TelloDB: A coverage-aware multi-index memory engine for data-local AI agents

Sharjeel Abbas [1]
[1]Independent Researcher

Tehreem Arooj, MPhil (Artificial Intelligence) [1]
[1]Independent Researcher

Abstract

Agent memory must preserve changing facts, retrieve evidence across sessions, and operate within the data boundary of its deployment. Systems that combine dense retrieval, full-text search, temporal state, and graph relations often distribute those functions across several services. This paper investigates whether the same logical functions can be collocated in one application process without discarding temporal history or source provenance.

We present TelloDB, a Rust prototype built from per-tenant SQLite databases, four FTS5 indexes, a persistent HNSW index, and local ONNX inference. Raw observations remain in the memory store; derived facts carry validity intervals and source memory or session identifiers. An intent-aware query planner combines semantic, lexical, session, temporal, profile, and graph candidates, then hydrates the selected evidence.

We evaluate retrieval feasibility using a recorded LoCoMo run covering 1,538 eligible questions. The system retrieves at least one annotated source session in the top eight for 95.8% of questions. Category results range from 80.4% for multi-hop questions to 98.1% for open-domain questions. Mean observed query time is 94 ms in the recorded run. The metric does not require complete evidence retrieval or a correct generated answer, and the historical log lacks hardware metadata and matched baselines. The result therefore supports the feasibility of the collocated design, not a claim of cross-system superiority. An implementation audit identifies the correctness, isolation, and reproducibility work required before enterprise or edge-device deployment.

Keywords: agent memory, temporal database, embedded systems, hybrid retrieval, provenance, local inference, privacy

1 Introduction

Long-context language models do not remove the need for durable memory. A context window disappears when a session ends, and a longer prompt can still mix current facts with statements that have become stale. Models also use evidence unevenly across long prompts [7]. Retrieval-augmented generation reduces the amount of text presented to a model [6], but a flat retrieval index does not represent fact supersession, temporal validity, typed relationships, or source provenance.

Agent-memory systems have responded by combining several representations. HydraDB, the main conceptual influence for this work, joins temporal graph state with vector, lexical, and reranking stages [16]. Zep/Graphiti, Mem0, MemGPT, and Generative Agents make related choices about temporal knowledge, consolidation, or external memory [1, 13–15]. These designs show why memory is more than nearest neighbour search. They leave a separate systems question: how much of this memory substrate can run inside one locally administered process?

That question matters when conversations, source code, clinical notes, or internal documents cannot be sent to a hosted embedding or memory provider. It also matters on personal computers and intermittently connected devices, where several database and inference services are difficult to operate. Collocation may reduce deployment and network costs, but it moves durability, isolation, and resource contention into one failure domain.

TelloDB is a Rust prototype for studying that trade-off. Its main query path uses per-tenant SQLite files, SQLite FTS5, a process-wide HNSW index, and local ONNX models. It retains raw memory records, creates new versions instead of overwriting old facts, and can return source memory and session identifiers with retrieved context. “Local” has a narrow meaning in this paper: after model assets are provisioned, the core engine does not send memory text to an external embedding service. It does not imply encrypted storage, automatic regulatory compliance, or a hardened network perimeter.

1.1 Research questions

The study addresses three questions:

RQ1
How can temporal, semantic, lexical, and relational representations share one process while preserving source evidence and fact history?

RQ2
What retrieval behaviour does the resulting prototype exhibit on long multi-session conversations, and which question classes remain difficult?

RQ3
Which correctness, tenant-isolation, and reproducibility risks must be resolved before private enterprise or edge-device deployment can be claimed?

1.2 Contributions and claim boundary

This paper makes four contributions:

1. It specifies a collocated memory architecture that separates raw observations, fact versions, lexical indexes, graph relations, and dense vectors while retaining source memory and session identifiers.
2. It describes a version-preserving temporal model and a coverage-aware retrieval pipeline that combines session routing with semantic, lexical, structured, and relational candidates.
3. It reports a source-grounded feasibility evaluation on LoCoMo, including category counts, stage timings, metric semantics, and provenance limitations.
4. It derives concrete failure conditions from the implementation, including non-atomic secondary indexing, process-wide vector-label collisions, an unscoped MCP path, and incomplete deletion and temporal endpoints.

The contribution is architectural and empirical, not a new retrieval algorithm. The recorded benchmark has no matched baseline, answer-generation score, hardware manifest, or repeated trials. We therefore do not compare its 95.8% session-hit result numerically with published answer-accuracy results. The paper treats Raspberry Pi and enterprise operation as evaluation targets, not demonstrated outcomes.

2 Background and related work

2.1 Memory beyond a context window

MemGPT treats the model context as a scarce memory tier and moves information between working context and external storage through explicit calls [13]. Generative Agents stores observations, reflections, and plans, then ranks them using relevance, recency, and importance [14]. CoALA organises agent memory into working, episodic, semantic, and procedural forms [21]. Together, these systems establish that an agent needs memory policy and state management, not only a larger prompt.

RAG provides a non-parametric evidence store for generation [6]. Dense Passage Retrieval separates query and passage encoding [5], and cross-encoder reranking improves the second stage by scoring query-passage pairs jointly [12]. TelloDB uses this retrieve-then-rerank pattern, but neural relevance is one input to a database query plan rather than the sole memory representation.

2.2 Temporal and graph memory

Temporal memory must distinguish a new observation from a correction to an old fact. Zep/Graphiti represents changing conversation and business data in a temporal knowledge graph [15]. Mem0 extracts and consolidates long-term memories [1]. GraphRAG uses entity relations and community summaries to support questions that are poorly served by flat chunk retrieval [4].

HydraDB is the closest conceptual predecessor. Its public report combines a temporal graph, sliding-window chunk enrichment, multiple vector representations, lexical scoring, graph expansion, and cross-encoder reranking [16]. The report does not disclose enough deployment detail to determine how many independently operated services are required. Our comparison therefore concerns the logical design, not an unsupported claim about HydraDB's physical deployment.

Table 1: Reported emphasis of representative agent-memory systems. The table summarises the cited publications and is not a feature-completeness ranking.

System	Main memory abstraction	Retrieval or update emphasis	Deployment emphasis in the cited work
MemGPT	Hierarchical context	Movement between context and external memory	Memory management policy
Generative Agents	Observation stream and reflection	Relevance, recency, and importance	Simulated-agent behaviour
Zep/Graphiti	Temporal knowledge graph	Evolving entities and relations	Agent-memory service architecture
Mem0	Extracted long-term memories	Extraction and consolidation	General agent-memory layer
HydraDB	Temporal relational state	Vector, lexical, graph, and reranking streams	Production-oriented memory substrate
TelloDB	Raw memories and versioned facts	Collocated FTS5, HNSW, graph, temporal, and session paths	One locally administered process

2.3 Retrieval and embedded execution

HNSW organises vectors into a hierarchy of proximity graphs for approximate nearest-neighbour search [9]. Dense similarity does not replace exact lexical matching for names, dates, identifiers, and quoted phrases. BM25 remains a standard probabilistic ranking method [17], and SQLite FTS5 implements it inside the embedded database [18]. Reciprocal rank fusion combines ranked lists without assuming that their scores share a scale [2].

SQLite is both self-contained and serverless: application code reads and writes the database without a separate database daemon [19, 20]. This property makes it useful for studying collocation. It does not make an application single-file or failure-free. Model weights, vector checkpoints, database files, and native inference libraries remain part of the deployed artifact.

2.4 Vector systems and recovery

Vector database research addresses query processing, updates, and scale beyond standalone nearest-neighbour libraries. Milvus separates storage and execution within a distributed vector data-management system [23]. VBASE integrates approximate vector search with relational operators while preserving top-k query semantics [26]. TelloDB studies a smaller embedded setting, but faces the same basic requirement that vector identifiers and relational rows remain in agreement.

Database recovery work such as ARIES uses write-ahead logging, redo, and undo to recover a transactional state after failure [10]. TelloDB relies on SQLite's WAL for relational state, but its external HNSW checkpoint is outside that recovery protocol. This difference motivates the index-convergence invariant in Section 5.

2.5 Benchmarks and reporting

LoCoMo contains long, multi-session conversations grounded in personas and temporal event graphs. Its conversations average roughly 600 turns and 16,000 tokens, and its tasks include question answering, event summarisation, and multimodal dialogue generation [8]. LongMemEval tests information extraction, multi-session reasoning, temporal reasoning, knowledge updates, and abstention across 500 questions [25].

Both benchmarks ultimately concern answer quality. Retrieving one annotated session is an intermediate measurement. It does not show that every required session was found or that an answer model used the evidence correctly. This paper reports session retrieval because that is what the supplied evaluator records. It keeps that result separate from published end-to-end answer scores.

2.6 Privacy and edge inference

On-device inference can reduce dependence on network services and keep raw inputs near their source, but memory limits, model size, heat, and device-specific acceleration constrain practical deployments [24, 27]. NIST

frames privacy as risk management rather than a property supplied by one component [11]. Local embedding removes one disclosure path: memory text need not be sent to a hosted embedding API. Disk encryption, transport security, access control, deletion, logging, and backup policy remain separate requirements.

3 Problem formulation and design criteria

3.1 System model

We model an observation as

$$m = \langle i, u, e, s, x, k, t_d, t_i, \ell \rangle, \tag{1}$$

where i is a memory identifier, u a tenant, e an entity, s a session, x the source text, k the memory kind, t_d document time, t_i ingest time, and ℓ lifecycle metadata. Every derived object should retain enough of (i, s) to identify its source memory or session.

A fact version is a tuple $r = \langle f, v, t_{\text{from}}, t_{\text{to}}, p, \sigma \rangle$, where f is a fact key, v its value, p its provenance, and σ its status. Inserting a replacement closes the previous validity interval and adds a `supersedes` relation instead of deleting the earlier record. For a query time t_q,

$$\text{current}(f, t_q) = \arg\max_{r \in V_f} \{ r.t_{\text{from}} \mid r.t_{\text{from}} \leq t_q < r.t_{\text{to}} \}. \tag{2}$$

An open upper bound denotes the current version. This model borrows the valid-time distinction from temporal databases, although the current implementation is not a complete bitemporal database [3].

3.2 Design criteria

The prototype is evaluated against five criteria:

D1: Local execution
Once public model assets are present, ingestion, embedding, indexing, and retrieval do not require a hosted memory, embedding, graph, or vector service.

D2: Evidence preservation
A derived fact or candidate retains source memory or session identifiers.

D3: Temporal update
Replacing a fact changes its current status without erasing the earlier claim or its validity interval.

D4: Heterogeneous retrieval
One query plan can combine paraphrase similarity, exact terms, sessions, time constraints, and typed relations.

D5: Bounded operation
The principal data and inference paths share one application process and an inspectable data directory.

These criteria create tensions rather than independent feature requests. Derived cards can improve retrieval but lose source wording if provenance is broken. A process-wide HNSW index avoids a vector service but must share durability and tenant-isolation rules with per-tenant SQLite files. Lifecycle expiration bounds growth but can remove evidence needed by a later query. A research-grade implementation must state these failure boundaries, not only the successful data path.

3.3 Trust and privacy boundary

The trusted boundary contains the TelloDB process, model cache, configuration, and data directory. The steady-state core path embeds and ranks memory text locally. A client may send retrieved context to an external answer model, but that action is outside the privacy claim. First-use model acquisition is also outside the steady-state boundary because FastEmbed may download public model files.

The prototype assumes a trusted host operating system and administrator. Tenant databases and vector checkpoints are not encrypted at rest. The server has API-key and bearer-token mechanisms, but it requires TLS termination, secret management, network restriction, and endpoint-level review for deployment. Separate SQLite files provide a storage boundary between tenants; the HNSW substrate remains shared and has a label-collision defect described in Section 8.

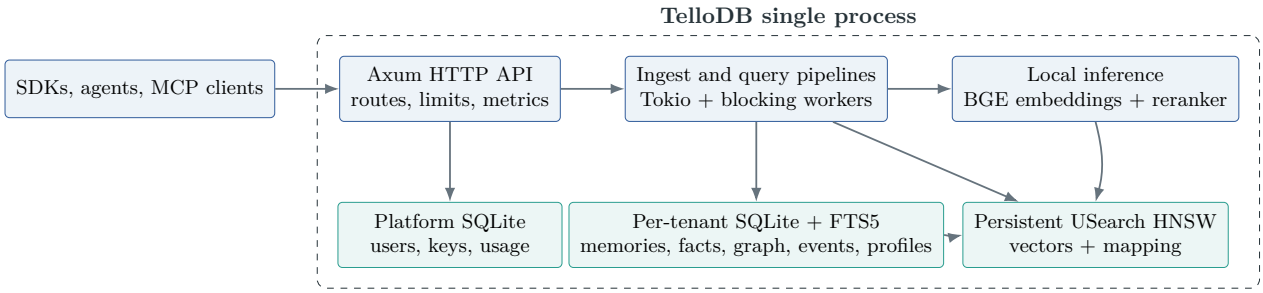

Figure 1: The TelloDB process. SQLite holds durable tenant state and four FTS5 indexes; HNSW supplies dense candidate generation. All primary query paths run in one application process. Model weights and database files remain external assets on disk.

3.4 Evidence used in this study

The study uses three forms of evidence. First, source inspection maps design claims to the executed Rust implementation rather than to README statements. Second, unit tests exercise parsing, authentication, fact supersession, point-in-time selection, query planning, storage, and vector persistence. Third, a historical LoCoMo log provides per-category retrieval counts and aggregate stage timings.

These sources have different force. Code inspection shows that a mechanism exists, but not that it behaves correctly under load or failure. Unit tests cover specified cases, but do not establish performance or tenant isolation. The LoCoMo log measures one retrieval configuration, but lacks the metadata needed for a controlled comparison. Section 6 defines the reported metric and the limits placed on its interpretation.

4 Architecture

4.1 One process, several memory models

Figure 1 shows the main deployment unit. An Axum HTTP server handles REST and MCP requests on a Tokio runtime. Shared engine state owns the inference layer, vector index, tenant database manager, platform store, ranking configuration, and rate limiter. Most REST handlers derive a tenant from an authenticated principal. The current MCP handler is an exception and is treated as a defect in Section 8.

Each tenant is opened lazily as a pooled SQLite database. The schema contains 32 ordinary or virtual tables in the audited source. Four are FTS5 indexes for raw memories, session routes, temporal events, and shadow questions. Other tables hold memories, memory cards, graph edges, metric records, fact versions, profiles, aliases, vector-to-memory mappings, lifecycle artifacts, and deletion tombstones. A ledger table is declared, but the audited ingest path does not insert turns into it. SQLite runs in write-ahead-log mode with foreign keys, a busy timeout, memory-backed temporary storage, and a connection pool.

The HNSW index is process-wide. Side maps associate vector identifiers with entities, while SQLite maps those identifiers back to durable memories. Checkpointing writes the index and its mapping to disk. This split keeps ANN search in a specialised structure without adding a vector-database service, but crash consistency spans two persistence mechanisms rather than one transaction. The implementation periodically checkpoints the vector index and SQLite WAL files.

4.2 Ingestion

The ingest request contains an entity identifier, memory identifier, timestamp, text, optional relations, optional fact fields, and controls for mining, deduplication, consolidation, and semantic indexing. Authentication selects the tenant and scopes identifiers to the principal.

The pipeline shown in Figure 2 chunks documents according to a content-type hint, adds nearby dialogue context, and builds companion representations. It computes embeddings in batches and uses a content hash for idempotence. A lifecycle evaluator scores salience, novelty, confidence, specificity, temporal content, utility, and stability. Low-value or sensitive records can be assigned different retention classes or excluded from the vector index.

The storage stage retains the raw observation before it writes derived artifacts. These include subject-predicate-

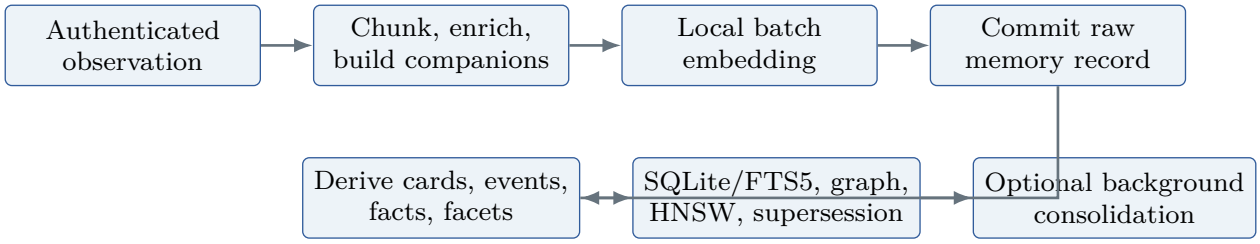

Figure 2: Ingestion keeps the source observation and writes derived representations in batches. Optional mining can be disabled for controlled benchmarks.

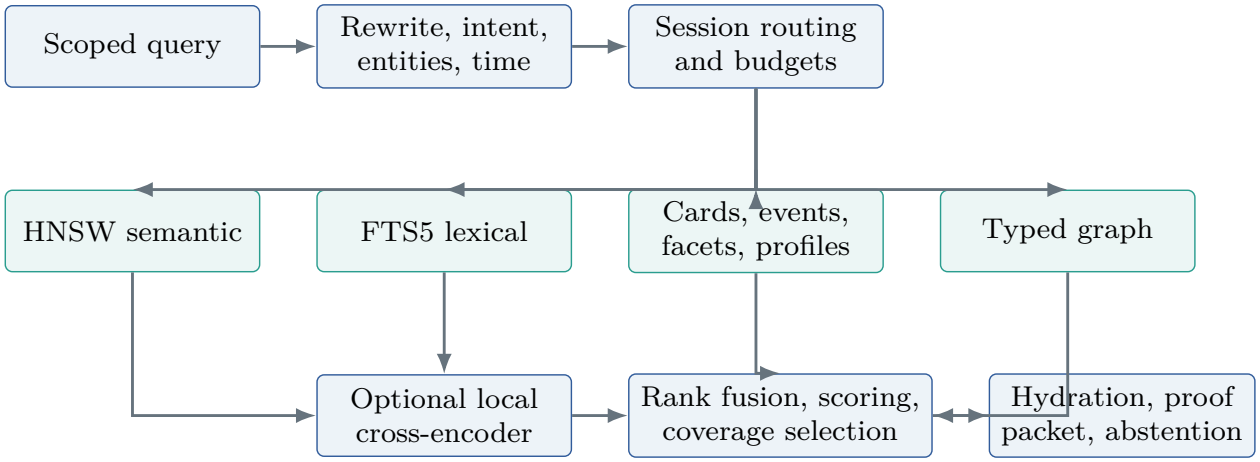

Figure 3: The query pipeline separates planning, candidate generation, reranking, fusion, and evidence hydration. Optional lanes are activated by the detected intent and available query features.

object cards, temporal events, shadow questions, facet postings, session-router records, profile facts, scenes, graph edges, and preferences. Fact registration marks an older version stale and links both directions of the supersession. The old source remains addressable. The schema and API also contain a numerical metric vault, but ordinary ingestion does not populate it in the audited revision. It is therefore excluded from the evaluated data path.

4.3 Consistency boundary

The source observation and all secondary representations do not share one atomic commit. SQLite transactions protect groups of relational writes, while HNSW updates and several derived-index operations happen across separate steps. Some secondary errors are logged or discarded after the primary record has been stored. A successful ingest response can therefore leave durable memory and secondary indexes out of agreement.

This boundary is central to the research problem. A complete database design would need an index journal or outbox, globally unique operation identifiers, and a replay watermark so that committed secondary work converges after a crash. The current prototype has checkpointing but no demonstrated exactly-once replay protocol. Section 8 treats recovery as an open correctness requirement.

4.4 Query planning and retrieval

The planner rewrites low-information request phrases, extracts named subjects, dates, lexical terms, ordinal constraints, and possible fact slots. It classifies general, numerical, temporal, recommendation, inference, and peripheral-mention intents with deterministic rules and an optional prototype-embedding classifier. Hard queries receive larger retrieval budgets and deterministic subqueries.

Session routing reduces the search space before chunk retrieval. It combines session-level FTS, entity pivots, time windows, query probes, and fallback routes using reciprocal-rank fusion. Candidate generation then searches HNSW, FTS5, and structured memory cards. Depending on the plan, the pipeline also searches temporal events, shadow questions, facets, scenes, profiles, and graph neighbours.

An optional BGE cross-encoder reranks a bounded seed set. Final scoring uses normalised retrieval signals and explicit adjustments:

$$S(m,q) = w_v S_v + w_l S_l + w_e S_e + w_t S_t + w_g S_g + B_{\text{card}} + B_{\text{session}} - P_{\text{stale}} - P_{\text{conflict}}. \tag{3}$$

The exact implementation has more lanes than this compact expression, including event, shadow-question, facet, profile, and evidence-density terms. Weights are loaded from a JSON file at startup or taken from defaults. Source comments associate some defaults with LoCoMo tuning. Because the recorded evaluation uses LoCoMo as well, the result must be treated as a development result until the weights are frozen and evaluated on an independent test set.

4.5 Coverage-aware session selection

Multi-session questions create a set-selection problem. Choosing the highest scoring sessions independently can return several near-duplicates while omitting a session that supplies a different part of the answer. The implementation therefore represents inferred query requirements and coverage facets as bit masks of at most 64 elements. Candidate cards are grouped by session, and each session records its covered requirements, facets, entity hits, temporal hits, and lexical support.

For coverage-style, decomposed, cross-entity, ordinal, or inference queries, the selector builds an ordering greedily. Given selected sessions A, it chooses the next session s using

$$J(s \mid A) = \alpha B(s) + \beta_r \Delta_r(s \mid A) + \beta_f \Delta_f(s \mid A) + \beta_e \Delta_e(s \mid A) + \beta_t \Delta_t(s \mid A) + L(s) - P(s \mid A), \tag{4}$$

where B is the base session score, Δ_r and Δ_f count newly covered requirements and facets, Δ_e and Δ_t reward new entity and temporal evidence, L is lexical support, and P penalises redundant or unsupported sessions. The selected masks are removed from the uncovered sets after each step. When coverage selection is inactive, sessions are ordered by base score. Candidate cards are then interleaved across the ordered sessions until the result budget is filled.

The current implementation orders all n candidate sessions, giving an $O(n^2)$ selection step after session features have been computed. Stopping after a session budget k would reduce that step to $O(kn)$. This selector is the most specific retrieval mechanism in the prototype and gives a falsifiable target for future work: compare complete-evidence recall and answer quality against score-only ordering under an equal context budget. The current any-session metric cannot establish that benefit.

4.6 Evidence, deletion, and maintenance

A query result can carry a proof packet with source memory and session identifiers, supporting card and event identifiers, entity and lexical hits, missing facets, and heuristic verification checks. A source-turn lookup is attempted, but normal ingestion does not populate the ledger table and proof construction can fall back to card text. The `strict` proof label does not select a distinct strict-verification branch in the audited revision. Proof packets are therefore provenance aids, not verified citations or formal proofs of answer correctness.

The intended deletion workflow removes a memory from indexes and the vector mapping, records a tombstone, and repairs affected fact slots where possible. The audited implementation contains a schema mismatch in this path, so successful deletion is not claimed. Background tasks checkpoint HNSW and tenant WAL files and expire records according to lifecycle policy. These maintenance paths have not yet been evaluated through crash injection.

5 Implementation

5.1 Runtime and storage

TelloDB is implemented in Rust 2021 with a declared minimum compiler version of 1.80. Axum supplies HTTP routing and Tokio schedules asynchronous work. CPU-heavy query execution runs on blocking workers; Rayon prepares ingestion batches in parallel. SQLite is compiled into the executable through `rusqlite`'s bundled feature. Each tenant has a separate database file, and a platform database stores identities, API keys, usage, and account metadata.

SQLite runs in write-ahead-log mode with foreign keys and a connection pool. FTS5 indexes stay beside the rows they reference. The USearch HNSW index and its vector-to-memory mapping are checkpointed separately. This arrangement removes network calls between retrieval components, but it does not give SQLite transactions authority over vector-index state.

5.2 Local inference and deployment unit

FastEmbed constructs BGE Small English embedding executors and BGE Reranker Base cross-encoders through ONNX Runtime. CPU is the default execution path. Supported builds can select CUDA or CoreML providers. The recorded LoCoMo run disabled cross-encoder reranking, so its result evaluates the first-stage retrieval configuration rather than the complete neural pipeline.

The deployment unit is one application executable, not one self-contained file. Model weights, tenant databases, configuration, and HNSW checkpoints remain external assets. ONNX Runtime or hardware-provider libraries may also be loaded dynamically. A reproducible offline release therefore needs preprovisioned model assets, hashes, and target-specific dependency manifests.

The current source hard-codes the BGE Small embedding model even though the README describes broader model selection. We report the executed model path rather than the documented alternatives.

5.3 Traceability from criteria to mechanisms

Table 2: Design criteria, implementation mechanisms, and present evidence.

Criterion	Implemented mechanism	Evidence and limitation
D1 Local execution	SQLite, FTS5, USearch, and ONNX Runtime in one process	Confirmed by source inspection; first-use model download remains external
D2 Evidence preservation	Source memory and session identifiers in derived records and proof packets	The ledger table is not populated by normal ingestion; no corpus-wide orphan audit has been run
D3 Temporal update	Validity intervals, status fields, and supersession edges	Unit tests cover fact replacement and point-in-time selection; the public temporal endpoint is incomplete
D4 Heterogeneous retrieval	Session routing plus HNSW, FTS5, cards, events, profiles, facets, and graph candidates	Observed in the LoCoMo query path; the contribution of each lane is not isolated
D5 Bounded operation	One server process and file-backed data directory	Architecture confirmed; resource bounds, scaling, and Raspberry Pi operation have not been measured

5.4 Target invariants and audited status

A database paper needs correctness properties that can be tested independently of retrieval quality. We use four target invariants:

I1: Tenant isolation
A query, vector label, or derived identifier from tenant u must never resolve to data owned by tenant u'.

I2: Provenance closure
Every live derived object must resolve to a live source record or an explicit tombstone.

I3: Temporal uniqueness
At most one fact version is current for a tenant, entity, and fact key at a query time.

I4: Index convergence
After recovery, every committed index operation is reflected once in each enabled secondary index, and no uncommitted operation is visible.

The audited revision does not establish all four invariants. Fact-version tests provide evidence for I3 in ordinary cases. Source identifiers support I2, but deletion is not reliable enough for closure to be claimed. Tenant-local SQLite row identifiers are reused as labels in a process-wide HNSW index, which can violate I1. Non-atomic secondary writes and the absence of a replay protocol leave I4 unproved. These are findings of the artifact audit, not properties hidden behind the retrieval score.

5.5 Artifact verification

The exact code-size and test details are reported in Appendix A, where they belong as artifact metadata. Two parallel CORS tests fail because they mutate shared process environment state. The affected path is separate from the LoCoMo evaluator, but the failures still prevent a clean reproducibility claim. A submission release should fix the race and run the suite in continuous integration on every supported architecture.

6 Evaluation methodology

6.1 Evaluation scope

The available artifact supports a feasibility evaluation of session retrieval. It does not support a matched comparison of architectures. We ask whether the recorded configuration can place at least one annotated source session in a fixed top-eight budget, how results vary by question type, and which stages account for the reported query time.

6.2 Dataset and filtering

The local LoCoMo data contains ten conversations, 272 sessions, and 1,986 raw question-answer records. The supplied evaluator removes records whose evidence field is null or empty, leaving 1,538 eligible questions. This filtering rule is reconstructed from the evaluator and recorded artifacts; it is not presented as an official LoCoMo protocol.

The eligible set contains 282 single-hop, 92 multi-hop, 841 open-domain, 321 temporal, and two adversarial questions. Of the 1,538 questions, 329 name more than one gold session. Those questions expose the main weakness of the reported metric: finding one gold session may leave other required evidence unretrieved.

6.3 Recorded configuration

Each query requests eight sessions and at most four chunks from each selected session. Ingestion concurrency is four. Cross-encoder reranking, semantic deduplication, and consolidation are disabled. The run therefore evaluates a first-stage configuration rather than every implemented retrieval component.

The historical log does not record CPU, GPU, RAM, operating system, storage, power draw, model hashes, or a source commit. A second file named `longmemeval_output.txt` also invokes the LoCoMo evaluator and cannot be used as a LongMemEval result.

6.4 Metric

Let G_q be the set of annotated evidence sessions for question q, and let $R_q^{(8)}$ be the first eight retrieved sessions. The evaluator reports

$$\text{AnyHit@8} = \frac{1}{|Q|} \sum_{q \in Q} \mathbb{1}\left[G_q \cap R_q^{(8)} \neq \emptyset \right]. \tag{5}$$

We call this question-level any-gold-session hit rate at eight. It is not complete-evidence recall: a multi-session question succeeds when the intersection contains one session. It also does not score ranking within the top eight, answer generation, faithfulness, or abstention.

6.5 Independence and tuning risk

The 1,538 questions are nested within only ten conversations and should not be treated as independent experimental units. We report exact counts and descriptive rates rather than a question-level significance test. A future comparison should use paired conversation-clustered bootstrap intervals or leave-one-conversation-out analysis.

The source comments describe default weights and a time-window bonus as empirically tuned on LoCoMo. The repository also contains a ranking tuner that searches configurations on supplied candidate dumps. No train, development, and held-out test separation is recorded for the historical run. Its 95.8% result is therefore development-set evidence and should not be interpreted as generalisation to unseen conversations.

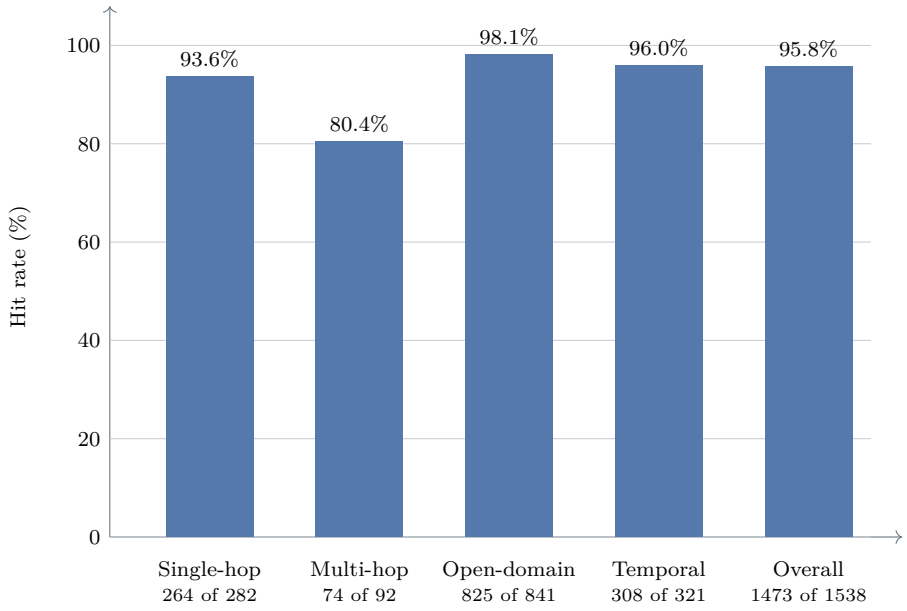

Figure 4: Question-level any-gold-session hit rate at eight in the recorded LoCoMo run. The evaluator's printed category table omits two adversarial questions; both are included in the overall count.

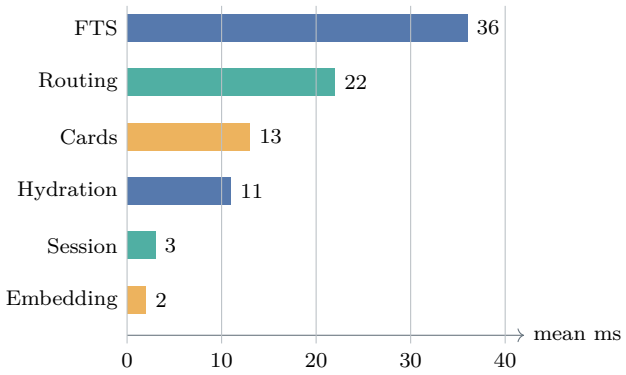

Figure 5: Mean engine-stage times in the same historical run. A reported zero denotes a rounded value below 1 ms or a disabled stage.

7 Results

7.1 Session retrieval

Figure 4 reports 1,473 hits among 1,538 eligible questions, giving an overall rate of 95.8%. Open-domain questions have the highest rate, 825/841 or 98.1%. Temporal questions reach 308/321 or 96.0%, and single-hop questions reach 264/282 or 93.6%. Multi-hop questions have the lowest rate, 74/92 or 80.4%. Both adversarial questions are hits.

The 65 misses are distributed across the four main categories: 18 single-hop, 18 multi-hop, 16 open-domain, and 13 temporal. Multi-hop contributes as many misses as single-hop despite having roughly one third as many questions. This is the clearest observed failure concentration. The current log does not contain enough per-query evidence detail to determine whether those misses originate in session routing, candidate generation, or final truncation.

7.2 Recorded stage times

The mean observed query time is 94 ms and the mean engine-reported total is 93 ms. FTS is the largest named stage at 36 ms, followed by session routing at 22 ms, memory-card search at 13 ms, and hydration at 11 ms. Embedding averages 2 ms. ANN, fusion, preference, graph, and reranking round to zero in the printed

summary.

These timings describe the recorded run only. Without hardware, corpus-state, cache, and repetition metadata, they cannot establish an interactive-latency threshold or an advantage from collocation. They do identify paths for a controlled profile: lexical search, routing, and hydration account for most of the named time.

The evaluator also reports 675 ms of amortised ingestion contribution per question and 772 ms total when ingestion is included. Conversation sessions are ingested once and reused across their questions, so this is not a per-memory write latency. The unweighted mean of the ten reported conversation-level batch-ingest averages is 5,426 ms; it is not a throughput measurement.

7.3 Answer to RQ2

The recorded configuration recovers at least one annotated source session for most eligible LoCoMo questions, but multi-hop questions remain substantially harder. This answers RQ2 only for development-set session retrieval. The evidence does not show complete evidence coverage, answer quality, superiority to another system, or generalisation beyond the tuned dataset.

7.4 Experiments required for comparative claims

A comparative evaluation should freeze the implementation and ranking parameters before testing. Retrieval conditions should include FTS-only, dense-only, FTS plus dense, routing-disabled, temporal-disabled, graph-disabled, and full TelloDB configurations under the same source text, candidate budget, and query order. Primary retrieval outcomes should include complete-evidence Recall@k, any-evidence Hit@k, MRR, nDCG, and Precision@k for $k \in \{1, 2, 4, 8, 16\}$.

One fixed answer model and prompt should then measure official answer score, exact match or token F1, citation correctness, faithfulness, and abstention. Systems measurements require repeated cold and warm runs with p50, p95, and p99 latency, ingestion throughput, peak memory, index size, startup time, and resource use. Fault injection and adversarial tenant tests are necessary for the database and privacy-related claims.

8 Discussion

8.1 Answers to the research questions

RQ1 concerns the coexistence of heterogeneous memory representations. TelloDB separates raw memories, fact versions, FTS indexes, graph edges, and vector mappings by type while connecting them through source memory, entity, and session identifiers. This supports mixed query planning without separate database services. The implementation does not yet give all representations a shared atomic commit or recovery protocol, so collocation should not be confused with transactional unity.

RQ2 concerns observed retrieval behaviour. The development-set LoCoMo run shows high any-gold-session coverage overall and a clear weakness on multi-hop questions. Because the metric rewards partial evidence and the weights may have been tuned on LoCoMo, it does not establish that hybrid or coverage-aware selection caused the result. The next experiment must compare complete evidence sets at a fixed context budget.

RQ3 concerns barriers to deployment claims. The audit finds four immediate classes of risk: tenant-unsafe vector labels, incomplete authentication coverage, non-atomic secondary indexing, and incomplete deletion or temporal paths. These findings answer the question more directly than an unmeasured claim about enterprise or Raspberry Pi readiness.

8.2 Coverage selection as the testable mechanism

The one-process architecture is useful engineering, but integration alone is a weak scientific claim. The greedy coverage selector gives the work a narrower hypothesis: under the same context budget, selecting sessions for marginal requirement, facet, entity, and temporal coverage should recover more complete evidence than sorting by independent relevance score.

That hypothesis has not been tested by the current evaluator. A defensible experiment would freeze query decomposition and candidate generation, then compare score-only and coverage-aware selection on the same candidate pools. Primary outcomes would be complete-evidence Recall@k, downstream answer F1, returned

context tokens, and paired differences clustered by conversation. This separates the proposed mechanism from gains caused by additional retrieval lanes or larger candidate sets.

8.3 Consistency and isolation

Collocation removes network boundaries and concentrates failures. SQLite protects relational transactions, but HNSW checkpointing and several secondary writes occur outside the primary transaction. A crash can leave a stored memory absent from a secondary index. Recovery research should add an index journal or outbox, idempotent operation identifiers, and a committed watermark. Fault injection can then test whether replay converges to the durable state.

The process-wide HNSW index has an immediate isolation defect. Its labels are SQLite row identifiers allocated independently by each tenant database. Two tenants can allocate the same label, allowing insertion or lookup to replace or remap another tenant's vector. Globally unique labels or separate per-tenant indexes are required before invariant I1 can hold.

The MCP handler also bypasses the principal-to-tenant flow used by the main handlers and opens the default tenant. In debug mode, the server can bind to `0.0.0.0` with a known fallback API key. These defaults are incompatible with an enterprise security claim. They should be removed, tested adversarially, and covered by endpoint-level authorisation tests.

8.4 Data locality is not complete privacy

After model provisioning, local embedding avoids sending raw memory text to a hosted embedding API. This is a data-locality property. It does not provide encryption at rest, TLS, formal non-interference, secure key storage, audit logging, or verified erasure from WAL files, checkpoints, and backups.

Deletion currently queries a column absent from the active memories schema. The temporal-query endpoint returns an empty result, and ordinary ingestion does not populate the metric vault. These paths must be repaired and tested before they are described as completed user-facing capabilities.

8.5 Ethics and data governance

This study uses the synthetic LoCoMo benchmark and does not recruit human participants. A deployed memory engine would process information that may be personal, confidential, or subject to retention rules. Keeping embeddings and indexes on an administered host reduces external transfer, but also concentrates sensitive history in database, WAL, model-cache, and checkpoint files. Deployment therefore requires a documented lawful basis, access and retention policy, deletion verification, and a decision about whether retrieved context may be sent to an external answer model.

8.6 Edge-device implications

One process and CPU support are useful prerequisites for edge deployment, but the default configuration is not a low-resource profile. It creates four BGE Small embedding executors and two BGE Reranker Base executors, sets a SQLite pool minimum of 16 connections, requests a 256 MiB cache, and configures a 1 GiB memory-mapped region. No Raspberry Pi binary, memory trace, latency distribution, temperature record, or energy measurement is available.

A Pi study should first define a bounded profile: one embedding executor, conditional or disabled reranking, a smaller connection pool, reduced cache and mapping settings, and fixed model hashes. The study should report the quality-resource frontier rather than asking only whether the process starts.

8.7 Threats to validity

The benchmark is one historical run whose exact source state and hardware are unknown. Ranking parameters may have been tuned on the same ten LoCoMo conversations. The evaluator filters 1,986 raw questions to 1,538 eligible questions, uses a permissive intersection metric, and does not retain enough raw per-query evidence for a complete error analysis. These conditions make the reported value suitable for development evidence, not a held-out performance claim.

LoCoMo questions are clustered within ten synthetic conversations. Results may differ for other languages, domain terminology, speech transcripts, code, or adversarial memory. The test suite is mostly unit-level and contains two order-dependent failures. Sustained concurrency, crash recovery, disk corruption, cross-tenant access, and verified deletion need dedicated system tests.

8.8 Path to a comparative study

The next study should isolate three effects. First, test coverage-aware selection against score-only ordering. Second, ablate each retrieval lane under a fixed candidate and token budget. Third, compare collocated and federated executions using identical models and ranking logic. A clean tagged release, held-out protocol, raw per-query records, repeated hardware runs, and conversation-clustered intervals are prerequisites for those comparisons.

9 Conclusion

TelloDB demonstrates that a temporal, lexical, relational, and dense agent-memory substrate can be assembled in one Rust process with local inference and file-backed storage. Its design retains raw memory records, versions changing facts, links derived records to source memory or session identifiers, and uses coverage-aware session selection for queries that appear to require several kinds of evidence.

A historical LoCoMo run records 1,473 any-gold-session hits among 1,538 eligible questions at a top-eight session budget. Multi-hop questions have the lowest category result. Since the weights may have been tuned on LoCoMo, the metric permits partial evidence, and the log lacks hardware and source metadata, this result is development evidence rather than a comparative benchmark.

The artifact audit also exposes the conditions under which the architecture is not yet safe to deploy: tenant-local vector labels can collide, secondary index updates are not atomic with primary storage, MCP tenant scoping is incomplete, and deletion and temporal endpoints require repair. These findings define the next research step. A held-out evaluation of coverage-aware selection, coupled with journaled index recovery and adversarial isolation tests, would turn the present feasibility study into a test of retrieval and database guarantees.

Author contributions

Sharjeel Abbas conceived and implemented the TelloDB prototype, built the evaluation harness, and led the systems analysis. Tehreem Arooj contributed the artificial-intelligence research framing, methodological review, and manuscript revision. Both authors approved the manuscript.

Data and artifact availability

The TelloDB source code is publicly available at `https://github.com/tellodb/tellodb` [22]. The implementation audit used local base commit `9be8de2` with uncommitted working-tree changes. The paper package includes the recorded aggregate benchmark values and their source locations. A tagged release and archival identifier are still needed to reproduce a submission version exactly.

References

[1] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025. doi: 10.48550/arXiv.2504.19413. URL https://arxiv.org/abs/2504.19413.

[2] Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 758–759, 2009. doi: 10.1145/1571941.1572114.

[3] Curtis Dyreson, Fabio Grandi, Wolfgang Käfer, Nick Kline, Nikos Lorentzos, Yannis Mitsopoulos, Angelo Montanari, Daniel Nonen, Elisa Peressi, Barbara Pernici, John F. Roddick, Nandlal L. Sarda, Maria Rita

Scalas, Arie Segev, Richard T. Snodgrass, Mike D. Soo, Abdullah Tansel, Paolo Tiberio, and Gio Wiederhold. A consensus glossary of temporal database concepts. *ACM SIGMOD Record*, 23(1):52–64, 1994. doi: 10.1145/181550.181560.

[4] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. From local to global: A graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024. doi: 10.48550/arXiv.2404.16130. URL https://arxiv.org/abs/2404.16130.

[5] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 6769–6781, 2020. doi: 10.18653/v1/2020.emnlp-main.550.

[6] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

[7] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638.

[8] Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13851–13870, 2024. doi: 10.18653/v1/2024.acl-long.747.

[9] Yu A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020. doi: 10.1109/TPAMI.2018.2889473.

[10] C. Mohan, Don Haderle, Bruce Lindsay, Hamid Pirahesh, and Peter Schwarz. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Transactions on Database Systems*, 17(1):94–162, 1992. doi: 10.1145/128765.128770.

[11] National Institute of Standards and Technology. Nist privacy framework: A tool for improving privacy through enterprise risk management, version 1.0. Technical report, National Institute of Standards and Technology, 2020. URL https://www.nist.gov/privacy-framework.

[12] Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with BERT, 2019. URL https://arxiv.org/abs/1901.04085.

[13] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. doi: 10.48550/arXiv.2310.08560. URL https://arxiv.org/abs/2310.08560.

[14] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22, 2023. doi: 10.1145/3586183.3606763.

[15] Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025. doi: 10.48550/arXiv.2501.13956. URL https://arxiv.org/abs/2501.13956.

[16] Soham Ratnaparkhi, Nishkarsh Srivastava, Aadil Garg, and Pratham Garg. Hydra DB: Beyond flat embeddings for production ai agents. Technical report, Hydra DB, 2026. URL https://benchmarks.hydradb.com/HydraDB.pdf. Accessed 28 July 2026.

[17] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009. doi: 10.1561/1500000019.

[18] SQLite Project. Sqlite fts5 extension, 2026. URL https://www.sqlite.org/fts5.html. Accessed 28 July 2026.

[19] SQLite Project. Sqlite is a self-contained system, 2026. URL https://sqlite.org/selfcontained.html. Accessed 28 July 2026.

[20] SQLite Project. Sqlite is serverless, 2026. URL https://sqlite.org/serverless.html. Accessed 28 July 2026.

[21] Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=1i6ZCvflQJ.

[22] TelloDB contributors. TelloDB: A temporal memory database for ai agents. https://github.com/tellodb/tellodb, 2026. Source-code repository. Accessed 28 July 2026.

[23] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, et al. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2614–2627, 2021. doi: 10.1145/3448016.3457550.

[24] Xubin Wang, Zhiqing Tang, Jianxiong Guo, Tianhui Meng, Chenhao Wang, Tian Wang, and Weijia Jia. Empowering edge intelligence: A survey on on-device AI models. *arXiv preprint arXiv:2503.06027*, 2025. doi: 10.48550/arXiv.2503.06027. URL https://arxiv.org/abs/2503.06027.

[25] Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. In *The Thirteenth International Conference on Learning Representations*, 2025. doi: 10.48550/arXiv.2410.10813. URL https://arxiv.org/abs/2410.10813.

[26] Qianxi Zhang, Shuotao Xu, Qi Chen, Guoxin Sui, Jiadong Xie, Zhizhen Cai, Yaoqi Chen, Yinxuan He, Yuqing Yang, Fan Yang, Mao Yang, and Lidong Zhou. VBASE: Unifying online vector similarity search and relational queries via relaxed monotonicity. In *17th USENIX Symposium on Operating Systems Design and Implementation*, pages 377–395, 2023. URL https://www.usenix.org/conference/osdi23/presentation/zhang-qianxi.

[27] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8):1738–1762, 2019. doi: 10.1109/JPROC.2019.2918951.

A Reproduction checklist

Table 3: Status of the current research artifact.

Item	Status
Engine source	Rust crate, version 0.1.0, edition 2021
Public repository	https://github.com/tellodb/tellodb
Audited base commit	`9be8de2`; working tree not clean
Implementation size	24,841 Rust lines across 47 source files
Unit tests	534/536 passed in repeated parallel run; two CORS env-race failures
Benchmark dataset	LoCoMo-10, 1,538 evaluated questions
Primary metric	Question-level any-gold-session hit rate at eight
Recorded overall result	95.8%
Recorded mean query time	94 ms observed, 93 ms engine-reported
Hardware metadata	Missing from historical output
LongMemEval result	Not available in the supplied artifacts
Raspberry Pi result	Not yet measured
External answer model	Not used in the reported recall run
Ranking validation	Defaults may have been tuned on LoCoMo; no held-out split recorded
Raw per-query result archive	Not available in the supplied benchmark summaries

B Principal API surface

Table 4: Research-relevant engine endpoints.

Endpoint	Intended purpose	Audited status
`POST /ingest`	Store an observation and derived artifacts	Implemented; secondary writes are not one atomic commit
`POST /batch-ingest`	Batch the ingestion pipeline	Implemented and used by the evaluator
`POST /query`	Run hybrid retrieval and evidence checks	Implemented; main evaluated query path
`GET /temporal/query`	Restrict retrieval to a time interval	Incomplete; returns an empty result
`POST /graph/query`	Query typed graph edges	Implemented; not evaluated here
`POST /graph/walk`	Traverse a bounded graph neighbourhood	Implemented; not evaluated here
`POST /analytics/query`	Compute metric aggregates	API exists; normal ingestion does not populate metrics
`POST /memory/inspect`	Inspect source, lifecycle, and versions	Implemented; not evaluated here
`POST /memory/delete`	Delete and record a tombstone	Blocked by a schema mismatch in the audited revision
`POST /mcp`	Expose JSON-RPC memory tools	Uses the default tenant outside the normal auth flow

C Default ranking parameters

Table 5: Selected defaults from the implementation. Source comments associate some defaults with LoCoMo tuning, so they are not independently validated parameters.

Signal	Weight	Signal	Weight
Lexical	0.05	Entity	0.05
Temporal	0.05	Event	0.18
Session ANN	0.12	Shadow question	0.12
Facet	0.08	Profile	0.10
Graph	1.00	Evidence density	0.03
Stale penalty	-0.08	Contradiction penalty	-0.10

D Proposed edge-device protocol

For a Raspberry Pi 5 study, define a low-resource profile before measurement: one embedding executor, disabled or conditional reranking, a smaller SQLite pool, and reduced cache and memory-map settings. Record the board revision, RAM capacity, storage medium, power supply, cooling, operating-system image, kernel, Rust toolchain, ONNX Runtime build, thread counts, model hashes, and compiler flags. Run each condition after a cold start and after warm-up. Report model-load time, peak resident memory, database and index size, ingestion items per second, query latency percentiles, hit rate at eight, CPU temperature, throttling events, and energy per query measured at the wall. Compare at least two embedding models and a lexical-only control.