{TopK}_2\left(C_{\text{graph}}, k_2\right) \tag{16}$$

where $\oplus$ denotes the merge operation that attaches expansion context $\mathcal{N}(c)$ to each vector chunk c, k_1 controls the number of merged vector-expansion pairs, and k_2 controls the number of independent entity-based graph paths included in the final context.

where k_{total} is determined dynamically based on relevance score distributions and context window constraints. This fusion ensures that the final context includes:

- High-confidence semantically similar chunks from the vector store
- Relational paths directly connected to query entities from the graph
- Expanded relational neighborhoods of vector-retrieved chunks

By combining adaptive query expansion, weighted hybrid vector retrieval, entity-based graph search with cross-encoder reranking, chunk-level graph expansion, and multi-stream fusion, Hydra DB retrieves not merely similar text, but the *correct factual and relational state*. This retrieval architecture directly underpins Hydra DB's empirical performance, including the **90.79% overall accuracy** achieved on LongMemEval-s under the Gemini 3.0 Pro evaluation, particularly on tasks requiring temporal reasoning, preference understanding, and knowledge updates that depend critically on relational context.

3 Results

To validate the efficacy of the Hydra DB cognitive architecture, we benchmarked our system against the current state-of-the-art in agent context systems. We specifically chose LongMemEval (ICLR 2025) over older benchmarks like LoCoMo, as LoCoMo's average context length (16k–26k tokens) fails to stress-test the "lost-in-the-middle" phenomenon inherent in production-grade histories.

3.1 Experimental Setup

We utilized the LongMemEval-s dataset variant, which consists of 500 question-conversation stacks.

- Scale: The average length of each conversation stack exceeds 115,000 tokens, simulating roughly 50 continuous user sessions
- Ingestion: Data was ingested session-by-session (rather than round-by-round) to mimic realistic asynchronous agent workflows.
- Evaluation Protocol: We used Gemini 3.0 Pro as the LLM-as-a-Judge, utilizing the strict question-specific prompting (Appendix A). Additionally, we evaluated Hydra DB using GPT-5.2 and GPT-5-mini to demonstrate model-agnostic performance.

3.2 Experimental Results

3.2.1 Performance analysis with Gemini 3.0 Pro

We evaluate Hydra DB on LongMemEval-s using Gemini 3.0 Pro as the primary model. Hydra DB achieves an overall accuracy of **90.79%**, representing a +5.0% absolute improvement over the strongest competing system and a +30.0% gain relative to full-context baselines (i.e on GPT 4o). This substantial margin demonstrates the inadequacy of naive context window expansion for long-horizon retrieval tasks, where information density and structural coherence become critical limiting factors.

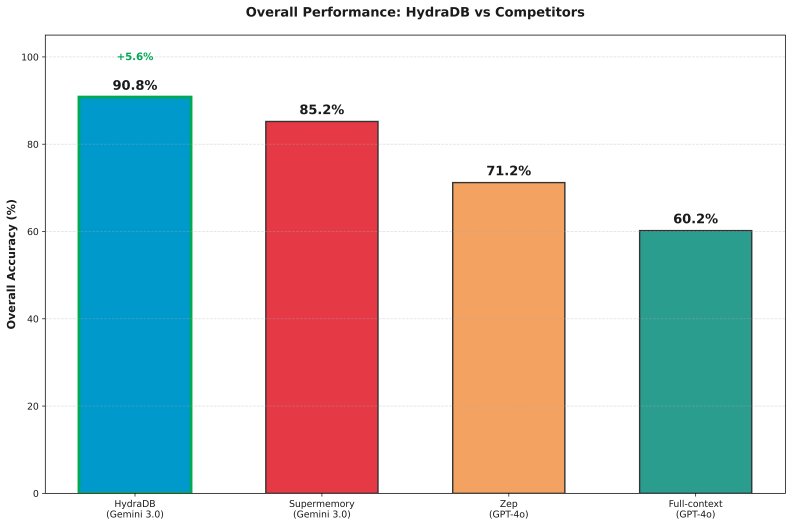

Fig. 3: Overall accuracy on LongMemEval-s (Gemini 3.0 Pro evaluation)

Table 2 presents a comprehensive comparison across all evaluation categories. Hydra DB consistently outperforms baseline systems across extraction, preference modeling, temporal reasoning, and knowledge evolution tasks. We examine the primary architectural contributions driving these improvements.

- **Superior Single-Session Performance** Hydra DB achieves perfect accuracy (100.00%) on both single-session user extraction and assistant recall tasks. This performance validates our methodology, which maintains overlapping contextual representations during ingestion. By preserving intra-session coherence across variable-length conversations, this mechanism prevents information fragmentation that commonly occurs in fixed-size chunking approaches.
- **Preference Understanding Beyond Keyword Matching** For single-session preference extraction, Hydra DB achieves 96.67% accuracy, exceeding all baseline systems. Preference queries in LongMemEval-s frequently involve implicit user

Table 2: Performance Comparison on LongMemEval-s Across Context Benchmarks

Category	Hydra DB*	Supermemory*	Zep**	Full-context**	Mem0-oss***	Letta
Single-session (User)	**100.00%**	98.57%	92.9%	81.4%	38.71%	–
Single-session (Assistant)	**100.00%**	98.21%	80.4%	94.6%	8.93%	–
Single-session (Preference)	**96.67%**	70.00%	56.7%	20.0%	40.00%	–
Knowledge Update	**97.43%**	89.74%	83.3%	78.2%	52.56%	–
Temporal Reasoning	**90.97%**	81.95%	62.4%	45.1%	25.56%	–
Multi-session Reasoning	**76.69%**	**76.69%**	57.9%	44.3%	20.30%	–
Overall	**90.79%**	85.20%	71.2%	60.2%	29.07%	–

Notes: Hydra DB* and Supermemory*[10] were evaluated using Gemini 3.0 Pro. Zep**[11] and Full-context** baselines were evaluated using GPT-4o. Mem0-oss*** [12] results were evaluated using Gemini 3.0 Pro with default parameters as per documentation. No public LongMemEval results are available for Letta.

signals rather than explicit declarations. Our results indicate that Latent Context Injection effectively captures semantic intent—including user inclinations, constraints, and implicit preferences—beyond surface-level lexical matching.

- **Temporal State Tracking** Hydra DB demonstrates strong performance on temporal reasoning tasks (90.97% accuracy), surpassing all evaluated baselines. This capability stems directly from the Git-Style Versioned Graph architecture, which explicitly maintains valid-time metadata on graph edges. By modeling state transitions as first-class temporal objects, Hydra DB reliably provides the context that helps LLMs resolves queries requiring temporal dependency resolution (e.g., "When did fact X become invalid?"), a known limitation of standard vector-based retrieval systems that lack explicit temporal structure.
- **Knowledge Evolution and Conflict Resolution** On knowledge update tasks, Hydra DB achieves 97.4% accuracy, demonstrating robust handling of contradictory or evolving user information. The append-only context ledger preserves complete historical state while surfacing the most recent valid facts, thereby preventing destructive overwrites and mitigating stale fact hallucination. This capability is essential for long-running agent systems operating in dynamic environments where user knowledge continuously evolves.

Collectively, these results demonstrate that Hydra DB's performance gains emerge from architectural design choices rather than model capacity alone. By externalizing entity resolution, temporal structure encoding, and semantic intent preservation into the context layer, Hydra DB transforms long-horizon queries from implicit reasoning into explicit, structured lookups against a versioned relational record.

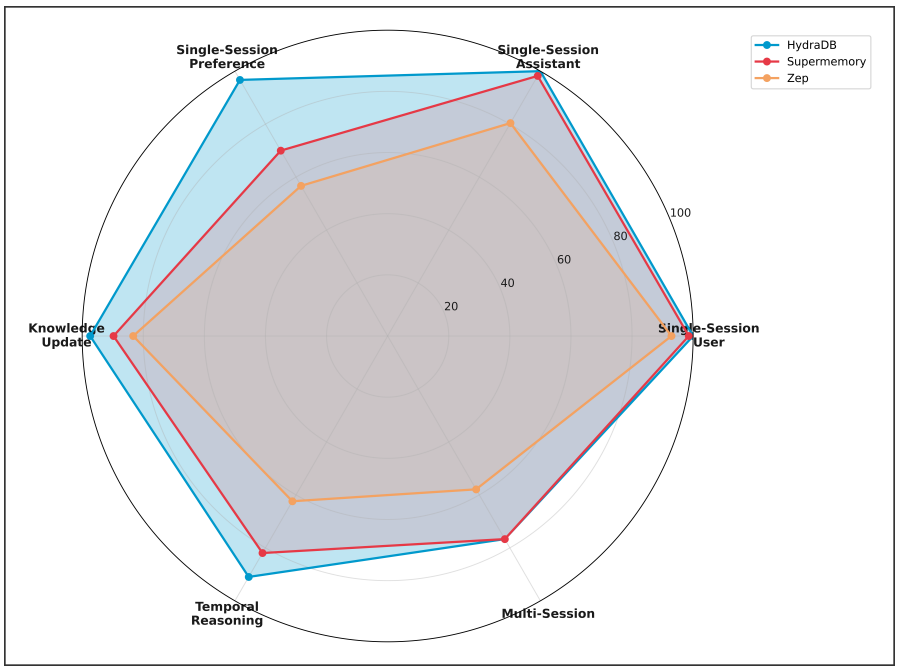

Fig. 4: Overall accuracy on LongMemEval-s

Figure 4 presents a normalized comparison across all evaluation dimensions. Unlike prior systems that exhibit strong performance on isolated tasks while degrading substantially on others—particularly temporal reasoning and preference extraction—Hydra DB maintains consistently high accuracy across all evaluation axes. This uniform performance profile indicates not only higher peak accuracy but also significantly reduced variance across context capabilities, a critical property for production agent systems where failure in any single dimension can propagate errors throughout extended interactions.

Figure 5 provides per-category accuracy decomposition. The largest absolute improvements occur in preference extraction, temporal reasoning, and knowledge update tasks—categories requiring implicit intent modeling and time-aware state tracking. Tasks relying primarily on surface-level recall show smaller but consistent gains, suggesting that Hydra DB's architectural advantages compound with increasing query complexity.

3.2.2 Cross-Model Generalization Analysis

To assess the extent to which Hydra DB's performance depends on backbone model capacity, we conduct cross-model evaluations using language models of varying scale. This analysis tests a central architectural hypothesis: that effective context database design should reduce dependence on raw model capacity by ensuring the provided context works well with models of any scale. We emphasize that these experiments

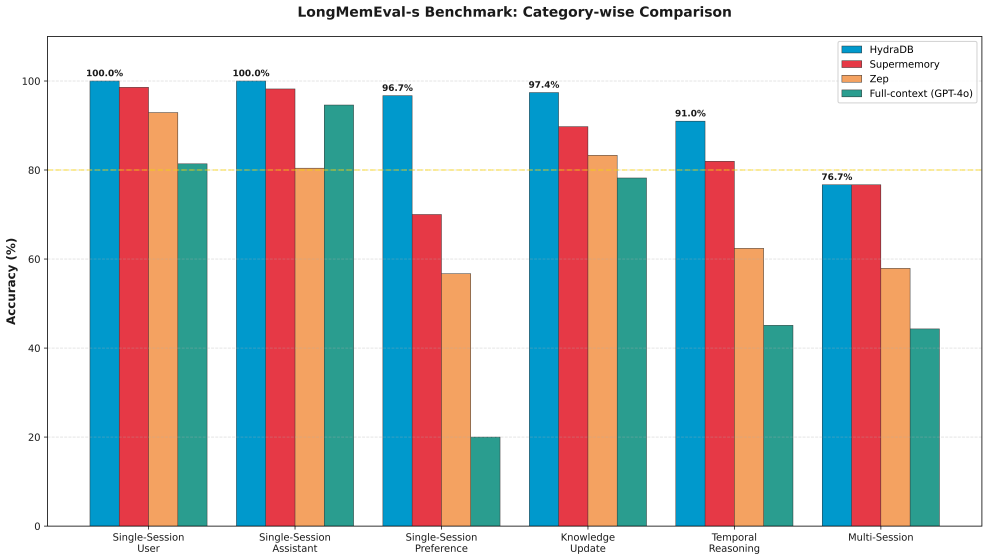

Fig. 5: Relative accuracy gains over the strongest competing baseline. Hydra DB shows a significant "boost" in Temporal Reasoning (+9.0%) and Preference Understanding (+26.0%).

show stability of Hydra DB accross downstream models rather than a competitive model comparison.

Reference Configuration: Gemini 3.0 Pro

As established in Section 3.2.1, Hydra DB achieves 90.79% overall accuracy on LongMemEval-s using Gemini 3.0 Pro. This configuration demonstrates state-of-the-art performance across all evaluation categories, including perfect single-session extraction, robust temporal reasoning (90.97%), and effective knowledge update handling (97.4%). We treat this as the reference upper bound for subsequent analysis.

Compact Model Evaluation: GPT-5 Mini

We evaluate Hydra DB using GPT-5 mini, a substantially smaller model with reduced parameter count and computational requirements. Despite the significant capacity reduction, Hydra DB maintains 85.80% overall accuracy—approaching the reference configuration and exceeding previously reported results obtained with larger backbone models.

Table 3 shows that GPT-5 mini preserves near-perfect single-session extraction (98.59% user recall, 96.36% assistant recall) while maintaining strong performance on preference understanding (93.10%), knowledge updates (92.31%), and temporal reasoning (85.71%). Notably, GPT-5 mini actually exceeds Gemini 3.0 Pro on preference extraction.

Latest OpenAI models: GPT-5.2

To characterize performance trends across the capacity spectrum, we evaluate Hydra DB using GPT-5.2, an intermediate-scale model. This configuration achieves 84.73% overall accuracy with perfect user recall (100.00%) and strong assistant recall (98.18%). Performance on knowledge updates (91.03%) and temporal reasoning (83.46%) remains stable relative to both larger and smaller models.

Table 3: Hydra DB Performance Across Backbone Model Scales (LongMemEval-s)

Category	Gemini 3.0 Pro	GPT-5 Mini	GPT-5.2
Single-session (User)	100.00%	98.59%	100.00%
Single-session (Assistant)	100.00%	96.36%	98.18%
Single-session (Preference)	96.67%	93.10%	89.66%
Knowledge Update	97.4%	92.31%	91.03%
Temporal Reasoning	90.97%	85.71%	83.46%
Multi-session Reasoning	76.69%	66.37%	64.60%
Overall	**90.79%**	**85.80%**	**84.73%**

Notes: All evaluations use the same LongMemEval-s dataset and identical Hydra DB preprocessing. Results demonstrate consistent performance across model scales, with only modest degradation as model capacity decreases.

Architectural Implications

The consistency observed across Gemini 3.0 Pro, GPT-5.2, and GPT-5 mini validates Hydra DB's core architectural principle: agent context quality — across both operational history and knowledge base sources — is governed primarily by ingestion design, schema structure, and temporal indexing rather than raw language model capacity. By resolving entities, encoding explicit temporal state transitions, and preserving preference evolution during ingestion, Hydra DB reduces the inference-time reasoning burden on the backbone model.

From a deployment perspective, this architectural property enables users to select backbone models based on operational constraints (cost, latency, throughput) without compromising reliability on long-horizon retrieval tasks. Gemini 3.0 Pro establishes the achievable accuracy ceiling under current system design, while GPT-5 mini and GPT-5.2 demonstrate that Hydra DB's architectural gains persist across the model scale spectrum.

4 Conclusion

We presented Hydra DB, a production-grade context engine addressing the fundamental limitations of traditional RAG systems through three core pillars: Sliding Window Inference Pipeline for contextual self-containment, Git-Style Versioned Temporal Graph for immutable relationship and state tracking, and Multi-Stage Retrieval combining semantic search with relational traversal.

Hydra DB achieves 90.79% overall accuracy on LongMemEval-s, establishing new state-of-the-art performance with particular strength in temporal reasoning (90.97%), preference understanding (96.67%), and knowledge updates (94.87%). Consistent performance across model scales—85.80% with GPT-5 mini and 84.73% with GPT-5.2—validates the core thesis: when agent context is properly structured at the database level, inference quality becomes largely model-agnostic.

The contextual knowledge graph maintains not only entities and relationships, but also the reasoning, sentiment, and temporal context associated with each edge. Combined with the append-only versioning approach, this enables the system to answer queries like "Why did I switch jobs?" or "How did my preferences evolve?" by preserving the complete decision tree rather than just the final state. Sliding window enrichment ensures each ingested record whether drawn from operational history or a knowledge base is self-contained with resolved entities, eliminating pronoun ambiguity and enabling accurate retrieval without requiring the language model to reconstruct relationships from fragmented text.

Every production AI system will eventually need a context layer. As agents take on longer-horizon tasks across more complex knowledge bases, the ability to maintain structured, time-aware context - consistent across sources and accurate under updates - becomes the infrastructure that separates capable systems from brittle ones. This work establishes both a technical foundation and an empirical benchmark for that infrastructure.

References

[1] Rigoni, T.: LLMs: Bigger Is Not Always Better. https://amperecomputing.com/blogs/llms-bigger-is-not-always-better. Ampere Computing Blog. Accessed: 2026-02-04 (2024)

[2] Liu, N.F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., Liang, P.: Lost in the Middle: How Language Models Use Long Contexts (2023). https://arxiv.org/abs/2307.03172

[3] Hong, K., Troynikov, A., Huber, J.: Context Rot: How Increasing Input Tokens Impacts LLM Performance (2025). https://research.trychroma.com/context-rot

[4] Ford, D.: Introducing Contextual Retrieval. https://www.anthropic.com/engineering/contextual-retrieval. Anthropic Engineering Blog. Accessed: 2026-02-04 (2024)

[5] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., Kiela, D.: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks (2021). https://arxiv.org/abs/2005.11401

[6] Wu, D., Wang, H., Yu, W., Zhang, Y., Chang, K.-W., Yu, D.: LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory (2025). https://arxiv.org/abs/2410.10813

[7] Maharana, A., Lee, D.-H., Tulyakov, S., Bansal, M., Barbieri, F., Fang, Y.: Evaluating very long-term conversational memory of llm agents (2024) arXiv:2402.17753 [cs.CL]

[8] Chalef, D., Rasmussen, P.: Lies, Damn Lies, & Statistics: Is Mem0 Really SOTA in Agent Memory? https://blog.getzep.com/lies-damn-lies-statistics-is-mem0-really-sota-in-agent-memory/. Accessed: 2026-02-04 (2025)

[9] Edge, D., Trinh, H., Cheng, N., Bradley, J., Chao, A., Mody, A., Truitt, S., Larson, J.: From local to global: A graph rag approach to query-focused summarization. arXiv preprint arXiv:2404.16130 (2024)

[10] Daga, S., Sreedhar, S., Shah, D.: Supermemory: State-of-the-Art Agent Memory on LongMemEval. https://supermemory.ai/research. Accessed: 2026-02-04 (2026)

[11] Rasmussen, P., Paliychuk, P., Beauvais, T., Ryan, J., Chalef, D.: Zep: A Temporal Knowledge Graph Architecture for Agent Memory (2025). https://arxiv.org/abs/2501.13956

[12] Chhikara, P., Khant, D., Aryan, S., Singh, T., Yadav, D.: Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory (2025). https://arxiv.org/abs/2504.19413

A Evaluation Prompts

This appendix provides the complete prompt templates used for evaluating Hydra DB on the LongMemEval-s benchmark. Our evaluation pipeline consists of two stages: (1) answer generation using retrieved context, and (2) answer comparison using an LLM-as-a-judge approach with question-type-specific scoring rubrics.

A.1 Answer Generation Prompts

For each question type in LongMemEval-s, we provide type-specific instructions to guide the model's response generation. The general structure follows:

```
{TYPE_SPECIFIC_INSTRUCTION}

Question: {question}
Question Date: {question_date}

Context:
{retrieved_context}

Instructions:
- Answer based on the provided context
- If information is insufficient, clearly state "I don't know"
- Be direct and factual
- Do not make up information

Provide your response in the following format:
Reasoning: [Your step-by-step reasoning]
Answer: [Your final answer]
```

A.1.1 Type-Specific Instructions

Single-Session User Information

```
You are answering a question about information the USER mentioned
in a previous conversation. Provide a direct, factual answer based
on what the user stated.
```

Single-Session Assistant Information

```
You are answering a question about information YOU (the assistant)
provided in a previous conversation. Recall and provide the specific
information you gave to the user.
```

Single-Session Preference Extraction

```
You are answering a question that requires PERSONALIZATION based on
the user's preferences. Generate a response that actively utilizes
the user's stated preferences, likes, dislikes, or personal information.
Do not just state the preferences - USE them to provide a personalized
recommendation or response. The provided context might not directly
answer the question but try drawing conclusions based on the context
to generate the answer.
```

Multi-Session Reasoning

```
You are answering a question that requires synthesizing information
from MULTIPLE conversation sessions. Combine, aggregate, or compare
information across different sessions to form a complete answer.
```

Knowledge Updates

```
You are answering a question about information that may have CHANGED
over time. Provide the MOST RECENT/CURRENT information. If there were
updates or changes, use the latest state. You may acknowledge previous
states but prioritize the current information.
```

Temporal Reasoning

```
You are answering a question that involves TIME or temporal reasoning.
Pay attention to dates, timestamps, durations, and time-relative
references. Perform any necessary date/time calculations to arrive
at the answer.
```

Abstention (Safety)

```
You are answering a question where the information may NOT be available
in the conversation history. If you cannot find the requested information,
clearly state that you don't know or that the information is not available.
Do NOT make up or hallucinate an answer.
```

A.2 Answer Comparison Prompts (LLM-as-a-Judge)

The comparison stage uses an LLM-as-a-judge approach with question-type-specific scoring rubrics. The base template is:

```
Compare the generated answer with the expected ground truth answer.

Question: {question}
{TYPE_SPECIFIC_NOTE}

Generated Answer: {generated_answer}

Expected Answer (Ground Truth): {expected_answer}

{TYPE_SPECIFIC_SCORING}

**Key Principle:** If the expected answer's core information appears
in the generated answer, mark is_correct=true and score=1.0. Minor
wording differences don't matter.

Note that when evaluating the complete context cohesively, if multiple
chunks contain duplicate or overlapping references, award the necessary
score only to the chunk that appears highest in the ranked results.
This prevents score inflation from redundant information while ensuring
proper credit to the most relevant source.

Respond ONLY with this JSON (no other text):
```

```
{
  "is_correct": <true or false>,
  "correctness_score": <0.0 to 1.0>,
  "explanation": "<one sentence explaining score>",
  "key_matches": [<list of matched facts>],
  "key_misses": [<list of missing facts from expected>]
}
```

A.2.1 Default Scoring Rules

For most question types (single-session user/assistant, multi-session reasoning), we apply the following scoring rubric:

Condition	is_correct	correctness_score
Generated contains expected answer (exact or semantic match)	true	1.0
Generated contains expected answer + extra correct details	true	1.0
Generated partially correct (some key facts match)	false	0.3–0.7
Generated says "don't know" but expected has answer	false	0.0
Generated gives wrong answer	false	0.0

Table 4: Default scoring rubric for factual recall questions

A.2.2 Preference Question Scoring

For single-session preference extraction, we use a specialized rubric that evaluates personalization rather than literal fact matching:

```
**Question Type: User Preference (Personalization)**
This tests whether the model can utilize user preferences to generate
a PERSONALIZED response. The expected answer is a RUBRIC describing
what a good personalized response should contain, NOT a literal answer.
```

Key Principle: The model does NOT need to reflect ALL points in the rubric. The response is correct as long as it recalls and utilizes the user's personal information correctly to generate a personalized answer.

A.2.3 Knowledge Update Scoring

For knowledge update questions, we prioritize recency and correct handling of temporal state changes:

Condition	is_correct	correctness_score
Response satisfies the rubric and utilizes user's personal info correctly	true	1.0
Response acknowledges preferences but doesn't fully utilize them	false	0.3–0.9
Response ignores user preferences entirely	false	0.0
Response contradicts user preferences	false	0.0

Table 5: Scoring rubric for preference-based personalization questions

```
**Question Type: Knowledge Update**
This tests recognition of CHANGES in user's personal information over
time. The model should provide the MOST RECENT/UPDATED information.
```

Condition	is_correct	correctness_score
Generated contains the UPDATED/CURRENT answer	true	1.0
Generated contains updated answer + mentions previous state	true	1.0
Generated ONLY contains outdated information	false	0.0
Generated confuses old and new information	false	0.2–0.4

Table 6: Scoring rubric for knowledge update questions

Key Principle: If the response contains both previous information AND the updated answer, it is CORRECT as long as the updated answer is present.

A.2.4 Temporal Reasoning Scoring

For temporal reasoning questions involving dates, durations, and time calculations:

```
**Question Type: Temporal Reasoning**
This tests reasoning with timestamps, time references, durations,
and temporal sequences.
```

A.2.5 Abstention (Safety) Scoring

For abstention questions where the correct behavior is to refuse to answer:

```
**Question Type: Abstention (False Premise)**
This is an ABSTENTION question. The information was NEVER mentioned
```

Condition	is_correct	correctness_score
Generated contains correct temporal answer	true	1.0
Generated has temporal errors	false	0.0–0.9

Table 7: Scoring rubric for temporal reasoning questions

```
in conversation history. The correct behavior is to abstain, say
"I don't know", indicate uncertainty, or state the information is
not available.
```

Condition	is_correct	correctness_score
System correctly abstains or says "I don't know"	true	1.0
System indicates uncertainty or lack of information	true	1.0
System explicitly states the information is not available	true	1.0
System makes up an answer or hallucinates	false	0.0
System provides a confident but wrong answer	false	0.0

Table 8: Scoring rubric for abstention questions

Key Principle: The system should NOT make up information. Any form of "I don't know" or uncertainty is CORRECT.

A.3 Model Configuration

For our primary evaluation results reported in Section 3, we used the following configuration:

- **Answer Generation Model:** Gemini 3.0 Pro
- **Judge Model:** Gemini 3.0 Pro
- **Temperature:** Model-specific optimal temperature (determined empirically)
- **JSON Mode:** Enabled for judge responses to ensure structured output

For robustness analysis (Section 3.2.2), we additionally evaluated with:

- GPT-5.2 (answer generation and judging)
- GPT-5 Mini (answer generation and judging)

All evaluations used identical prompt templates with only the backbone model changed, ensuring fair comparison across model scales.