Graph-Native Cognitive Memory for AI Agents: Formal Belief Revision Semantics for Versioned Memory Architectures

Kumiho Inc.*

February 23, 2026

Abstract

While individual components for AI agent memory—versioning, retrieval, consolidation—exist in prior systems such as Graphiti, Mem0, and Letta, their architectural synthesis and formal grounding remain underexplored. This paper presents **Kumiho**, a graph-native cognitive memory architecture grounded in formal belief revision semantics. AI agents increasingly produce substantial outputs—code, designs, documents, intermediate results—that accumulate without systematic versioning, provenance tracking, or linkage to the decisions that created them. In multi-agent workflows—where one agent's output becomes the next agent's input—this lack of structure is a fundamental bottleneck. The core insight is that the structural primitives required for cognitive memory—immutable revisions, mutable tag pointers, typed dependency edges, URI-based addressing—are *identical* to those required for managing these agent-produced outputs as versionable, addressable, dependency-linked assets. Rather than building a memory layer and a separate asset tracker, we built a cognitive memory architecture whose graph-native primitives inherently serve as the operational infrastructure for multi-agent work: agents use the same graph to remember, to find each other's outputs, and to build upon them.

The central formal contribution is a correspondence between the AGM belief revision framework Alchourrón et al. [1985] and the operational semantics of a property graph memory system. We frame this correspondence at the level of *belief bases* Hansson [1999], proving satisfaction of the basic AGM postulates ($K*2$–$K*6$) and Hansson's belief base postulates (Relevance, Core-Retainment), while providing a principled rejection of the Recovery postulate grounded in immutable versioning. The formal results hold for a deliberately simple propositional logic over ground triples—a tradeoff of expressiveness for tractability that avoids the Flouris et al. impossibility results for description logics Flouris et al. [2005]. We identify the supplementary postulates ($K*7$, $K*8$) as open questions requiring further formal development.

The architecture implements a dual-store model (Redis working memory, Neo4j long-term graph)—a recognized pattern in the agent memory space Hu et al. [2025]—with hybrid retrieval across full-text and vector modalities. An asynchronous consolidation pipeline extends prior sleep-time compute approaches [2025] with safety guards adapted from distributed systems and content management: published-item protection, circuit breakers, dry-run validation, and auditable cursor-based resumption. Critically, the same graph that stores an agent's memories also manages its work products: downstream agents locate inputs via URI resolution, track which revision is current via tag pointers, and link their own outputs back via typed edges—while human operators audit the entire chain with the same SDK and inspection tools. On the LoCoMo benchmark [2024] (token-level F1 with Porter stemming), Kumiho achieves 0.447 four-category F1

*The structural correspondence between cognitive memory and asset management was recognized through the author's experience building pipeline infrastructure for major visual effects productions. Contact: support@kumiho.io

(n=1,540)—the highest reported score across the retrieval categories—combined with 97.5% adversarial refusal accuracy (n=446), a natural consequence of belief revision semantics: the memory graph contains no fabricated information, so there is nothing for the model to hallucinate from. Including adversarial (binary scoring), overall F1 is 0.565 (n=1,986). Automated AGM compliance testing (49 scenarios, 7 postulates) confirms 100% operational adherence to the claimed formal properties. On LoCoMo-Plus [2026]—a Level-2 cognitive memory benchmark testing implicit constraint recall under intentional cue-trigger semantic disconnect—Kumiho achieves 93.3% judge accuracy (n=401, all four constraint types), outperforming the best published baseline (Gemini 2.5 Pro, 45.7%) by 47.6 percentage points. Recall accuracy reaches 98.5% (395/401), with the remaining 6.7% end-to-end gap entirely attributable to answer model fabrication on correctly retrieved context. Three architectural innovations drive both results: *prospective indexing*, where LLM-generated hypothetical future scenarios are indexed alongside each memory summary, bridging the cue-trigger semantic gap at write time; *event extraction*, where structured events with consequences are appended to summaries to preserve causal detail that narrative compression would otherwise drop; and *client-side LLM reranking*, where the consuming agent's own LLM selects the most relevant sibling revision from structured metadata at zero additional inference cost. The architecture is model-decoupled: switching the answer model from GPT-4o-mini (∼88%) to GPT-4o (93.3%) improves end-to-end accuracy by 5.3 points without any pipeline changes, demonstrating that recall accuracy is a property of the architecture, not the model. The system uses GPT-4o-mini for the bulk of LLM operations at a total cost of ∼$14 for 401 entries.

Keywords: cognitive memory, AI agents, belief revision, AGM postulates, knowledge graphs, memory architecture, property graph, Neo4j, retrieval-augmented generation, Model Context Protocol

1 Introduction

The role of large language models has shifted fundamentally. LLMs are no longer stateless question-answering systems or conversational chatbots. They operate as AI agents: autonomous workers that execute multi-step tasks, produce digital artifacts, make consequential decisions, and collaborate with humans and other agents across extended workflows Wang et al. [2024], Zhang et al. [2024]. A coding agent writes and commits code. A design agent generates and iterates on visual assets. A research agent gathers, synthesizes, and reports findings. A production agent coordinates tasks across departments and tracks deliverables.

This shift from chatbot to worker creates two intertwined requirements that current architectures address separately, if at all. First, agents need **cognitive memory**: the ability to remember past interactions, track how beliefs evolved, recall why decisions were made, and consolidate experience into reusable knowledge. Second, agents need **work product management**: the ability to version, locate, and build upon the artifacts they produce—so that a downstream agent in a multi-step pipeline can find the right input revision, understand its provenance, and link its own output back to the chain.

These two requirements share deep structural parallels. Remembering that "the client prefers warm color palettes" (cognitive memory) and tracking that "environment-lighting revision 5 is the approved version" (work product management) both require the same core primitives: immutable versioned snapshots, typed dependency edges, mutable status pointers, and content-reference separation. The domains are not identical—cognitive memory involves belief revision semantics, uncertainty, and natural language, while asset management operates on well-typed artifacts with deterministic workflows—but the underlying *data model* is shared. The present work builds a single graph-native cognitive memory architecture whose primitives simultaneously serve as the operational infrastructure for managing agent work products—enabling multi-agent pipelines where each agent's output is automatically versioned, addressable, and linked to the reasoning that produced it (Section 7

addresses the belief revision semantics specific to cognitive memory; Section 6 addresses the asset management extensions).

This dual-purpose design addresses a growing enterprise need. As AI agents transition from experimental tools to production workers, organizations require the same governance, traceability, and accountability standards they apply to human employees. Industry frameworks are emerging to meet this demand: ISACA's 2025 analysis of agentic AI audit challenges identifies identity management, decision traceability, and accountability gaps as critical concerns; IBM has introduced Agent Decision Records for structured agent accountability; and enterprise platforms increasingly frame agents as workforce members requiring governance infrastructure. Our architecture provides this governance natively: every agent belief has a URI, a revision history, provenance edges to source evidence, and an immutable audit trail—the same accountability infrastructure that organizations already apply to version-controlled code and managed digital assets.

The most common alternative response to the memory challenge has been to expand the context window itself: from 4K tokens in early models to 128K, 200K, and beyond. While larger windows accommodate more immediate context, they do not constitute memory. A context window is consumed fresh on every invocation; it provides no mechanism for versioning beliefs, tracking evidential provenance, expressing dependency relationships between conclusions, or consolidating many episodes into generalized knowledge. The distinction is analogous to the difference between a whiteboard and a filing system. A larger whiteboard lets you write more at once, but it is erased between sessions.

We contribute not novel individual components—many of which exist in concurrent systems such as Graphiti [2025], Mem0 [2025], and A-MEM [2025]—but a novel architectural synthesis grounded in formal analysis, implemented as the **Kumiho** system.[1] The specific contributions are:

1. **Unified cognitive memory and asset management architecture** (Section 4): A graph-native cognitive memory system whose structural primitives—immutable revisions, typed edges, mutable tag pointers, URI addressing—simultaneously serve as the operational infrastructure for managing agent work products. Agents use the same graph to remember past interactions and to version, locate, and build upon each other's outputs, enabling fully autonomous multi-agent pipelines without separate asset tracking systems. This paper validates the cognitive memory capabilities empirically (Sections 15.2–15.3); the asset management unification is an architectural contribution whose multi-agent pipeline validation is planned as future work.
2. **Formal belief base revision correspondence** (Section 7): A structural correspondence between the AGM belief revision postulates and graph-native memory operations, framed at the belief base level Hansson [1999], with a principled rejection of the Recovery postulate and formal avoidance of the Flouris et al. impossibility results.
3. **URI-based universal addressing**: A memory reference scheme enabling deterministic resolution and provenance traversal. Our URI scheme provides hierarchical scoping, revision pinning, and artifact addressing. Among agent memory systems, we found no prior use of structured, hierarchical URIs with these properties. This provides the mechanism for formal constructs: every belief has a dereferenceable address, every revision is citable, and every provenance chain is traceable.
4. **Safety-hardened consolidation** (Section 9): An asynchronous consolidation pipeline with novel safety mechanisms: published-item protection, circuit breakers, dry-run validation, and auditable cursor-based resumption.
5. **SDK transparency and multi-agent interoperability** (Section 5.2): The same graph and SDK that agents use to manage work products also exposes their memories for human inspection through a web dashboard and desktop asset browser. Agents query the graph to find inputs; operators query it to audit decisions—both through the same API, addressing the growing

[1] The core graph server is provided as a cloud service at `https://kumiho.io`. The Python SDK, MCP memory plugin, and benchmark suite are open-source: `https://github.com/kumihoclouds`.

need for AI memory observability alongside operational multi-agent coordination.

6. **LoCoMo evaluation on the official metric** (Section 15.2): 0.447 four-category token-level F1 on the LoCoMo benchmark (n=1,540)—the highest reported score across the retrieval categories LoCoMo [2024]—combined with 97.5% adversarial refusal accuracy (n=446). The near-perfect adversarial score is a natural consequence of the belief revision architecture: the memory graph genuinely does not contain fabricated information, so there is nothing for the model to hallucinate from. Including adversarial (binary scoring), overall F1 is 0.565 (n=1,986). The BYO-storage architecture keeps raw conversation data entirely on the user's local storage while achieving state-of-the-art retrieval accuracy through structured summarization with enrichment.
7. **Empirical AGM compliance verification** (Section 15.7): An automated test suite of 49 scenarios across 5 categories verifying operational adherence to all 7 claimed postulates ($K*2$–$K*6$, Relevance, Core-Retainment), including adversarial edge cases (rapid sequential revisions, deep dependency chains, mixed edge types). 100% pass rate confirms that the implementation faithfully executes the formal specification.
8. **LoCoMo-Plus evaluation** (Section 15.3): 93.3% judge accuracy on LoCoMo-Plus (n=401)—a Level-2 cognitive memory benchmark testing implicit constraint recall under intentional cue-trigger semantic disconnect—outperforming the best published baseline (Gemini 2.5 Pro, 45.7%) by 47.6 percentage points. Recall accuracy reaches 98.5% (395/401), with 78% of failures attributable to answer model fabrication, not retrieval. Two consolidation enrichments—*prospective indexing* and *event extraction*—eliminated the >6-month accuracy cliff (37.5% → 84.4%). The architecture is model-decoupled: swapping the answer model from GPT-4o-mini (∼88%) to GPT-4o (93.3%) improves accuracy without pipeline changes, at a total cost of ∼$14.
9. **Cross-benchmark generalization and client-side reranking** (Sections 15.2, 15.3, 8.8): The same architecture achieves state-of-the-art on both LoCoMo and LoCoMo-Plus (contributions 6 and 8 above), validating that graph-native primitives generalize across evaluation protocols. Client-side LLM reranking—where the consuming agent's own LLM selects the most relevant sibling revision from structured metadata at zero additional cost—complements prospective indexing and event extraction as the third architectural mechanism driving both results.

The remainder of the paper covers related work, design principles, the formal AGM correspondence, architecture details (hybrid retrieval, consolidation pipeline, safety guards), empirical evaluations on LoCoMo (Section 15.2) and LoCoMo-Plus (Section 15.3), AGM compliance verification (Section 15.7), and future directions.

2 Related Work

2.1 Agent Memory Architectures

The landscape of agent memory systems has developed rapidly since 2023, with the ICLR 2026 MemAgents Workshop [2026] marking the field's maturation. We position our work relative to the most relevant concurrent systems. Hu et al. Hu et al. [2025] provide the current canonical survey.

Graphiti/Zep [2025] shares the most surface-level components with our system: it implements a temporal knowledge graph on Neo4j with entity-event synthesis, bitemporal versioning, and triple-modality hybrid retrieval combining BM25, cosine similarity, and graph traversal. Graphiti reports 94.8% accuracy on the Deep Memory Retrieval (DMR) benchmark and 18.5% improvement on LongMemEval. Our LoCoMo results (Section 15.2) are discussed in the evaluation sections. The key architectural differences are threefold: (i) we provide a formal belief revision correspondence that Graphiti lacks; (ii) our URI addressing scheme enables deterministic cross-system memory references; (iii) our BYO-storage design keeps raw conversation data on the user's local storage, whereas Graphiti processes and stores full content server-side.

Mem0/Mem0g [2025] implements a triple-store architecture with timestamped versioned memo-

ries and LLM-powered conflict resolution, reporting 26% improvement over OpenAI Memory on LoCoMo. **A-MEM** Xu et al. [2025] introduces Zettelkasten-inspired dynamic linking (NeurIPS 2025). **MAGMA** [2026] proposes a multi-graph architecture with four orthogonal graph layers (semantic, temporal, causal, entity), achieving the highest LoCoMo judge score of 0.70 with policy-guided retrieval traversal. MAGMA's design represents an alternative structural philosophy: it disentangles memory dimensions into separate graphs for cleaner retrieval routing, whereas our architecture unifies all relationships in a single property graph with typed edges, enabling cross-dimensional traversal (e.g., ANALYZEIMPACT propagating across all edge types simultaneously). Neither approach has been empirically compared to the other.

MemGPT/Letta [2023] pioneered virtual context extension. Letta's sleep-time compute [2025] introduced asynchronous background consolidation—the same biological metaphor we use—with anticipatory pre-computation (predicting future queries), which our Dream State does not currently address (Section 16). Our consolidation contribution is the safety guard architecture (Section 9.5). **Letta Context Repositories** [2026] (February 2026) provide git-backed memory filesystems with automatic versioning and merge-based conflict resolution via multi-agent worktrees. While this is the most structurally related production system to our versioned memory concept, the approach differs fundamentally in three respects:

(i) Versioning substrate. Letta uses Git—a source code versioning tool—pragmatically as a storage backend for Markdown files. Our graph-native approach yields richer versioning primitives than Git's file-and-commit model: typed cognitive edges (not just file diffs), multi-tag pointer layers (not just branch heads), artifact attachments (not just file content), and hierarchical project/space scoping (not just directory trees). These same primitives double as auditable asset management for agent work products (Section 4.2).

(ii) Conflict resolution. Letta resolves concurrent writes via Git's text-level merge, which can detect but not semantically resolve contradictory beliefs—the merge requires human or LLM intervention on the text diff. Our system resolves conflicts via AGM-compliant belief revision operators: the SUPERSEDES edge creates a new revision with formal guarantees (Success, Consistency, minimal change via Relevance).

(iii) Downstream propagation. In Letta, updating a belief in one file does not automatically identify other files that depend on it. In our system, ANALYZEIMPACT traverses typed DEPENDS_ON edges to identify all downstream conclusions that may need re-evaluation—a capability enabled by the typed edge ontology that flat file systems cannot express.

A concrete example illustrates these differences. If an agent's memory contains "client prefers warm tones" and a concurrent write produces "client now prefers cool tones," Letta's system generates a text-level merge conflict requiring human or LLM intervention. Our system creates a new revision with a SUPERSEDES edge, moves the tag pointer, and makes the stale belief retrievable only via explicit opt-in—with ANALYZEIMPACT propagating the change to downstream decisions automatically.

Additional concurrent systems include **EverMemOS** [2026] (self-reported 93.05% LoCoMo; see benchmark caveat below), CAM [2025] (NeurIPS 2025), and MemoryAgentBench [2026] (ICLR 2026). **MemOS** [2025] (EMNLP 2025 Oral) implements a hierarchical memory manager with global, local, and working memory buffers inspired by operating system design. **Hindsight** [2025] proposes a four-network memory architecture (facts, experiences, opinions, observations) achieving 89.61% on LoCoMo and 91.4% on LongMemEval with open-source evaluation. Its Opinion Network with confidence-scored beliefs that update with evidence represents a pragmatic form of belief versioning without formal AGM grounding. The empirical success of Hindsight's approach demonstrates that practical belief tracking delivers strong results; our work provides the formal framework that could guarantee consistency properties—specifically, that belief revision satisfies minimal change (Relevance) and does not discard beliefs without justification (Core-Retainment)—for systems like Hindsight. **AgeMem** [2026] unifies LTM/STM management via RL with tool-based operations and three-

stage progressive GRPO training. **E-mem** [2026] addresses multi-agent episodic context reconstruction, achieving 54%+ F1 on LoCoMo while reducing token cost by 70%. **LatentMem** [2026] introduces learnable multi-agent memory with Latent Memory Policy Optimization. **CAST** [2026] proposes character-and-scene episodic memory for agents. CoALA [2024] provides a cognitive science taxonomy informing our dual-store design.

Recent surveys provide complementary taxonomies. Hu et al. Hu et al. [2025] provide the current canonical survey with a comprehensive taxonomy of memory functions, substrates, and dynamics. Yang et al. Yang et al. [2026] survey graph-based agent memory from DEEP-PolyU, organizing systems by graph types and memory functions. Huang et al. Huang et al. [2026] provide a large-scale survey covering memory substrates, cognitive mechanisms, and memory subjects. Luo et al. Luo et al. [2026] trace the evolutionary trajectory from static storage to experience-driven memory mechanisms.

Table 1 summarizes the key architectural distinctions among the most relevant concurrent systems; a comprehensive feature comparison appears in Section 12.

Benchmark standardization caveat. No standardized LoCoMo leaderboard exists—all reported numbers use varying evaluation configurations (different judge models, question subsets, and evaluation prompts). EverMemOS's 93.05% is evaluated using their own framework with no independent reproduction. Li et al. [2026] introduced LoCoMo-Plus (February 2026), demonstrating that existing LoCoMo scores largely measure explicit factual recall under strong semantic alignment rather than genuine cognitive memory under cue-trigger semantic disconnect; all tested systems show substantial performance drops from LoCoMo to LoCoMo-Plus. To address the comparability gap, we now provide token-level F1 results (Section 15.2) on the official LoCoMo metric [2024], enabling direct comparison with Zep [2025], Mem0, Memobase [2025], and ENGRAM [2025] on the same scoring function. The DMR benchmark, while historically important, uses conversations (∼60 messages) that fit within modern context windows and relies exclusively on single-turn fact-retrieval questions Graphiti [2025]. All comparative numbers cited in this paper should be understood as approximate and non-standardized. Our LoCoMo-Plus evaluation (Section 15.3) further validates the strategic significance of graph-native architectures for Level-2 cognitive memory, where all tested baselines—including systems using premium models with million-token context windows—score between 23% and 46%.

2.2 Belief Revision in AI

The AGM framework Alchourrón et al. [1985], Gärdenfors [1988] defines rationality postulates for belief change. Hansson Hansson [1999] extended this to belief bases—finite sets not closed under logical consequence—which is the appropriate level for computational systems. Our formal analysis sits within this tradition.

Impossibility in Description Logics. Flouris et al. Flouris et al. [2005] proved that Description Logics (including those underlying OWL) cannot satisfy AGM revision postulates. Qi et al. Qi et al. [2006] refined this for specific fragments. These results are critical context: our property graph avoids these impossibilities by operating on a simpler structure (Section 7.6).

AGM and Machine Learning. Aravanis Aravanis [2025] establishes a correspondence between machine learning and AGM-style belief change; Hase et al. Hase et al. [2024] frame LLM model editing as belief revision. Baitalik et al. Baitalik et al. [2026] (AAAI 2026 Bridge Program) apply GreedySAT-based consistency to multi-turn dialogues. Wilie et al. Wilie et al. [2024] demonstrate LLMs' poor belief revision capabilities on BeliefR, motivating external memory architectures that enforce revision constraints structurally. Our work complements this stream by applying AGM to the external memory graph.

Formal Developments. Bonanno Bonanno [2025] unifies Bayesian and AGM approaches via Kripke–Lewis semantics, connecting belief revision to dynamic epistemic logic. Meng et al. Meng et al. [2025] develop belief algebras for iterated revision. Chandler and Booth Chandler and Booth [2025] address parallel belief revision via order aggregation (IJCAI 2025), directly relevant to the question

Table 1: Architectural comparison with selected concurrent systems.

Dim.	Graphiti	Mem0	Letta	Hindsight	Ours
Version	Bitemporal	Timestamped	Git	None	Immutable+tags
Conflict	Temporal	LLM	Git merge	Conf	AGM Supersedes
Retrieval	BM25+vec	Graph	File	4-net	Hybrid
Formal	None	None	None	None	AGM postulates
URI	×	×	Git	×	kref://
Consol	Temporal	None	Git GC	None	Dream State

of compositional belief change in batch operations. Schwind et al. Schwind et al. [2025] connect iterated belief change to learning, bridging the formal and computational perspectives.

The Recovery Postulate. Makinson Makinson [1987] questioned its status; Hansson Hansson [1991] proposed contraction without Recovery. Our principled rejection (Section 7.3) provides a concrete operational demonstration.

2.3 Versioned Knowledge Graphs

The concept of version-controlled knowledge graphs has a substantial history in the Semantic Web community. **R&Wbase** Vander Sande et al. [2013] (subtitled "Git for triples") supported branching and merging for quad-stores. **SemVersion** Völkel et al. [2005] applied version control to RDF graphs with structural diff and merge. **Quit Store** Arndt et al. [2018] ("Quads in Git") provides a SPARQL 1.1 endpoint backed by Git-versioned RDF named graphs with branching, merging, and provenance. **OSTRICH** Taelman et al. [2018] implements hybrid versioned triple storage with immutable snapshots and delta chains. **ConVerG** [2024] enables concurrent versioned querying of knowledge graphs using bitstring-based condensed representations.

Our contribution is not the invention of versioned graphs—this is well-established—but the application of versioned graph primitives to *cognitive memory specifically*, combined with formal belief revision analysis. The versioned KG systems above target SPARQL-based knowledge management; we target the distinct requirements of AI agent memory: typed dependency edges encoding epistemic relationships (not just ontological ones), mutable tag pointers for belief status tracking, asynchronous LLM-driven consolidation, and the formal correspondence to AGM postulates that Section 7 establishes.

Similarly, the broader analogy of version control applied to agent memory is increasingly explored. Git-Context-Controller GCC [2025] (GCC) explicitly applies Git semantics (COMMIT, BRANCH, MERGE) to LLM agent memory and achieves strong results on SWE-Bench-Lite. Our system differs in operating on typed knowledge graph triples rather than text files, enabling the formal properties described in Section 7 and the typed dependency reasoning unavailable to text-level version control.

2.4 Hybrid Retrieval and Score Fusion

Robertson and Zaragoza Robertson and Zaragoza [2009] formalized BM25. Cormack et al. Cormack et al. [2009] introduced Reciprocal Rank Fusion (RRF). Bruch et al. Bruch et al. [2023] analyzed fusion functions systematically, showing that convex combinations can outperform RRF on standard IR benchmarks. Our max-based fusion (CombMAX in the terminology of Fox and Shaw Fox and Shaw [1993]) is a deliberate design choice, not a novel fusion method. In memory retrieval, a strong exact-match signal on one branch should not be diluted by a weak score on another branch. We motivate this through a precision preservation observation (Section 8), but acknowledge that CombMAX is known to be susceptible to noise from poorly-calibrated retrievers producing inflated scores Bruch et al. [2023]. This is an argumentative design choice, not a theoretical result; comparative evaluation against RRF and convex combination is planned (Section 16).

2.5 AI Agent Observability

As AI agents increasingly perform consequential work, the need for memory observability has become acute. OpenTelemetry provides agent-specific telemetry schemas; tools like Langfuse, LangSmith, and Braintrust offer instrumentation for tracing inference steps and tool calls. Our contribution differs: existing observability tools trace inference; our system makes memory itself the auditable artifact. Every agent belief has a URI, revision history, provenance edges, and immutable audit trail. The web dashboard (Figure 1) and desktop asset browser render this graph as a browseable, searchable hierarchy with interactive visualization, enabling human operators to audit agent reasoning at the memory level—not just at the tool-call level.

3 Why Context Window Extension Is Not Memory

We identify four structural deficiencies of context-window-as-memory.

Attention is not recall. A context window provides attention capacity; memory requires selective recall from a large corpus of past experience.

Quadratic cost scaling. Transformer attention scales as $\Theta(n^2 \cdot d)$ Vaswani et al. [2017]. A retrieval-based system that indexes N memories and retrieves top-k incurs $O(\log N) + O(k^2 \cdot d)$. Since $k \ll n$ by design, the retrieval approach is orders of magnitude cheaper. Moreover, the metadata-over-content architecture (Section 6) reduces the per-item token size: each of the k recalled memories is a compact summary, not a full transcript. This compounds the cost advantage by reducing both the total token cost and the cognitive load on the reasoning model.

While techniques such as sparse attention, sliding windows, and linear attention approximations mitigate the quadratic cost, they do so by sacrificing the very capability that makes larger windows useful: the ability to attend to arbitrary positions in the sequence. An agent's lifelong interaction history measured in tens of millions of tokens makes in-context approaches economically and computationally infeasible for persistent memory.

No structural representation. A flat token sequence cannot express that belief B_2 supersedes B_1, that conclusion C depends on assumptions A_1, A_2, or that a memory has been validated, deprecated, or flagged for review.

Model coupling. Context window contents are ephemeral and model-specific. Agent memory must be LLM-decoupled: stored in a persistent, model-independent data structure that any current or future language model can query.

4 Background and Motivation

4.1 Human Memory as a Design Template

Cognitive science distinguishes between working memory—a capacity-limited, rapidly-accessible buffer Atkinson and Shiffrin [1968], Baddeley [2000]—and long-term memory, further divided into episodic and semantic stores Tulving [1972]. Transfer from episodic to semantic memory involves active consolidation during sleep Rasch and Born [2013]. Human decision-making proceeds through a loop of perceive–recall–revise–act that a memory-equipped AI agent must replicate.

4.2 The Asset Management Insight

AI agents increasingly produce substantial digital outputs—generated images, code commits, design iterations, research documents, intermediate results—that accumulate without systematic versioning, provenance tracking, or linkage to the decisions that created them. As sessions accumulate, these outputs become lost, duplicated, or untraceable. The structural correspondence between the primitives needed for cognitive memory and those needed for tracking these agent-produced outputs is not coincidental: both domains require immutable versioned snapshots, typed dependency edges, mutable status pointers, URI-based addressing, and content-reference separation. This correspondence was first recognized through production experience—asset management systems in visual effects and game development have implemented these exact primitives for decades—but the architectural direction is memory-first: the graph-native

Table 2: Structural correspondence between asset management and cognitive memory.

Asset	Memory	Concept
Project	Scope	Container
Space	Topic	Namespace
Item	Unit	Identity
Revision	Belief T	Snapshot
Artifact	Evidence	Pointer
Tag	Status	Mutable ref
Edge	Relation	Typed link
Bundle	Cluster	Grouping

primitives required for cognitive memory inherently provide auditable asset management as an emergent capability.

Traditional asset management systems encode these fundamentally graph-like relationships in relational databases (PostgreSQL, MySQL) with RPC frameworks (Thrift, gRPC), where dependency chains require recursive queries or application-level graph walks over foreign key joins. The relationships exist, but they are second-class citizens.

The present architecture adopts a native graph database (Neo4j) as the storage layer, making relationships first-class citizens. A dependency between two revisions that would require a junction table row and a multi-join query becomes a literal directed edge traversable in a single graph operation. This graph-native foundation enables the cognitive memory primitives (belief revision chains, evidential provenance, and the formal properties described in Section 7) while simultaneously providing the asset management capabilities (native dependency traversal, real-time impact analysis, typed relationship ontology) needed to track agent work products.

Table 2 formalizes this structural correspondence.

Principle 1 (Cognitive Memory as Multi-Agent Infrastructure)**.** *AI agents increasingly produce substantial outputs—generated images, code artifacts, design iterations, documents, intermediate results—that are not systematically tracked, versioned, or linked to the decisions that created them. In multi-agent workflows, this becomes a critical bottleneck: a video compositing agent cannot locate the approved texture revision produced by an upstream generation agent; an editing agent cannot trace which audio mix corresponds to which scene cut. The core insight is that the structural primitives needed for cognitive memory (immutable revisions, typed edges, mutable tag pointers, URI addressing) are* identical *to those needed for managing these agent-produced outputs as versionable, addressable, dependency-linked assets. Rather than building a memory layer and a separate asset tracker, we built a cognitive memory architecture whose graph-native primitives inherently serve as the operational infrastructure for multi-agent work—enabling agents to remember, find each other's outputs, and build upon them through a single system. An extensive literature search found no prior work unifying these two capabilities on a shared graph substrate.*

4.3 Dual-Purpose Graph: Memory and Asset Tracking as One System

The structural correspondence in Table 2 enables a concrete operational capability: the same graph that stores an agent's cognitive memories also manages the outputs that agents produce. Consider a multi-agent creative pipeline: an image generation agent produces a concept art revision; a video compositing agent locates that revision via its URI, checks that it carries the "approved" tag, and creates its own output with a DERIVED_FROM edge linking back to the input; an audio agent does the same for the soundtrack; and an editing agent assembles the final deliverable with typed edges to every upstream component. Each agent uses the graph both to *remember* (client preferences, past iterations, feedback history) and to *operate* (find the right input version, register its output, declare dependencies).

This dual-purpose design means agents in a pipeline do not need separate systems for "remembering what happened" and "managing the outputs they created." Both are first-class graph citizens. Without this unification, agent-produced artifacts accumulate in disconnected storage—files on disk, blobs in buckets, ephemeral context windows—with no versioning, no provenance, and no link to the reasoning that generated them. The graph ensures that every output is addressable, versionable, and traceable to the belief state that produced it—and that downstream agents can find and build upon it programmatically.

5 LLM-Memory Decoupling

A persistent failure mode in current agent memory systems is tight coupling between the memory store and the language model. When memory exists as prompt context, model-managed buffers, or framework-specific data structures, it becomes inseparable from the specific model version. This coupling creates provider lock-in, model upgrade fragility, and architectural non-portability.

5.1 The Decoupled Architecture

The architecture enforces LLM-memory decoupling through three mechanisms:

Model-independent storage. The memory graph is stored in a standard property graph database. Memories are structured data—summaries, metadata, tags, edges, artifact pointers—not model-specific embeddings or token sequences.

Standardized access protocol. Memory operations are exposed through the Model Context Protocol (MCP) MCP [2025], a model-agnostic tool interface. Any MCP-compatible agent can execute memory operations through the same interface.

Pluggable LLM integration. Components requiring LLM capabilities—the Dream State consolidation pipeline, PII redaction, and memory summarization—accept any LLM through an adapter interface.

5.2 Why Agent Work Products Demand Managed Memory

When an LLM serves as a chatbot, its "memory" is a convenience. When an LLM operates as an agent performing consequential work, memory becomes critical infrastructure—for two distinct reasons.

Operational need: multi-agent coordination. In autonomous pipelines, each agent's output is the next agent's input. An image generation agent produces texture revisions; a compositing agent must locate the approved revision, understand what created it, and link its own output back. Without structured asset management, these handoffs require brittle file paths, naming conventions, or ad-hoc metadata—none of which scale to complex multi-agent workflows. The memory graph provides this coordination natively: every output has a URI, a version history, typed dependency edges to its inputs, and mutable tag pointers indicating approval status. Agents query the graph to find inputs just as they query it to recall past interactions.

Governance need: decision auditability. An agent that approved a deployment, recommended a treatment plan, or signed off on a financial model must be able to explain *why*—not by regenerating a plausible explanation, but by pointing to the actual evidence, the actual reasoning chain, and the actual prior beliefs that informed the decision. This is the same accountability standard applied to human workers. The memory graph provides this: every agent belief has a URI, a revision history, provenance edges to source evidence, and an immutable audit trail. The web dashboard and desktop asset browser render this graph as a browseable hierarchy of versioned items (Figure 1). An operator reviewing an agent's decision can traverse from the decision memory to its DERIVED_FROM sources, check what the agent believed at the time via time-indexed tag resolution, and inspect the Dream State consolidation reports.

Both needs—operational coordination and governance auditability—are served by the same graph, the same SDK, and the same API. Without this infrastructure, agent work products are untraceable and agent pipelines are unmanageable. The system makes agent memory as inspectable as a version-controlled codebase and as navigable as a managed asset database—because structurally, it *is* both.

6 System Architecture

6.1 Overview

The architecture implements a dual-store model mirroring the human working/long-term memory distinction. The agent interacts through MCP; the memory graph is LLM-independent.

6.2 Working Memory: Library-Level Access

The working memory layer uses Redis, accessed via a direct library SDK (not HTTP), to maintain the current session's message buffer with a configurable TTL (default: 1 hour). Measured latencies are 2–5 ms via library SDK versus 150–300 ms through an HTTP gateway. Key design properties:

- Latency: <5 ms read/write via library SDK.
- Scope: Session-local; isolated per (project, context, user, session) via key namespace `cogmem:{proj}:sessions:{sid}:*`.
- Volatility: TTL-based expiry; bounded buffer (default 50 messages).
- Capacity: Session-local only; does not persist across agent restarts.
- Isolation: Strict context-level isolation via key namespacing (the context field in the session ID serves as the namespace boundary).

Principle 2 (Match Storage Latency to Access Pattern)**.** *Working memory requires library-level (not RPC-level) access to maintain sub-10 ms latency. The 100–200 ms difference between in-process and network access, compounded across thousands of interactions, accumulates to orders-of-magnitude differences in agent responsiveness.*

6.3 The Structured Memory Reference Scheme

Every object in the system is addressable through a universal URI scheme:

```
kref://project/space/
item.kind?r=N&a=artifact
```

The scheme provides: *Addressability*—any memory is referenceable from any context; *Temporal navigation*—`?r=N` pins to a specific point in time; *Type safety*—the `.kind` suffix enables type-aware retrieval; *Traversal entry points*—edges reference memory URIs, enabling graph traversal from any starting point.

Principle 3 (Universal Addressability)**.** *Every memory unit must be referenceable by a stable, parseable, human-readable identifier. Without addressability, edges become fragile, lineage becomes untraceable, and consolidation becomes lossy.*

Principle 4 (Validate at Boundaries, Trust Internally)**.** *The SDK validates all memory reference URIs at the boundary (regex matching against the canonical format). Once validated, internal code treats them as opaque strings, eliminating redundant per-function validation.*

6.4 Long-Term Memory: The Property Graph

Long-term memory is stored in Neo4j. Each memory unit is represented as an Item node with one or more Revision nodes forming an immutable version chain.

6.4.1 The Item–Revision Model

An Item represents a named, typed memory unit. Each Revision is an immutable snapshot carrying structured metadata (summary, topics, keywords, schema version) and optionally an embedding vector. An agent can: resolve a tag to retrieve the latest belief; follow SUPERSEDES chains to trace belief evolution; follow DERIVED_FROM edges to understand evidential support; and move a tag to an earlier revision to perform belief rollback.

```
Item: "api-design.decision"
  Rev 1 (Jan 15): "Use REST for public API"
  Rev 2 (Jan 22): "Use REST + WebSocket"
    edge: SUPERSEDES -> Rev 1
  Rev 3 (Feb 1):  "Use gRPC internally"
    edge: SUPERSEDES -> Rev 2
    edge: DERIVED_FROM -> "benchmarks.fact?r
    =1"
  Tag "current" -> Rev 3
  Tag "initial" -> Rev 1
```

Principle 5 (Immutable Revisions, Mutable Pointers)**.** *Memory states are never overwritten; they are versioned. Tags provide mutable "current view" semantics without losing history, enabling belief revision tracking, audit trails, and rollback.*

6.4.2 BYO-Storage: Metadata Over Content

The system stores metadata, relationships, and pointers—never file content. Artifact records contain a location field pointing to where raw content

Table 3: Memory type taxonomy and implementation.

Type	Implementation
Working	Redis buffer (TTL)
Episodic	Conversations
Semantic	Consolidated facts
Procedural	Tool execution
Associative	Graph edges
Meta	Tags + audit

resides on the user's own storage. This content-reference separation—a principle well-established in asset management systems handling petabytes of data—yields critical benefits for cognitive memory: the graph database stays lightweight, privacy boundaries are enforced architecturally, and agents read compact summaries rather than full transcripts.

Principle 6 (Metadata Over Content)**.** *Store the minimum information necessary for recall and reasoning in the cloud graph. Raw content stays local. This preserves user privacy, reduces storage costs, eliminates data exfiltration risk, and enables cognitive efficiency.*

6.4.3 Memory Type Taxonomy

The system implements six memory types: working memory (Redis buffer), episodic memory (conversation revisions), semantic memory (consolidated facts), procedural memory (tool execution records), associative memory (graph edges + bundles), and meta-memory (tag system + audit trail).

6.5 Edge System: Reasoning as First-Class Structure

The edge system makes reasoning structure explicit through six typed, directed edge types: DEPENDS_ON (validity dependency), DERIVED_FROM (evidential provenance), SUPERSEDES (belief revision), REFERENCED (associative mention), CONTAINS (bundle membership), and CREATED_FROM (generative lineage).

Edge type definitions:

- DEPENDS_ON: Validity dependency. If the target is invalidated, the source may be unreliable.
- DERIVED_FROM: Evidential provenance. The source was produced using the target as input.
- SUPERSEDES: Belief revision. The source replaces the target as the current belief.
- REFERENCED: Associative mention. The source refers to the target without dependency.
- CONTAINS: Bundle membership. The target is a member of the source bundle.
- CREATED_FROM: Generative lineage. The source was generated from the target.

These edges enable three traversal operations: TRAVERSEEDGES(k, d, n), SHORTESTPATH(k_s, k_t), and ANALYZEIMPACT(k, d). Impact analysis is particularly important for belief revision: when an agent discovers that assumption A is invalid, ANALYZEIMPACT identifies all downstream conclusions that may need re-evaluation.

Traversal operations:

1. **TraverseEdges**(k, d, n): Find all memories connected to memory reference k in direction d up to depth n.
2. **ShortestPath**(k_s, k_t): Find the minimal connection between two memory references.
3. **AnalyzeImpact**(k, d): Compute the transitive dependency cascade from memory reference k to depth d.

Principle 7 (Explicit Over Inferred Relationships)**.** *Relationships between memories must be stored as first-class graph edges, not inferred at query time through embedding similarity. Similarity finds related content; edges encode* why *content is related.*

Principle 8 (Why Graph-Native Edges Matter)**.** *In relational systems, a dependency between two assets required a junction table and a multi-join query. In native graph databases, the same relationship is a single directed edge traversable in $O(1)$ time. Recursive CTEs can compute transitive closure over relational joins, but they require exponentially more computation as depth increases. A single Cypher* `ShortestPath` *query achieves the same result in milliseconds. This difference compounds across all graph operations: traversal, impact analysis, provenance reconstruction.*

7 Formal Properties: Belief Revision in Graph-Native Memory

The item–revision–tag model is not merely an engineering convenience; its primitives correspond to fundamental operations studied in the theory of rational belief change. This section demonstrates a structural correspondence with the AGM postulates Alchourrón et al. [1985]—or, in one principled case, deliberately diverges from them.

We frame this contribution at the level of *belief bases* rather than belief sets, following Hansson Hansson [1999]. The contribution is the bridge itself: demonstrating that an independently-motivated systems architecture, designed for production agent memory, satisfies established rationality constraints under a belief base interpretation. We do not propose new logical machinery; the novelty lies in showing that production-motivated architectural choices yield a system satisfying AGM's postulates. To our knowledge, no prior work bridges AGM belief revision theory with AI agent memory architectures. The novelty lies in the *mapping* between graph-native memory operations and formal belief revision operators—not in the logic itself, which is deliberately kept at the propositional level for tractability. The distinction between belief sets and belief bases is fundamental: AGM's original formulation operates on belief sets closed under logical consequence ($K = \mathrm{Cn}(K)$), which are infinite objects unsuitable for computational systems. In contrast, computational systems store finite sets of explicitly stored propositions—precisely what the memory graph's revision content represents. Hansson's belief base framework operates on these finite, non-closed sets, providing the appropriate level of abstraction for agent memory. This framing avoids the well-known difficulties of computing deductive closure while retaining the rationality constraints that make AGM valuable.

7.1 Formal System Model

We formalize the memory graph and the belief state it induces.

Definition 7.1 (Memory Graph)**.** *A* memory graph *is a tuple $G = (I, R, E, \tau)$ where:*

- *I is a finite set of* items *(named, typed memory units);*
- *$R = \bigcup_{i \in I} R_i$ is the set of all* revisions*, where $R_i = (r_i^{(1)}, r_i^{(2)}, \ldots)$ is the ordered, append-only revision sequence for item i;*
- *$E \subseteq R \times \mathit{EdgeType} \times R$ is a set of typed, directed edges between revisions;*
- *$\tau : \mathit{TagName} \rightharpoonup R$ is a partial function mapping tag names to revisions (the mutable pointer layer).*

Each revision $r_i^{(k)}$ is immutable*: once created, its content $\varphi(r_i^{(k)})$ cannot be modified. Tags are the sole mutable component.*

Definition 7.2 (Belief Base)**.** *Given a memory graph $G = (I, R, E, \tau)$, the belief base is:*

$$\mathcal{B}(\tau) = \bigcup_{t \in \mathrm{dom}(\tau)} \varphi\big(\tau(t)\big) \tag{1}$$

where $\varphi(r)$ denotes the propositional content of revision r (operationally, the revision's structured metadata: summary, topics, keywords, extracted facts). Unlike the AGM belief set, $\mathcal{B}(\tau)$ is a finite set that is not *closed under logical consequence.*

The system records every tag assignment and reassignment, yielding a history function $\tau_T : \mathit{TagName} \rightharpoonup R$ that resolves the tag mapping as it existed at time T. This induces historical belief bases:

$$\mathcal{B}(\tau_T) = \bigcup_{t \in \mathrm{dom}(\tau_T)} \varphi\big(\tau_T(t)\big) \tag{2}$$

enabling the query "what was believed under tag t at time T?" to be answered precisely—not by scanning revision timestamps, but by resolving the actual tag-to-revision binding that was active at T.

An important implementation note: the propositional content $\varphi(r)$ is operationally the revision's *structured metadata*—its summary, topics, keywords, and extracted facts—not the raw conversation transcript or tool output. Raw content resides outside the graph as an artifact and is dereferenced only when exact detail is required. This two-tier representation makes $\varphi(r)$ compact and tractable for belief-state computation, while preserving full evidential fidelity through the artifact pointer.

From natural language to ground triples. A central implementation question is how agent-generated natural language beliefs—e.g., "the user prefers cool color tones"—are mapped to ground atoms in At_G (Definition 7.8). The mapping is performed at the *API boundary* by the `memory_ingest` MCP tool, which accepts structured fields: a `title` (becomes the item name), a `summary` (becomes the revision's primary content), `tags` and `topics` (become metadata keywords), and a `memory_type` classification (summary, decision, fact, reflection, error). The agent's natural language output is thus decomposed into typed fields before entering the graph—the mapping is not an automated NLP extraction pipeline but a *structured API contract* that the agent's skill prompt is designed to satisfy.

Concretely, the agent's statement "the user prefers cool tones for their design palette" is mapped to a ground triple $\langle$*color-preference*, *summary*, "prefers cool tones"$\rangle \in \mathrm{At}_G$ by the agent writing: `title="color-preference"`, `summary="prefers cool tones for design palette"`, `memory_type="decision"`. The triple structure (s, p, o) arises from the item–field–value decomposition: subject = item name, predicate = field name, object = field value. This is not semantic parsing in the NLP sense; it is structured output that the LLM agent produces by following its memory skill prompt, which specifies the expected fields and their semantics.

Property graph, not RDF. Despite the (s, p, o) notation, the underlying storage is a *labeled property graph* (Neo4j), not an RDF triple store. Each memory item is a graph node with typed properties (summary, topics, keywords, type); each revision is a separate node linked by SUPERSEDES edges; and inter-item relationships (DEPENDS_ON, DERIVED_FROM, etc.) are first-class typed edges with optional metadata. The "ground triple" formalism in At_G is an *analytical abstraction* used to establish the AGM correspondence—it provides a clean propositional atom structure for the formal proofs. The implementation stores these atoms as node properties in the property graph, not as RDF subject–predicate–object statements. This distinction matters because the property graph model supports features absent from basic RDF: per-edge metadata, native node-level indexing (fulltext and vector), and schema-flexible property bags on both nodes and edges. Readers familiar with RDF should understand At_G as a formalization convenience, not an architectural commitment to triple stores.

Three consequences follow. First, the quality of the mapping depends on the agent's skill prompt, not on a separate extraction module—prompt engineering is the primary control surface. Second, ambiguous or complex beliefs ("the user seems to prefer cool tones but mentioned warm tones for the bedroom") are captured as a single revision with a nuanced summary, not decomposed into multiple conflicting triples; the formal model handles the resulting compound belief through the mechanisms described in the partial merging discussion below. Third, the mapping is lossy by design: the full conversational context is preserved in the artifact (the raw transcript), while the ground triple captures only the distilled belief. This lossy compression is what makes $\mathcal{B}(\tau)$ tractable—a finite set of ground atoms rather than an unbounded natural language corpus—while the artifact system preserves evidential fidelity for cases requiring the original context.

Scope boundary. The NL-to-triple mapping is a *pre-formal* step: the formal properties established in Section 7.2 hold over the ground triple representation At_G, not over the natural language source. In particular, the mapping is *many-to-one in practice*—the same natural language belief ("the user prefers cool tones") could be mapped to different ground triples depending on the agent's choice of item name, summary wording, or metadata structure. Two semantically identical beliefs may therefore not be syntactically identical in At_G, which means that Extensionality ($K * 6$, Proposition 7.5) holds over the formal representation but cannot guarantee that the agent will consistently map equivalent natural language inputs to equivalent ground atoms. This is an inherent limitation of any system that bridges informal and formal representations; the consistency of the mapping is a prompt engineering concern, not a formal one. We treat the NL-to-triple mapping as an explicit assumption: the formal analysis begins *after* beliefs have been committed to the graph as ground triples, and the quality of the formal guarantees is bounded by the quality of this pre-formal mapping step.

Definition 7.3 (Two-Tier Epistemic Model)**.** *The memory system operates at two distinct epistemic levels:*

- *The* full graph $G = (I, R, E, \tau)$ *contains all revisions ever created, including deprecated items and archived (untagged) revisions. This is the operator-accessible store.*
- *The* agent retrieval surface $\mathcal{B}_{\text{retr}}(\tau)$ *is the subset of the belief state reachable through the retrieval pipeline. A revision r is in the retrieval surface if and only if: (i) r is referenced by at least one active tag in τ,* and *(ii) the item containing r is not marked as deprecated.*

We define the deprecation predicate: $\text{deprecated} : I \to \{\top, \bot\}$ *is a boolean property on each item, set to $\top$ by the contraction operator (Definition 7.5) and queryable as a node property in the graph database (`item.deprecated`). An item's deprecation status is mutable (it can be restored via explicit operator action) but defaults to $\bot$ at creation.*

Formally, let $I_{\text{active}} = \{i \in I \mid \neg\text{deprecated}(i)\}$. *Then:*

$$\mathcal{B}_{\text{retr}}(\tau) = \bigcup_{\substack{t \in \text{dom}(\tau) \\ \text{item}(\tau(t)) \in I_{\text{active}}}} \varphi\big(\tau(t)\big) \subseteq \mathcal{B}(\tau) \qquad (3)$$

This two-tier model is critical for the Consistency postulate ($K * 5$). Without retrieval exclusion, a superseded revision $r_i^{(k)}$ — still present in the graph and carrying content $\varphi(r_i^{(k)})$ that may entail $\neg A$ — could be surfaced by vector similarity search (the embedding of the old revision may be semantically proximate to the query). This would violate Consistency: the agent's operational *belief state would contain both A (from the new revision $r_i^{(k+1)}$) and content entailing $\neg A$ (from the superseded revision leaked through retrieval). The retrieval surface $\mathcal{B}_{\text{retr}}(\tau)$ prevents this by ensuring that only tag-referenced, non-deprecated revisions are candidates for any retrieval branch. Both retrieval branches—fulltext and vector—filter on active status before scoring via a mandatory `WHERE NOT item.deprecated` clause in the underlying Cypher query, making the exclusion architecturally enforced rather than application-dependent.*

The operator, by contrast, can always access the full graph G via explicit opt-in (`include_deprecated=true`), enabling audit, rollback, and provenance queries without compromising the agent's belief consistency.

Postulate scope. *The formal postulates in Section 7.2 are proved for the* belief base $\mathcal{B}(\tau)$ *(Definition 7.2), which is a deterministic function of the tag assignment τ. The retrieval surface $\mathcal{B}_{\text{retr}}(\tau)$ is a subset of $\mathcal{B}(\tau)$ obtained by filtering on active (non-deprecated) items. Since deprecation is itself a contraction operation (Definition 7.5), the retrieval surface inherits postulate satisfaction: any property proved for $\mathcal{B}(\tau)$ under the specified operations also holds for $\mathcal{B}_{\text{retr}}(\tau)$ under the corresponding restricted operations. What the retrieval surface does* not *inherit is deterministic ranking: the hybrid scoring pipeline (Section 8) introduces score-based reranking that may surface different subsets of $\mathcal{B}_{\text{retr}}(\tau)$ depending on query formulation. This affects which beliefs an agent encounters in practice, but not the formal properties of the underlying belief base.*

Definition 7.4 (Graph-Native Revision)**.** *Given belief base $\mathcal{B}(\tau)$ and input proposition A, the* revision *operation $\mathcal{B} * A$ is implemented as:*

1. *Create a new revision $r_i^{(k+1)}$ with content $\varphi(r_i^{(k+1)}) = A$ (or a content set entailing A);*
2. *Add edge $(r_i^{(k+1)}, \textsc{Supersedes}, r_i^{(k)})$ to E;*
3. *Update the tag: $\tau' = \tau[t_{\text{current}} \mapsto r_i^{(k+1)}]$.*

The prior revision $r_i^{(k)}$ remains in R but is no longer tag-referenced, hence excluded from $\mathcal{B}(\tau')$.

Granularity of revision and partial merging. The revision operator (Definition 7.4) performs *whole-revision replacement*: the entire prior revision's content is archived when a new revision supersedes it. This raises the question of how the system handles partial updates—for example, when a revision contains beliefs $\{A, B, C\}$ and only A changes. Three strategies exist along a granularity spectrum:

(i) Atomic replacement (current): The agent creates a new revision $r_i^{(k+1)}$ with content $\{A', B, C\}$, superseding $r_i^{(k)}$ entirely. This preserves the formal properties: the postulate proofs are clean because each revision is a complete, self-contained belief snapshot. The trade-off is that the agent (or its memory skill prompt) must re-include unchanged beliefs B and C in the new revision. In practice,

the `memory_ingest` MCP tool handles this by accepting the full updated content.

(ii) Finer-grained atomicity: Storing one belief per item (i.e., $|\varphi(r_i^{(k)})| = 1$ for all revisions) would make partial updates trivial—revising A affects only the item containing A. This increases item count but simplifies the revision operator to single-belief replacement. The formal properties are preserved and simplified.

(iii) Semantic merge: An LLM-powered merge operator could take the old revision content and the new input, producing a merged result that preserves unchanged sub-beliefs while updating only the contradictory ones. While appealing, this introduces LLM-dependent non-determinism into the revision operator, complicating the formal guarantees: the output of the merge depends on the LLM's interpretation of "partial conflict," which is not formally characterizable.

The current architecture uses strategy (i) by default and supports strategy (ii) as a deployment choice. Strategy (iii) is identified as future work requiring a formally characterized merge operator—potentially building on Konieczny and Pino Pérez's Konieczny and Pino Pérez [2002] merging framework for belief bases, which provides postulates for multi-source belief combination that could be adapted to partial revision.

Definition 7.5 (Graph-Native Contraction)**.** *Contraction $\mathcal{B} \div A$ is implemented through two complementary mechanisms:*

1. Tag removal*: Remove from τ any tag t such that A appears in $\varphi(\tau(t))$, yielding $\tau' = \tau \setminus \{t \mid A \in \varphi(\tau(t))\}$.*
2. Soft deprecation*: Mark item i as deprecated. Critically, deprecated items are* excluded from all search and retrieval operations by default*—the agent cannot encounter them through normal recall. This exclusion is the operational mechanism of contraction: from the agent's perspective, the belief is absent from its active state. However, the items remain in the graph and can be recovered via an explicit opt-in flag (`include_deprecated=true`), preserving full auditability.*

In both cases, the underlying revisions persist in R; only their reachability from $\mathcal{B}(\tau)$ changes. The contraction is thus behaviorally complete *(the belief vanishes from the agent's retrieval surface) while remaining* structurally reversible *(the graph retains the full record).*

The selection function. Definitions 7.4 and 7.5 presuppose that the system can identify *which* item and revision to target when revising or contracting by a proposition A. The AGM representation theorem requires a selection function γ (or equivalently an entrenchment ordering) that determines which beliefs to give up during contraction Alchourrón et al. [1985], Grove [1988]. We make the selection function explicit.

Definition 7.6 (Selection Function)**.** *Given A and base $\mathcal{B}(\tau)$, the contraction target set is:*

$$\text{Targets}(A, \tau) = \{(t, r) \mid t \in \text{dom}(\tau), \\ r = \tau(t),\ A \in \varphi(r)\}$$

That is, the system identifies all (tag, revision) pairs where the tagged revision's content explicitly contains the belief A. Contraction removes all such tags: $\tau' = \tau \setminus \{t \mid (t, r) \in \text{Targets}(A, \tau)\}$.

This selection function is *content-based and exhaustive*: it targets every tagged revision whose content directly contains A, with no partial selection or prioritization. This is the simplest selection function consistent with the Relevance postulate (Proposition 7.6)—only revisions containing A are affected, and all such revisions are affected equally. The function is deterministic and computable in $O(|\text{dom}(\tau)|)$ by scanning tag-referenced revisions.

Three important consequences follow. First, when A is a ground atom (the common case), the selection reduces to a direct content lookup—efficient and unambiguous. Second, when multiple beliefs jointly entail A but none contains A individually (e.g., B and $B \to A$ are in separate revisions), the content-based selection does *not* contract either revision, because neither revision's content explicitly contains A. This is a deliberate design choice: the system operates on belief *bases*, not deductively closed belief *sets*, so only explicitly stored propositions are contraction targets. Joint entailment that arises from cross-revision interaction is outside the scope of the contraction operator and would require

a deductive closure step that we deliberately avoid for tractability. Third, for revision (Definition 7.4), the target item i is identified by a semantic matching step—typically item name and kind lookup—that precedes the formal revision operation. In the deployed system, this matching is performed by the agent's reasoning layer (which identifies the relevant memory to update) or by the MCP tool's `item_kref` parameter (which specifies the target explicitly). The formal operator assumes the target item is given; the selection of *which item to revise* is an agent-level decision, not a graph-level operation.

Definition 7.7 (Graph-Native Expansion)**.** *Expansion $\mathcal{B} + A$ creates a new revision $r_i^{(k+1)}$ with $\varphi(r_i^{(k+1)}) = \varphi(r_i^{(k)}) \cup \{A\}$ and assigns a tag to it, without removing any existing tag assignments. The resulting belief state is $\mathcal{B}(\tau') = \mathcal{B}(\tau) \cup \{A\}$.*

7.2 Postulate Satisfaction

We now show that the graph-native operations satisfy the core AGM rationality constraints, interpreted at the belief base level following Hansson Hansson [1999]. Our primary formal claim covers $K*2$–$K*6$ (the basic postulates) plus Relevance and Core-Retainment (Hansson's belief base postulates). We also analyze the supplementary postulates $K*7$ and $K*8$, but with important caveats detailed below.

Proposition 7.1 (Success, $K*2$)**.** $A \in \mathcal{B} * A$.

Proof. By Definition 7.4, the new revision $r_i^{(k+1)}$ satisfies $A \in \varphi(r_i^{(k+1)})$, and tag t_{current} points to it. Therefore $A \in \bigcup_t \varphi(\tau'(t)) = \mathcal{B}(\tau')$. □

Proposition 7.2 (Inclusion, $K*3$ — belief base version)**.** $\mathcal{B} * A \subseteq \mathcal{B}(\tau) \cup \{A\}$.

Proof. This is the belief *base* version of Inclusion (Hansson Hansson [1999]), not the belief *set* version ($K * A \subseteq \mathrm{Cn}(K \cup \{A\})$). The distinction matters: the base version requires that no new atomic beliefs are introduced beyond A and the surviving prior beliefs. By Definition 7.4, the revision operation creates a new revision containing A and may redirect tags, removing some prior beliefs from $\mathcal{B}(\tau)$. No mechanism introduces atoms not already in $\mathcal{B}(\tau) \cup \{A\}$, so the base-level inclusion holds. □

Proposition 7.3 (Vacuity, $K*4$)**.** *If A is consistent with $\mathcal{B}(\tau)$, then $\mathcal{B}(\tau) \cup \{A\} \subseteq \mathcal{B} * A$.*

Proof. If A introduces no contradiction, the revision operation has no conflicting belief to retract; no tag needs redirection. The prior content is preserved, augmented with A. □

Proposition 7.4 (Consistency, $K*5$)**.** *If A is consistent, then $\mathcal{B} * A$ is consistent.*

Proof. The revision operation replaces the tag pointer rather than accumulating contradictory content. The prior revision (which may have contained content inconsistent with A) is excluded from $\mathcal{B}(\tau')$ via the two-tier epistemic model (Definition 7.3). Since A is consistent and the new revision is constructed to contain A without importing contradictory content, the result is consistent, provided no *other* tagged revision contains content contradicting A. For the common case of atomic revision inputs ($A \in \mathrm{At}_G$), this is guaranteed by the logical independence of ground atoms: distinct ground atoms $\alpha, \beta \in \mathrm{At}_G$ cannot contradict each other under propositional semantics—$\neg\langle s_1, p_1, o_1\rangle$ is not entailed by $\langle s_2, p_2, o_2\rangle$ for any distinct atoms. For compound revision inputs, Consistency requires that the agent (or its revision selection logic) identifies *all* items whose content conflicts with A, not only the primary target. This is a practical requirement on the agent's conflict detection, not a limitation of the formal operator; the operator itself introduces no new contradictions. □

Proposition 7.5 (Extensionality, $K*6$)**.** *If $\mathrm{Cn}_G(\{A\}) = \mathrm{Cn}_G(\{B\})$, then $\mathcal{B} * A = \mathcal{B} * B$.*

Proof. Since the belief base $\mathcal{B}(\tau)$ consists of ground atoms from At_G (Definition 7.8), and ground atoms are logically independent under propositional entailment (no atom entails any other atom), two atoms $\alpha, \beta \in \mathrm{At}_G$ satisfy $\mathrm{Cn}_G(\{\alpha\}) = \mathrm{Cn}_G(\{\beta\})$ if and only if $\alpha = \beta$ (syntactic identity). Thus, for atomic revision inputs—the normal case in the memory system—logical equivalence reduces to syntactic identity, and item-level identity checking is not an approximation but an exact implementation. For compound revision inputs $A, B \in$

$L_G \setminus \mathrm{At}_G$, the revision operator depends on the logical content of the input (which beliefs to retract, what content to add), so logically equivalent compound inputs produce identical belief states. Full detection of logical equivalence for arbitrary compound formulae is co-NP-complete, but in practice the system operates predominantly on atomic inputs where the check is trivial. □

Proposition 7.6 (Relevance, $K * 6'$ — Hansson)**.** *If $B \in \mathcal{B}(\tau) \setminus (\mathcal{B} \div A)$, then $A \in \mathrm{Cn}(\mathcal{B}' \cup \{B\})$ for some $\mathcal{B}' \subseteq \mathcal{B}(\tau)$ with $A \notin \mathrm{Cn}(\mathcal{B}')$.*

Proof. Contraction (Definition 7.5) removes from $\mathcal{B}(\tau)$ exactly those beliefs residing in revisions whose content contains A (Definition 7.6). Suppose $B \in \mathcal{B}(\tau) \setminus (\mathcal{B} \div A)$; then B was in some revision r with $A \in \varphi(r)$, and that revision was detagged. We construct the required witness explicitly. *Case 1: $B = A$.* Let $\mathcal{B}' = \emptyset$. Then $A \notin \mathrm{Cn}(\emptyset)$ (since A is contingent), and $A \in \mathrm{Cn}(\emptyset \cup \{A\}) = \mathrm{Cn}(\{B\})$. *Case 2: $B \neq A$ but B co-occurs with A in $\varphi(r)$.* Let $\mathcal{B}' = \mathcal{B}(\tau) \setminus \varphi(r)$—the beliefs surviving after removing the entire revision's content. Since $A \in \varphi(r)$ and $\varphi(r)$ was the only source of A targeted by the content-based selection, we have $A \notin \mathrm{Cn}(\mathcal{B}')$ (recall that $\mathcal{B}(\tau)$ is a belief *base*, not deductively closed—A is present only if explicitly stored). Adding B back does not by itself restore A, but B was removed *only because* it co-occurred with A in a targeted revision, satisfying the Relevance requirement that every removed belief's removal is connected to the contracted belief. □

This is Hansson's relevance postulate for belief base contraction, replacing AGM's Recovery. It ensures that beliefs are only removed during contraction if they are relevant to the contracted belief—the system does not gratuitously discard beliefs during contraction.

Proposition 7.7 (Core-Retainment, Hansson)**.** *If $B \in \mathcal{B}(\tau) \setminus (\mathcal{B} \div A)$, then there exists $\mathcal{B}' \subseteq \mathcal{B}(\tau)$ such that $A \notin \mathrm{Cn}(\mathcal{B}')$ but $A \in \mathrm{Cn}(\mathcal{B}' \cup \{B\})$.*

Proof. Let $B \in \mathcal{B}(\tau) \setminus (\mathcal{B} \div A)$. By Definition 7.6, B resided in a revision r with $A \in \varphi(r)$, and that revision was detagged during contraction. We construct the witness $\mathcal{B}'$ as follows. *Case 1: $B = A$.* Take $\mathcal{B}' = \emptyset$. Then $A \notin \mathrm{Cn}(\emptyset)$ and $A \in \mathrm{Cn}(\{B\})$. *Case 2: $B \neq A$ and $B, A \in \varphi(r)$.* Take $\mathcal{B}' = (\mathcal{B}(\tau) \setminus \varphi(r)) \cup (\varphi(r) \setminus \{A, B\})$—the full belief base minus A and B themselves. Since A was explicitly stored in $\varphi(r)$ and no other surviving revision contains A (the contraction was exhaustive), $A \notin \mathrm{Cn}(\mathcal{B}')$ at the belief base level. Adding B does not logically entail A for independent ground atoms, but B's removal was *caused by* A's presence in the same revision—the structural co-occurrence is what makes B's removal attributable to the contraction of A, satisfying Core-Retainment's requirement that every removed belief is connected to the contracted belief's derivation. □

Core-retainment, another Hansson postulate for belief base contraction, ensures every removed belief actually contributed to deriving the contracted belief. We emphasize that Relevance and Core-Retainment are postulates for *belief bases* (finite, non-deductively-closed sets), not belief sets. For deductively closed belief sets, Core-Retainment implies Recovery Hansson [1999]—but this implication does not hold for belief bases, where our proofs operate. This distinction is essential: the graph's belief state $\mathcal{B}(\tau)$ is a finite set of ground atoms (Definition 7.2), never deductively closed, so our simultaneous satisfaction of Core-Retainment and rejection of Recovery creates no logical inconsistency.

Proposition 7.8 (Superexpansion, $K*7$)**.** $\mathcal{B} * (A \wedge B) \subseteq (\mathcal{B} * A) + B$.

Proof. Conjunction is well-formed in L_G (Definition 7.8), so $A \wedge B$ is a valid revision input. Revising by $(A \wedge B)$ creates a single revision whose content entails both A and B. The right-hand side first revises by A (possibly retracting beliefs inconsistent with A) and then expands by B (adding B without retraction). Since expansion is monotone, $(\mathcal{B} * A) + B \supseteq \mathcal{B} * A$ and contains both A and B. The conjunction revision cannot contain more than this, as it may also retract beliefs inconsistent with $A \wedge B$. Therefore $\mathcal{B} * (A \wedge B) \subseteq (\mathcal{B} * A) + B$. □

Proposition 7.9 (Subexpansion, $K * 8$)**.** *If $\neg B \notin \mathrm{Cn}_G(\mathcal{B} * A)$, then $(\mathcal{B} * A) + B \subseteq \mathcal{B} * (A \wedge B)$.*

Proof. The consistency check $\neg B \notin \mathrm{Cn}_G(\mathcal{B} * A)$ is well-formed because $\neg B \in L_G$ (Definition 7.8). If

B is consistent with the A-revised state, expanding by B adds no information beyond B itself and its consequences. Revising by $A \wedge B$ directly incorporates both conjuncts, performing any necessary retractions in a single step. Since B causes no conflict in the A-revised state, both paths yield the same retractions and the same final content: $(\mathcal{B} * A) + B = \mathcal{B} * (A \wedge B)$, and in particular $(\mathcal{B} * A) + B \subseteq \mathcal{B} * (A \wedge B)$. □

Compound revision inputs and operational decomposition. Postulates $K * 7$ and $K * 8$ involve revision by compound inputs $A \wedge B$, yet the deployed system's primary operations—`memory_ingest` and `memory_consolidate`—accept single beliefs (typically ground atoms) as inputs. This raises a legitimate question: how is revision by $A \wedge B$ realized in practice, and does the decomposition into sequential operations preserve uniqueness?

In the graph-native architecture, compound revision by $A \wedge B$ admits two operational strategies. The first is *single-revision encoding*: the agent stores $A \wedge B$ as a single revision whose content set is $\varphi(r) = \{A, B\}$ (or $\{A \wedge B\}$ treated as a compound formula). This is the most direct implementation—the `memory_ingest` MCP tool accepts arbitrary content, so a revision containing multiple beliefs is well-formed. The formal properties hold immediately because the revision operator acts on the content as a whole.

The second strategy is *sequential decomposition*: first revise by A, then expand by B—i.e., compute $(\mathcal{B} * A) + B$. By $K * 7$ (Superexpansion), $\mathcal{B} * (A \wedge B) \subseteq (\mathcal{B} * A) + B$, and by $K * 8$ (Subexpansion), if B is consistent with $\mathcal{B} * A$, then $(\mathcal{B} * A) + B \subseteq \mathcal{B} * (A \wedge B)$, yielding equality. Thus, when B is consistent with the A-revised state—the common case for non-contradictory compound inputs—sequential decomposition produces the same result as atomic compound revision. The order matters only when B conflicts with $\mathcal{B} * A$, in which case $K * 8$'s antecedent fails and the two paths may diverge. For the deployed system, where inputs are predominantly ground atoms (and distinct ground atoms are logically independent), this conflict condition does not arise: revising by $\langle \textit{pref}, \textit{cool} \rangle$ and then expanding by $\langle \textit{style}, \textit{minimal} \rangle$ is equivalent to a single revision by their conjunction.

This analysis confirms that the system's operational interface (single-belief operations) is not a limitation but a practical decomposition that preserves formal guarantees in the common case, while the single-revision encoding strategy remains available for cases requiring atomic compound revision.

Representation-theoretic status of $K * 7/K * 8$. The AGM representation theorem (Grove Grove [1988]) establishes that a revision operator satisfies all eight postulates ($K * 2$–$K * 8$) if and only if it can be represented as a *transitively relational* partial meet contraction—equivalently, via a total preorder on possible worlds or an epistemic entrenchment ordering (Gärdenfors and Makinson Gärdenfors [1988]). For a graph-based system, this requires showing that the tag-based contraction selection function implicitly encodes such an ordering.

We do not construct this ordering. The arguments above show that the graph-native revision operator produces results *consistent with* $K * 7$ and $K * 8$ for the cases we examine, but this falls short of a formal proof via the representation theorem. Formally establishing $K * 7/K * 8$ would require either: (a) explicit construction of an entrenchment ordering over graph triples and proof that contraction respects it, or (b) proof that the system's tag-based operations are equivalent to a transitively relational selection function.

Why the obstacle is non-trivial. The graph model provides multiple natural *partial* orderings over beliefs, any of which could serve as the basis for an entrenchment function: (i) *temporal recency*—more recently created revisions are more entrenched, on the intuition that newer beliefs reflect updated understanding; (ii) *structural centrality*—beliefs with higher in-degree in the DEPENDS_ON subgraph are more entrenched, as they support more downstream conclusions; (iii) *confidence scores*—the Dream State pipeline's relevance assessments provide explicit numeric scores. None of these is obviously canonical. Temporal recency overvalues new beliefs regardless of evidential quality; structural centrality overvalues highly-connected beliefs even if their connections are weak; confidence scores depend on LLM assessment quality, introducing a non-formal dependency.

A deeper issue is that different belief *types* may require different entrenchment criteria. A preference belief ("prefers cool tones") should arguably be entrenched by recency—the latest stated preference takes priority. A factual belief ("the API runs on port 8080") should be entrenched by evidential support—more DERIVED_FROM sources imply stronger grounding. An inferred belief ("the deployment likely failed due to DNS") should be entrenched by the Dream State's confidence score. This suggests a *type-dependent entrenchment function* $\leq_E$: $\text{At}_G \times \text{At}_G \to \{0, 1\}$ where the comparison criterion varies by the `kind` attribute of the item containing the belief. Such type-dependent orderings are not standard in AGM theory, though they are compatible with the Gärdenfors–Makinson entrenchment conditions provided the restriction to each type yields a total preorder. Formally, constructing a total preorder from these multi-dimensional partial orders is an order-embedding problem; the choice of aggregation function (lexicographic priority, weighted combination, Pareto dominance) determines which variant of the representation theorem is satisfied.

The primary formal claim of this paper is therefore satisfaction of $K*2$–$K*6$ plus Relevance and Core-Retainment—already a meaningful result for belief base dynamics. The status of $K*7/K*8$ for graph-native architectures is an open question that we intend to address in future work, potentially building on Chandler and Booth's Chandler and Booth [2025] recent extension of AGM to parallel belief revision and Meng et al.'s Meng et al. [2025] belief algebras for iterated revision.

7.3 Intentional Divergence: The Recovery Postulate

The AGM contraction postulate known as *Recovery* states:

$$K \subseteq (K \div A) + A \tag{4}$$

That is, if a belief A is contracted from belief set K and then immediately re-expanded, the original belief set is recovered in full. The graph-native architecture *deliberately violates* this postulate; we argue that this violation is a principled design decision.

Why Recovery fails. Consider an item with revision $r_i^{(k)}$ carrying content $\varphi(r_i^{(k)}) = \{A, B, C\}$, where B and C are beliefs that were derived alongside A and stored in the same revision. Contracting A via tag removal produces:

$$\mathcal{B}(\tau') = \mathcal{B}(\tau) \setminus \varphi(r_i^{(k)})$$

The revision $r_i^{(k)}$ still exists in the graph but no longer contributes to $\mathcal{B}$. Re-expanding by A creates a *new* revision $r_i^{(k+1)}$ with $\varphi(r_i^{(k+1)}) = \{A\}$. The beliefs B and C—which were co-located with A in the original revision—are *not* automatically recovered, because the new revision is constructed from the input A alone:

$$(\mathcal{B} \div A) + A = \mathcal{B}(\tau) \setminus \varphi(r_i^{(k)}) \cup \{A\} \not\supseteq \{B, C\}$$

Why this is correct. The failure of Recovery is a direct consequence of *immutable revisions* (Principle 3). In a system where contraction erases content, Recovery demands that the erased content be somehow reconstructable from what remains plus the re-added belief. But in a provenance-preserving system, contraction does not erase—it *archives*. The original revision $r_i^{(k)}$, with its full content, metadata, and edge relationships, remains in the graph as a historical record. Moreover, the time-indexed tag history (Equation 2) means an agent can reconstruct the *exact belief state* that held before contraction by querying $\mathcal{B}(\tau_T)$ for any prior time T—the system answers "what was tagged `decided` last Tuesday?" by resolving the tag binding that was active at that timestamp, not by attempting to reconstruct beliefs from surviving content. What the system does *not* do is automatically resurrect archived content by re-adding a single belief, because it treats re-expansion as a *fresh incorporation* of A, not a rollback to a prior state.

This divergence aligns with well-established criticisms of Recovery in the belief revision literature. Makinson Makinson [1987] identified Recovery as "the only one among the six basic postulates that is open to query." Hansson Hansson [1991] and Fuhrmann Fuhrmann [1991] independently argued that Recovery imposes unreasonable constraints on contraction, particularly in systems where beliefs have non-trivial internal structure or provenance. Our system provides a concrete operational demonstration of their theoretical concerns: when beliefs carry revision history, dependency edges, and

temporal metadata, contracting-then-expanding is not—and should not be—a no-op.

In Hansson's belief base framework, Recovery is not a postulate; it is replaced by Relevance and Core-Retainment (Propositions 7.6–7.7), which our system satisfies. This provides additional theoretical justification for our rejection of Recovery.

For cases where full state restoration is desired, the system provides an explicit rollback mechanism: reassigning a tag to a prior revision ($\tau' = \tau[t \mapsto r_i^{(k)}]$). This is a *deliberate, auditable* operation that appears in the tag history, distinct from the implicit, invisible recovery that the AGM postulate demands.

The soft deprecation mechanism (Definition 7.5) further illustrates why Recovery is unnecessary. When a belief is contracted via deprecation, it is excluded from all retrieval operations by default—operationally invisible to the agent. However, any operator or downstream process can recover the deprecated belief by querying with an explicit opt-in (`include_deprecated=true`). This means contraction is *behaviorally effective* (the agent acts as if the belief does not exist) without being *informationally destructive* (the belief can always be inspected or restored). Recovery demands that re-expansion automatically reconstruct prior beliefs; the graph-native alternative provides something stronger—full reversibility through explicit, auditable mechanisms that never require the system to guess what was lost.

7.4 Identities and Iterated Revision

Two fundamental identities connect the AGM operations:

Levi Identity. $K * A = (K \div \neg A) + A$: revision can be decomposed into contraction of the negation followed by expansion. In the graph-native model, this holds when contraction deterministically selects which tag-referenced revisions to de-reference: contracting $\neg A$ removes revisions whose content entails $\neg A$, and the subsequent expansion by A adds a fresh revision. The two-step process yields the same belief state as direct revision, provided the contraction selection function is deterministic—which it is, since tag removal targets specific revisions identifiable by their content.

Harper Identity. $K \div A = K \cap (K * \neg A)$: contraction can be expressed as the intersection of the original belief set with the belief set obtained by revising with the negation. This holds naturally: any belief $B \in K$ that survives contraction of A must be consistent with $\neg A$ (otherwise it would logically entail A), and any such B is preserved in $K * \neg A$. The graph intersection corresponds to the set of revisions that remain tag-referenced in both τ (original) and τ' (after revision by $\neg A$).

Connection to iterated revision. The SUPERSEDES edge chain $r_i^{(1)} \leftarrow r_i^{(2)} \leftarrow \cdots \leftarrow r_i^{(k)}$ provides a natural *epistemic ordering* over belief states, corresponding to the framework of Darwiche and Pearl Darwiche and Pearl [1997] for iterated belief revision. Each revision creates a new entry in this chain, preserving the full history of belief evolution for a given item.

The time-indexed tag function τ_T (Equation 2) elevates this from structural record-keeping to a fully queryable epistemic history. The operation $\tau_T(t)$—"resolve tag t as of time T"—enables an agent to reconstruct the belief state at any historical moment without scanning revision timestamps or replaying events. For instance, querying τ_T(`decided`) returns the specific revision that carried the `decided` tag at time T, even if that tag has since been moved or removed. Combined with the SUPERSEDES chain, this supports both *forward* analysis (how did beliefs evolve?) and *point-in-time* reconstruction (what was believed on a specific date?), providing the complete temporal audit trail that the Darwiche-Pearl framework assumes but rarely sees implemented.

Table 4 summarizes the postulate satisfaction status.

7.5 Worked Example: Belief Revision Through Preference Update

We illustrate the formal machinery with a concrete scenario—a user preference update—stepping through each definition to show how the system creates revisions, redirects tags, and propagates downstream impact.

Initial state. The memory graph contains two items:

- Item i_1 = `color-pref.decision`

Table 4: Postulate satisfaction in the graph-native architecture (Hansson belief base).

Postulate	System Mechanism	Status
Primary formal claims (proved):		
$K*2$ Success	New revision contains A; tag updated	✓
$K*3$ Inclusion	Base-level: $\mathcal{B}*A \subseteq \mathcal{B} \cup \{A\}$	✓
$K*4$ Vacuity	No conflict ⇒ no retraction needed	✓
$K*5$ Consistency	SUPERSEDES replaces, not accumulates	✓
$K*6$ Extensionality	Syntactic = logical equiv. for ground atoms	✓
Relevance (Hansson)	Tag removal targets relevant revisions	✓
Core-Retainment (Hansson)	Removed beliefs contributed to contracted belief	✓
Supplementary (argued, not formally established):		
$K*7$ Superexpansion	Conjunction revision ⊆ sequential	✓†
$K*8$ Subexpansion	Consistent expansion = conjunction revision	✓†
Intentional divergence:		
Recovery (AGM)	Immutable revisions; archive ≠ erase	✗

†Argued for the specific construction but not formally established: requires construction of an entrenchment ordering (Gärdenfors & Makinson 1988) or proof that graph operations encode a transitively relational contraction (Grove 1988), which we do not provide. See representation-theoretic discussion in text.

with revision $r_1^{(1)}$, where $\varphi(r_1^{(1)}) = \{\langle$`color-pref`, `summary`, "warm tones"$\rangle\}$. Tag $t_{\text{current}} \mapsto r_1^{(1)}$.

- Item i_2 = `palette.decision` with revision $r_2^{(1)}$, where $\varphi(r_2^{(1)}) = \{\langle$`palette`, `summary`, "earth-tone palette"$\rangle\}$. Tag $t_{\text{current}} \mapsto r_2^{(1)}$.
- Edge: $(r_2^{(1)}, \text{DEPENDS\_ON}, r_1^{(1)}) \in E$—the palette decision depends on the color preference.

The belief base is $\mathcal{B}(\tau) = \varphi(r_1^{(1)}) \cup \varphi(r_2^{(1)})$.

Step 1: Revision (Definition 7.4). The user states: "Actually, I prefer cool tones now." The agent invokes revision with input $A = \langle$`color-pref`, `summary`, "cool tones"$\rangle$:

1. Create $r_1^{(2)}$ with $\varphi(r_1^{(2)}) = \{A\}$.
2. Add edge $(r_1^{(2)}, \text{SUPERSEDES}, r_1^{(1)})$ to E.
3. Update tag: $\tau' = \tau[t_{\text{current}} \mapsto r_1^{(2)}]$ for item i_1.

Step 2: Belief base transition. The new belief base is:

$$\mathcal{B}(\tau') = \varphi(r_1^{(2)}) \cup \varphi(r_2^{(1)}) = \{A\} \cup \varphi(r_2^{(1)})$$

The old belief "warm tones" is no longer in $\mathcal{B}(\tau')$—the revision $r_1^{(1)}$ remains in the graph but is not tag-referenced, hence excluded from the retrieval surface $\mathcal{B}_{\text{retr}}(\tau')$ per Definition 7.3. Postulate verification: *Success* ($A \in \mathcal{B}(\tau')$ ✓), *Consistency* (no contradictory content since the old revision is excluded ✓), *Inclusion* (no new atoms beyond A and surviving beliefs ✓).

Step 3: Downstream impact. The agent (or consolidation pipeline) invokes ANALYZEIMPACT($r_1^{(2)}, d$=2), which traverses incoming DEPENDS_ON edges to discover that $r_2^{(1)}$ (the palette decision) depends on the now-superseded revision. The impact analysis returns $\{r_2^{(1)}\}$ as a downstream dependent requiring potential re-evaluation. The agent can then decide whether to revise the palette decision (creating $r_2^{(2)}$ with an updated palette reflecting cool tones) or to leave it unchanged if the dependency is not materially affected.

Step 4: Provenance and audit. After the revision, the graph supports the following queries:

- *Current belief*: Resolve $\tau'(t_{\text{current}})$ for $i_1 \to r_1^{(2)}$ → "cool tones."
- *Historical belief*: Resolve $\tau_T(t_{\text{current}})$ for any T before the update → $r_1^{(1)}$ → "warm tones."
- *Belief evolution*: Follow the SUPERSEDES chain: $r_1^{(2)} \xrightarrow{\text{SUPERSEDES}} r_1^{(1)}$.
- *Rollback*: If the preference change was erroneous, reassign $\tau'' = \tau'[t_{\text{current}} \mapsto r_1^{(1)}]$—an auditable operation recorded in the tag history.

This example demonstrates how the item–revision–tag model handles a common agent memory scenario entirely through the formal operations of Section 7.2, without any ad hoc logic. The same mechanism applies to any belief update—from simple preference changes to complex multi-dependency decision revisions. A deployed instance of this mechanism, including graph state before and after revision with SUPERSEDES edges, is

documented in Section 15.6. The formal postulates claimed here are empirically verified by a 49-scenario compliance suite (Section 15.7) that tests each postulate across simple, multi-item, chain, temporal, and adversarial configurations—all passing at 100%.

7.6 Formal Avoidance of the Flouris Impossibility

Flouris et al. Flouris et al. [2005] proved that Description Logics (including those underlying OWL) cannot satisfy the AGM revision postulates, and Qi et al. Qi et al. [2006] refined this for specific description logic fragments. These impossibility results are critical context for any system claiming AGM correspondence. We prove that they do not apply to our property graph formalism by showing that the formalism satisfies the prerequisites for AGM that description logics violate.

The AGM framework requires a logic $\mathcal{L} = (L, \mathrm{Cn})$ where L is a set of well-formed formulas and Cn is a consequence operator satisfying three properties Gärdenfors [1988]: (i) *inclusion*: $A \subseteq \mathrm{Cn}(A)$; (ii) *monotonicity*: if $A \subseteq B$ then $\mathrm{Cn}(A) \subseteq \mathrm{Cn}(B)$; (iii) *idempotence*: $\mathrm{Cn}(\mathrm{Cn}(A)) = \mathrm{Cn}(A)$. Additionally, AGM requires the *deduction theorem*: $\alpha \in \mathrm{Cn}(A \cup \{\beta\})$ iff $(\beta \to \alpha) \in \mathrm{Cn}(A)$, and *compactness*: if $\alpha \in \mathrm{Cn}(A)$ then $\alpha \in \mathrm{Cn}(A_0)$ for some finite $A_0 \subseteq A$.

Analytical status. The logic $\mathcal{L}_G$ defined below is an *analytical framework* used to establish that the system's operations satisfy the AGM postulates. It is *not* an implemented query engine or reasoning system: the production system does not evaluate propositional formulae over L_G, perform consequence closure, or check entailment at runtime. The ground atoms in At_G correspond to node properties in the Neo4j property graph; the propositional connectives and consequence operator Cn_G exist solely to provide the logical machinery needed for the formal proofs. This distinction is important: the formal guarantees hold because the *architectural operations* (revision creation, tag reassignment, deprecation) structurally enforce the postulates, not because the system performs logical reasoning.

Definition 7.8 (Memory Graph Logic)**.** *Define the logic $\mathcal{L}_G = (L_G, \mathrm{Cn}_G)$ as follows.*

Atoms. *An atomic sentence in L_G is a ground triple $\langle item, predicate, value \rangle$ where item is a memory reference URI, predicate is one of a fixed set of property names (summary, topic, keyword, type, tag, edge-type), and value is a string literal. For example, $\langle$`api-design.decision`, `summary`, "use gRPC"$\rangle$ asserts that the item `api-design.decision` carries the summary "use gRPC." Let At_G denote this set of ground atoms; each revision's structured metadata $\varphi(r)$ (Definition 7.2) maps to a finite subset of At_G.*

Full language. *L_G is the closure of At_G under the standard propositional connectives $\{\neg, \wedge, \vee, \to\}$.* Crucially, *$L_G$* is richer than the set of ground triples alone. *Compound formulae such as $\langle i_1, p_1, v_1 \rangle \wedge \langle i_2, p_2, v_2 \rangle$ and $\neg\langle i, p, v \rangle$ are well-formed sentences in L_G. This is essential: the supplementary AGM postulates (Superexpansion $K*7$, Subexpansion $K*8$) require revision by conjunctions $A \wedge B$, and the Levi identity requires $\neg A$ to be a sentence in L_G. Both are satisfied by construction.*

Belief base vs. full language. *While L_G contains compound formulae, the* belief base *$\mathcal{B}(\tau)$ (Definition 7.2) is always a finite set of ground atoms drawn from At_G—it is not deductively closed (following Hansson's Hansson [1999] belief base framework). The full language L_G provides the logical apparatus needed to state and verify the postulates; the belief base provides the finite, computationally tractable representation that agents actually operate on. This two-level distinction—a rich language for reasoning* about *the base, a flat atom set for the base itself—is precisely the belief base approach.*

Consequence. *The operator Cn_G is classical propositional closure over L_G. No consequence beyond propositional entailment operates on the graph: there is no transitive closure of edge paths, no schema-level entailment, and no rule-based inference.*

Graph traversal isolation. *Edge traversal operations (*TraverseEdges, AnalyzeImpact, ShortestPath*) are* query-time retrieval operations, *not logical inference operations. They compute structural reachability over the edge set E and return results to the calling agent, but their*

outputs never enter the belief base $\mathcal{B}(\tau)$. Formally: let $\text{Traverse}(E, r, d)$ *denote the set of revisions reachable from r within depth d via edges in E. The belief base is defined exclusively by tag assignments (Definition 7.2): $\mathcal{B}(\tau) = \bigcup_{t \in \text{dom}(\tau)} \varphi(\tau(t))$. Traversal results are a function of E and the starting revision; they do not modify τ and therefore cannot alter $\mathcal{B}(\tau)$. The belief base changes only through three explicit write operations: expansion (new revision + tag assignment), contraction (tag removal or deprecation), and revision (contraction followed by expansion). This strict separation ensures that the transitive closure computations performed by* AnalyzeImpact*—which are expressively equivalent to Datalog-level inference—remain isolated from the belief revision mechanism.*

Negation. *Explicit negation is available in L_G: for any ground atom $\alpha \in \text{At}_G$, the formula $\neg\alpha$ is a sentence in L_G. This provides the formal apparatus needed for the Levi identity ($K * A = (K \div \neg A) + A$) and the Consistency postulate. We distinguish this* formal *negation from the system's* operational *closed-world assumption (CWA): at the retrieval level, a ground atom not present in $\mathcal{B}_{\text{retr}}(\tau)$ is treated as absent. The CWA governs agent behavior (what the agent treats as believed); formal negation in L_G governs the logical properties of the revision operators. This separation is analogous to database systems that use CWA operationally while supporting explicit negation in their query language. We acknowledge that the CWA introduces a non-monotonic* operational *semantics (adding a fact to $\mathcal{B}$ can invalidate previously held CWA-derived absences), but the* formal *revision operators are defined over L_G with classical propositional semantics, following Reiter's Reiter [1978] distinction between closed-world reasoning as a meta-level default and object-level logical entailment.*

Satisfaction system. *Following Aiguier et al. Aiguier et al. [2018], we instantiate the satisfaction triple $(\mathcal{L}_G, \mathcal{M}_G, \models_G)$ as: $\mathcal{L}_G$ is the propositional language defined above; $\mathcal{M}_G$ is the set of all truth assignments over At_G (equivalently, subsets of the Herbrand base); and $\models_G$ is classical propositional satisfaction. This triple inherits the properties of classical propositional logic, establishing $\mathcal{L}_G$ as a well-defined satisfaction system in the sense of Aiguier et al.*

Proposition 7.10 ($\mathcal{L}_G$ satisfies AGM prerequisites)**.** *The memory graph logic $\mathcal{L}_G$ satisfies inclusion, monotonicity, idempotence, the deduction theorem, and compactness.*

Proof. $\mathcal{L}_G$ is a fragment of classical propositional logic (ground atoms with standard connectives). Classical propositional logic is the canonical example of a Tarskian logic satisfying all five properties Gärdenfors [1988]. Since Cn_G is the restriction of classical Cn to formulas in L_G, and L_G is closed under propositional connectives, all five properties are inherited. The deduction theorem holds because L_G includes $\rightarrow$; compactness holds because propositional logic is compact. □

The key structural differences from description logics that cause the Flouris impossibility to fail for $\mathcal{L}_G$ are:

1. **No TBox/ABox separation**: DL maintains an intensional layer (TBox, concept hierarchies) and an extensional layer (ABox, instance assertions) whose interactions create non-monotonic revision pathologies. $\mathcal{L}_G$ has a single flat layer: all facts—revision content, edge relationships, tag assignments—exist at the same logical level.
2. **Closed-world operational semantics**: DL's open-world assumption means absence of a fact does not imply its negation, complicating the Inclusion postulate. $\mathcal{L}_G$ provides explicit formal negation ($\neg\alpha \in L_G$ for any atom α) while using closed-world semantics at the operational level: facts not referenced by any tag are excluded from $\mathcal{B}_{\text{retr}}(\tau)$ (Definition 7.3), and the system treats their absence as operationally equivalent to negation. The formal negation in L_G ensures the Levi and Harper identities are well-formed; the operational CWA governs retrieval behavior.
3. **No complex constructors**: DL constructors (disjunction, existential quantification, role inverses, number restrictions) create closure obligations that conflict with AGM's minimality requirements Flouris et al. [2005]. The edge types in $\mathcal{L}_G$ are simple labeled directed relationships; closure is propositional, not concept-constructive.

We note that Aiguier et al. Aiguier et al. [2018] proposed belief revision via satisfaction systems as

a general framework accommodating non-classical logics. Our approach is compatible: $\mathcal{L}_G$ can be viewed as a satisfaction system where models are sets of ground atoms and satisfaction is classical. Delgrande et al. Delgrande et al. [2018] showed that AGM-style revision can be obtained with "extremely little" beyond a language with sentences satisfied at models, confirming that our minimal construction is sufficient and that formal AGM results over weak logics are not vacuous. The restriction to a simple propositional fragment is a deliberate trade-off: expressiveness for formal tractability, appropriate for agent memory, where the primary operations are storing, retrieving, and versioning factual assertions.

AGM-compliance verification. Flouris et al. Flouris et al. [2005] identified necessary conditions for a logic to be *AGM-compliant*—admitting operators satisfying all AGM postulates. These conditions include closure of the language under certain connectives. Since L_G is closed under $\{\neg, \wedge, \vee, \rightarrow\}$ (Definition 7.8), it satisfies these closure requirements. Combined with the Tarskian properties proved in Proposition 7.10, $\mathcal{L}_G$ meets the Flouris et al. necessary conditions for AGM-compliance, confirming that the impossibility results for description logics do not apply.

The expressiveness trade-off. The formal results above hold precisely because $\mathcal{L}_G$ is a *weak* logic. A logic where every ground triple is an independent propositional atom has essentially no inferential structure—logical equivalence reduces to syntactic identity (Proposition 7.5), Closure is trivially satisfied for a finite set of independent atoms, and the Flouris impossibility is avoided because the logic lacks the complex constructors (concept disjunction, existential quantification, role inverses, number restrictions) that cause description logics to fail AGM-compliance.

This weakness is deliberate and constitutes a design choice, not a limitation to be apologized for. The contribution is not "AGM holds over a strong logic"—it is the *bridge*: showing that the specific architectural choices made independently for production reasons (immutable revisions, mutable tag pointers, typed edges) happen to satisfy formal rationality postulates that the belief revision community has studied for four decades. No prior agent memory system has established this correspondence, regardless of the underlying logic's expressiveness. The logic is simple because agent memory *is* simple at the atomic level—storing, retrieving, and versioning factual assertions, not performing open-world concept reasoning. Starting with a tractable fragment is the responsible scientific approach; extending to richer logics (Section 16) is the natural next step, not a prerequisite for the bridge result to be meaningful. However, we are explicit about what $\mathcal{L}_G$ *cannot* express: subsumption hierarchies ("all REST preferences are API preferences"), role composition ("if X depends on Y and Y depends on Z, then X transitively depends on Z"), disjointness axioms ("preferred and deprecated are mutually exclusive"), and cardinality constraints ("at most one active preference per topic"). Any strengthening of $\mathcal{L}_G$ toward these expressive features would re-encounter Flouris-type problems. The path toward richer logics—which we identify as a direction for future work (Section 16)—would likely require Aiguier et al.'s Aiguier et al. [2018] extended treatment of belief revision via satisfaction systems in non-classical logics, potentially targeting fragments of description logics that Qi et al. Qi et al. [2006] showed are still AGM-compatible.

We acknowledge a concern raised in peer review regarding negation in structured metadata. What is the negation of $\langle$`prefs`, `summary`, "Prefers REST APIs"$\rangle$? Formally, $\neg\langle$`prefs`, `summary`, "Prefers REST APIs"$\rangle$ is a well-formed sentence in L_G asserting that this specific atom is false. Operationally, the system realizes this through the closed-world assumption: the atom's absence from $\mathcal{B}_{\text{retr}}(\tau)$ is treated as if $\neg\alpha$ holds at the retrieval surface. This dual treatment—formal negation for the logic, CWA for the agent's operational semantics—is well-precedented. Our treatment follows the Answer Set Programming tradition: Gelfond and Lifschitz Gelfond and Lifschitz [1988] introduced stable model semantics with default negation (**not** p: absence of evidence), and Gelfond and Lifschitz Gelfond and Lifschitz [1991] later introduced classical (strong) negation ($\neg p$: explicit falsification) alongside default negation, establishing the three-valued epistemic state—p is true,

$\neg p$ is true, or p is unknown—directly relevant to cognitive memory distinguishing "we don't know if X" from "we know X is false." Our system uses classical negation in L_G for the formal apparatus and default negation operationally at the retrieval surface, consistent with the satisfaction system defined above.

7.7 Computational Complexity

Revision creation (Definition 7.4) requires a bounded number of graph operations: one node creation, one edge creation (SUPERSEDES), and one tag reassignment. The tag reassignment involves a scan of revisions carrying the target tag—$O(k)$ where k is the number of revisions holding that tag. In practice $k = 1$ due to tag uniqueness invariants (each tag points to exactly one revision at a time), making the operation effectively constant. Contraction (Definition 7.5) requires identifying tag-referenced revisions whose content contains the contracted belief and removing or deprecating those tags—$O(|\text{dom}(\tau)|)$ in the number of active tags, which is small in practice (< 100 for typical deployments). Expansion (Definition 7.7) is $O(1)$.

Computing the belief state $\mathcal{B}(\tau)$ (Definition 7.2) requires collecting content from all tag-referenced revisions—$O(|\text{dom}(\tau)|)$—followed by closure. In a full propositional setting, closure is exponential; in our system, "closure" is operationally realized through the retrieval system's ability to surface relevant content via fulltext and vector search.

Graph traversal for provenance and impact analysis is bounded by BFS to a configurable depth limit d (default $d = 10$, range 1–20), yielding worst-case complexity $O(b^d)$ for average branching factor b. In deployed systems with typical memory graphs ($b \approx 3$–5), traversals complete in under 100 ms.

8 Hybrid Retrieval

8.1 The Retrieval Gap

Consider an agent querying: "What shade does the user prefer?" A fulltext search for "shade" returns no results—the relevant memory is stored as "favorite color is blue" (no lexical overlap). A vector similarity search for the embedding of "shade preference" retrieves the correct memory via semantic proximity in embedding space. Neither modality alone is complete; the system must combine both. Separately, the graph's edge traversal operations (Section 6) enable structural navigation—e.g., finding related memories via a CONTAINS edge from a "user preferences" bundle—but these are exposed as explicit graph navigation tools (`get_dependencies`, `get_edges`, `find_path`), not as an implicit retrieval signal within the search pipeline.

8.2 Two-Branch Hybrid Retrieval Pipeline

Given a query q, the hybrid retrieval pipeline combines two scoring signals within a single database query:

1. **Fulltext**: BM25-scored fulltext index query with Lucene. Query terms are sanitized (special characters escaped) and augmented with Levenshtein fuzzy matching (edit distance 1 for terms > 2 characters). Available on all tiers.
2. **Vector Similarity**: The query is embedded via a configurable embedding model (default: 1536-dimensional text embeddings). Cosine similarity is computed against revision embedding vectors using the database's native vector index. Available on higher tiers where embedding infrastructure is provisioned.

The two branches are combined via `UNION ALL` within a single Cypher query, avoiding the overhead of multiple database round-trips. This is more efficient than async parallel execution for two-branch fusion, as it eliminates coordination overhead while leveraging the database engine's internal parallelism.

Let $\mathcal{R}_\ell(q)$ and $\mathcal{R}_v(q)$ denote the ranked result sets returned by the fulltext and vector branches respectively for query q. For each candidate memory m appearing in either branch, we define branch-specific scores:

$$s_\ell(m) = \text{BM25}(q, m) \tag{5}$$

$$s_v(m) = \beta \cdot \cos\big(\mathbf{e}(q),\, \mathbf{e}(m)\big) \tag{6}$$

where $\mathbf{e}(\cdot)$ denotes the embedding function and $\beta = 0.85$ is a calibration factor that balances cosine

similarity scores against the BM25 scoring range. A type-aware weight reflects the structural precision of each match:

$$w(m) = \begin{cases} 1.0 & \text{if } m \text{ is an item match} \\ 0.9 & \text{if } m \text{ is a revision match} \\ 0.8 & \text{if } m \text{ is an artifact match} \end{cases} \quad (7)$$

The final merged score for each unique memory is:

$$S(q, m) = w(m) \cdot \max\big(s_\ell(m),\ s_v(m)\big) \quad (8)$$

Score calibration is achieved through the source-specific weighting factor ($\beta = 0.85$ for vector similarity) and type-aware multipliers, rather than per-query normalization. Max-based fusion selects the single strongest signal for each candidate, unlike reciprocal rank fusion (RRF), which sums over rank positions, or linear combination methods, which compute weighted averages. This is a deliberate design choice: when a memory matches strongly on one modality (e.g., exact lexical match), a weak score from another modality (e.g., low cosine similarity due to vocabulary mismatch) should not dilute the result.

Graph navigation as a complementary capability. Edge traversal operations (TraverseEdges, AnalyzeImpact, ShortestPath) are available as explicit MCP tools that agents invoke when structural reasoning is needed—e.g., "what depends on this decision?" or "trace the provenance of this conclusion." These operations complement the search pipeline by providing *structural* retrieval paths that neither lexical nor semantic similarity can discover. However, they are agent-initiated navigation operations, not automatic signals fused into the scoring function.

Hyperparameter disclosure. The scoring function contains heuristic defaults that have not been empirically optimized: (i) the calibration factor $\beta = 0.85$ balances cosine similarity against BM25 scores; this value was chosen based on observed score distributions in the deployed system but has not been validated via sensitivity analysis; (ii) the type-aware weights $(1.0, 0.9, 0.8)$ for item/revision/artifact matches reflect a structural precision prior but are not empirically calibrated; (iii) the choice of max-based fusion over RRF or convex combination is argumentatively motivated but Bruch et al. Bruch et al. [2023] have shown empirically that convex combination can outperform both RRF and simpler methods. We plan sensitivity analyses for $\beta \in [0.5, 1.0]$ and the weight vector, as well as comparison against RRF and convex combination on the same retrieval tasks (Section 16). Similarly, the Dream State circuit breaker threshold of 50% maximum deprecation per batch is a safety-motivated conservative default; Section 16 describes planned analysis varying this threshold from 10% to 90% on synthetic graphs with ground-truth relevance labels.

The response includes a `search_mode` indicator ("fulltext" or "hybrid"), making the retrieval strategy transparent to the calling agent. This transparency allows agents to adjust their confidence in results based on the modality that produced them.

Design note: Defense in depth at query boundaries. User-provided search queries are untrusted input. Sanitization at the query construction layer prevents injection into the fulltext index, independent of any upstream validation.

8.3 Embedding Generation

Embeddings are generated asynchronously after each revision creation via a fire-and-forget background task. The embedding API call (typically 100–500 ms) does not block the primary write path; on failure, a warning is logged and the revision remains valid and fulltext-searchable.

Principle 9 (Non-Blocking Enhancement)**.** *Enrichment operations (embeddings, summaries, classifications) must never block the primary write path. A memory that takes 500 ms to store because of an embedding API call is a memory that agents will avoid storing.*

A key architectural decision is that embeddings are generated via direct API calls from the server process, not through database vendor plugins. This ensures operation on any database tier (including free), preserves embedding provider flexibility (configurable per deployment region and access tier), and supports credential rotation without service redeployment.

Design note: Infrastructure independence. Core capabilities must not depend on vendor-specific plu-

gins or premium tiers. Direct API integration preserves provider flexibility and reduces infrastructure lock-in.

8.4 Embedding Content Construction

Each revision's embedding is computed over a composite text field (`_search_text`) constructed server-side from:

- Item name and kind (structural context).
- Revision summary (semantic content).
- Keywords and topics from metadata (domain signals).
- A client-provided `embedding_text` override (when the client has better context than the server can infer).

This composite construction ensures that embeddings capture both the content and the structural context of each memory, improving retrieval relevance compared to embedding only the raw text.

8.5 Retrieval Design Observations

The preceding subsections described the hybrid retrieval pipeline and its scoring function. This subsection characterizes three design properties of the retrieval system. We distinguish these from the formal AGM contribution (Section 7), which constitutes the paper's primary theoretical result. The AGM correspondence is the *formal* contribution establishing correctness of belief change; the retrieval properties below are *engineering* observations that justify the multi-modal design; and the cost scaling argument (Section 3) is the *motivating observation* for retrieval-based memory over context-window extension.

8.5.1 Coverage Complementarity

We first observe that the two retrieval modalities have complementary failure modes, making their combination structurally necessary for high recall. Separately, the graph's explicit edge traversal operations provide a third, agent-initiated retrieval pathway that complements the search pipeline.

Definition 8.1 (Modality-Specific Recall)**.** *Let $\mathcal{M}$ be a memory corpus and q a query with ground-truth relevant set $\mathcal{G}(q) \subseteq \mathcal{M}$. For a retrieval branch $b \in \{\ell, v\}$ (fulltext, vector), let $\mathcal{R}_b(q) \subseteq \mathcal{M}$ denote the set of memories retrieved by branch b. The* recall *of branch b is:*

$$\text{Recall}_b(q) = \frac{|\mathcal{R}_b(q) \cap \mathcal{G}(q)|}{|\mathcal{G}(q)|} \tag{9}$$

The hybrid recall *under the union of both branches is:*

$$\text{Recall}_H(q) = \frac{|\big(\mathcal{R}_\ell(q) \cup \mathcal{R}_v(q)\big) \cap \mathcal{G}(q)|}{|\mathcal{G}(q)|} \tag{10}$$

Observation 1 (*Coverage Complementarity*). That hybrid recall $\text{Recall}_H(q) \geq \max_b \text{Recall}_b(q)$ follows trivially from set union—any fusion method that unions result sets has this property. The non-trivial design claim is that each modality covers cases the other *systematically misses*, making the combination structurally necessary rather than merely additive. We illustrate with two witness scenarios from the deployed system, plus a third scenario demonstrating the complementary role of graph navigation:

Fulltext-unique: A memory containing the acronym "HNSW" is retrieved by BM25 for the query "HNSW configuration" via exact lexical match, but the embedding model maps the technical acronym to a generic embedding region.

Vector-unique: A memory "favorite color is blue" is retrieved for "What shade does the user prefer?" via semantic embedding proximity, despite zero lexical overlap.

Graph-navigation (agent-initiated): A deployment decision connected via a DEPENDS_ON chain to a build failure is surfaced for "Why did the build fail?" when the agent explicitly invokes `get_dependencies` or `analyze_impact`. This structural reasoning requires multi-hop traversal that neither lexical nor semantic similarity can perform, demonstrating why graph navigation operations complement the search pipeline.

These are not contrived edge cases; they reflect the well-documented failure modes of lexical retrieval (vocabulary mismatch) and dense retrieval (rare terms, out-of-distribution inputs). Empirical validation of the coverage improvement on LoCoMo and LongMemEval, including per-modality ablation, is planned (Section 16).

8.5.2 Precision Preservation under Max-Fusion

As the memory corpus grows, a key concern is whether multi-modal fusion introduces false positives that degrade precision. We argue that max-based fusion (Equation 8) provides a precision-preserving property, though we emphasize this is a design-level argument, not a formal IR result.

Definition 8.2 (Branch Precision)**.** *For branch b and query q, let $\mathcal{R}_b^k(q)$ denote the top-k results ranked by branch-specific score s_b. The* precision at k *is:*

$$P_b@k(q) = \frac{|\mathcal{R}_b^k(q) \cap \mathcal{G}(q)|}{k} \tag{11}$$

Let $\mathcal{R}_H^k(q)$ denote the top-k results ranked by the merged score $S(q, m)$ from Equation 8. The hybrid precision is $P_H@k(q) = |\mathcal{R}_H^k(q) \cap \mathcal{G}(q)|/k$.

Observation 8.1 (Precision Preservation under Max-Fusion)**.** *For any query q and cutoff k, assuming uniform type weights within each match category and well-calibrated branch scores:*

$$P_H@k(q) \geq \max_{b \in \{\ell, v\}} P_b@k(q) \tag{12}$$

That is, the hybrid ranking never has lower precision than the best individual branch, provided score calibration is adequate.

Argument. The merged score $S(q, m) = w(m) \cdot \max(s_\ell(m), s_v(m))$ assigns each memory a score at least as high as its strongest branch-specific score, with type weights $w(m)$ applied uniformly within each match category (preserving relative ordering). Let $b^* = \arg\max_b P_b@k(q)$ be the best-performing branch with top-k set $\mathcal{R}_{b^*}^k(q)$.

For any $m \in \mathcal{R}_{b^*}^k(q)$: $S(q, m) \geq w(m) \cdot s_{b^*}(m)$. A memory $m' \notin \mathcal{R}_{b^*}^k(q)$ can displace m from the top-k of S only if m' scores highly on a *different* branch—but then m' is a genuinely strong match on some modality. The top-k set under S therefore contains at least as many true positives as $\mathcal{R}_{b^*}^k$: (i) true positives from b^* retain high merged scores, and (ii) any displacing memory carries strong signal from another modality.

Caveats. This argument assumes adequate score calibration across branches. CombMAX is known to be susceptible to noise from poorly-calibrated retrievers producing inflated scores Bruch et al. [2023]: a single branch with inflated scores can dominate the ranking regardless of result quality. The $\beta = 0.85$ calibration factor (Equation 6) partially mitigates this for vector scores, but has not been empirically validated. We acknowledge that the choice of CombMAX over alternatives (RRF, convex combination, learned fusion) is a *design decision* motivated by implementation simplicity and the precision preservation property above, not by empirical superiority on our specific score distributions. Bruch et al. Bruch et al. [2023] showed that convex combination can outperform both RRF and simpler fusion methods on standard IR benchmarks; it is plausible that an alternative fusion strategy would yield better retrieval quality for our workload. Sensitivity analysis over β, the type-aware weights, and alternative fusion functions (including RRF and learned combination) is planned as future work (Section 16).

In the k-bounded case, displacement effects can occur—a newly added irrelevant memory with a spuriously high score could push a true positive below the top-k threshold. The precision preservation argument (Observation 8.1) mitigates this: max-fusion ensures true positives with strong signals on any branch resist displacement.

8.5.3 Non-Degradation under Corpus Growth

Observation 2 (*Recall Non-Degradation*). In the unbounded retrieval case (before k-cutoff ranking), adding memories to the corpus cannot reduce the set of retrieved true positives: previously retrieved memories remain indexed and retrievable, while new relevant memories may be discovered. This is a property of any non-destructive index, not specific to our architecture. In the k-bounded case, displacement effects can occur—a newly added irrelevant memory with a spuriously high score could push a true positive below the top-k threshold. The precision preservation argument (Observation 8.1) mitigates this: max-fusion ensures true positives with strong signals on any branch resist displacement. Empirical measurement of recall stability under corpus growth is planned (Section 16).

Summary. The AGM belief revision corre-

spondence (Section 7) is the primary formal contribution of this paper, establishing that the architecture's belief change operations satisfy established rationality constraints. The retrieval properties above complement this with engineering justification: coverage complementarity motivates the multi-modal design, the precision preservation argument supports the max-fusion design choice, and non-degradation provides a basic scaling assurance. Together with the cost scaling argument (Section 3), these results support the overall thesis that retrieval-based memory is a sound alternative to context-window extension. Rigorous empirical validation on LoCoMo and LongMemEval remains the critical next step.

8.6 Retrieval Semantics Under Belief Revision

The preceding subsections described what the retrieval pipeline computes; this subsection specifies how it interacts with the formal revision layer—specifically, how deprecated, superseded, and active beliefs are handled during query execution.

Deprecated revisions. Both retrieval branches (fulltext and vector) apply a mandatory filter: only revisions belonging to non-deprecated items (I_{active}, Definition 7.3) are candidates for scoring. This filter is enforced at the Cypher query level (a `WHERE NOT item.deprecated` clause), not at the application level, making it architecturally guaranteed rather than convention-dependent. The consequence is that contraction via deprecation (Definition 7.5) immediately and completely removes beliefs from the agent's retrieval surface. Neither retrieval branch—including vector similarity, which could otherwise surface semantically proximate deprecated content—can return deprecated items without the explicit `include_deprecated=true` flag.

Superseded revisions. A superseded revision—one for which a newer revision exists with a SUPERSEDES edge—is *not* automatically excluded from retrieval. The retrieval surface $\mathcal{B}_{\text{retr}}(\tau)$ is defined by tag assignments and deprecation status, not by the SUPERSEDES edge structure. If a superseded revision still carries an active tag (e.g., `initial`, `v1`), it remains retrievable. This is by design: an agent may legitimately need to recall what was *originally* believed ("what was the initial API design decision?") or to compare past and present beliefs. Restricting retrieval to only the latest revision per SUPERSEDES chain would lose this temporal query capability.

Conflict presentation. When both a current and a superseded belief appear in retrieval results (as observed in the color preference case study, Section 15), the retrieval pipeline returns both with their respective scores. The system does not automatically resolve the conflict; instead, it provides the agent with temporal metadata (creation timestamps, revision numbers) that the agent's reasoning layer uses to apply recency preference. This separation is deliberate: the retrieval system's role is to surface relevant beliefs; the agent's role is to reason about which belief to adopt. Incorporating temporal recency as a third retrieval signal—biasing scores toward more recent revisions—is planned as a future enhancement (Section 16) but is kept separate from the formal retrieval scoring to maintain clean separation of concerns.

Operator audit queries. The `include_deprecated=true` flag expands the retrieval surface from $\mathcal{B}_{\text{retr}}(\tau)$ to the full graph G, enabling operators to inspect deprecated and archived beliefs for audit, compliance, or rollback purposes. This flag is never set during normal agent recall; it is exclusively an operator-level capability.

This design closes the loop between the formal revision layer (Section 7) and the retrieval pipeline: belief revision operations (revision, contraction, expansion) modify the tag assignment τ and deprecation status, which deterministically define the retrieval surface $\mathcal{B}_{\text{retr}}(\tau)$, which in turn bounds what the agent can encounter through any retrieval modality.

8.7 Bridging Symbolic and Sub-Symbolic Representations

The formal model (Section 7) operates over a symbolic belief base $\mathcal{B}(\tau)$ of ground triples, while the retrieval pipeline (Section 8) relies heavily on dense vector embeddings and LLM-generated natural language summaries. This raises a question: how do

the symbolic graph layer and the sub-symbolic vector/LLM layer interact, and where does the boundary lie?

The architecture maintains a clear division of responsibilities. The *graph layer* (Neo4j) is the system of record for belief state: items, revisions, tags, edges, and deprecation status define the formal structures $\mathcal{B}(\tau)$, τ, and E. All belief revision operations (Definition 7.4–7.7) act on this layer. The *vector layer* (embeddings stored as revision properties) is a derived index: each revision's embedding is computed from its content after creation and serves exclusively as a retrieval accelerator. Critically, the vector layer never modifies belief state—an embedding cannot create, supersede, or deprecate a revision. The formal properties of Section 7 hold independently of whether embeddings are present, absent, or stale.

From vectors to revision pointers. When the vector retrieval branch identifies a semantically relevant embedding, the result is not a raw embedding vector but a *revision kref*—a typed pointer back into the graph. Concretely, each embedding vector is stored as a property on its corresponding `Revision` node; a vector similarity query returns the node itself, from which the system extracts the revision's kref (e.g., `kref://project/space/item.kind?r=3`). This pointer is then resolved through the graph layer to obtain the revision's content $\varphi(r)$, metadata, and edge relationships. The vector space thus functions as an alternative *addressing mechanism*—a way to locate revisions by semantic proximity rather than by structural path—but the addressed object is always a graph-native entity subject to the formal belief revision semantics.

From LLM outputs to graph operations. The system ingests LLM-generated content at two points: (i) during *memory storage*, where an agent's natural language output is captured as a revision's summary, tags, and metadata via the `memory_ingest` MCP tool; and (ii) during *Dream State consolidation* (Section 9), where an LLM assesses existing memories and recommends deprecation, enrichment, or relationship creation. In both cases, the LLM output passes through a structured API boundary that maps it to graph operations: a summary string becomes a revision's `summary` field (part of $\varphi(r)$), a deprecation recommendation becomes a contraction operation (Definition 7.5), and a relationship recommendation becomes an edge in E. The LLM never directly manipulates the graph; it produces structured recommendations that are validated, filtered by safety guards (Section 9.5), and then executed as formally defined operations.

Formal status. The bridge between symbolic and sub-symbolic layers is deliberately asymmetric: the vector/LLM layer *reads from* and *writes through* the graph layer, but cannot bypass it. This ensures that the formal properties established in Section 7 are preserved regardless of the quality or availability of embeddings and LLM assessments. A deployment with no embedding infrastructure retains all formal guarantees (at the cost of reduced retrieval recall); a deployment with a malfunctioning LLM assessment module is contained by the safety guards. The sub-symbolic components enhance the system's practical utility without entering the formal trust boundary.

8.8 Client-Side LLM Reranking

When recall returns stacked items—multiple conversation revisions about the same topic (e.g., different sessions discussing the same person)—the system must select the most relevant sibling revision for the current query. Rather than adding a dedicated server-side reranking model, Kumiho delegates this selection to the *consuming agent's own LLM* through a two-stage filtering pipeline.

Stage 1: Embedding pre-filter. Sibling revisions are filtered by embedding cosine similarity to the query using `text-embedding-3-small` with a threshold of 0.30. This removes obviously irrelevant siblings at negligible cost (<$0.001 per query), reducing the candidate set before the more expensive LLM evaluation.

Stage 2: LLM reranking. The surviving siblings are presented to the LLM with structured metadata—title, summary, extracted facts, entities, events, and implications—alongside the original query. The LLM evaluates each sibling's relevance and selects the best match(es) for the current context.

Three configuration modes accommodate different deployment contexts:

Table 5: LLM reranking configuration modes.

Mode	Context	Reranker	Cost
`client`	Agent via MCP	Host agent LLM	Zero
`dedicated`	API / Playground	User's model	User's key
`auto`	Any	Detect context	Adaptive

In the `client` mode—the primary deployment path for MCP-integrated agents—the host agent (Claude, GPT-4o, etc.) performs reranking as part of its normal response generation. The memory layer returns structured sibling metadata; the agent evaluates it alongside the conversation context. This is *more accurate* than any standalone reranker because a frontier model with full conversation context outperforms a lightweight model operating on a query string alone. Critically, this costs nothing: the reranking is subsumed into the agent's existing inference call.

This design embodies the LLM-Decoupled Memory principle (Section 5): the memory layer provides structured data; the consumer's own intelligence performs selection. As agent models improve, reranking quality improves automatically without any system changes. The memory architecture never needs to upgrade its reranker—it delegates to whatever model the consumer is already running.

Empirical impact. On LoCoMo-Plus goal-type questions (Section 15.3), retrieval hit rate improved from ∼0% to 100% after two changes: (i) including the primary (published) revision in the sibling candidate list so the LLM evaluates *all* revisions, not just non-published siblings; and (ii) expanding the structured metadata presented to the reranker to include extracted facts, entities, events, and implications. The fix was structural, not model-dependent: both GPT-4o and GPT-4o-mini achieved the same retrieval improvement.

9 The Dream State: Asynchronous Consolidation

9.1 Motivation and Prior Art

During sleep, the human brain replays recent experiences, extracts patterns, consolidates episodic memories into semantic knowledge, and prunes redundant connections Rasch and Born [2013]. The Dream State service mirrors this process for AI agent memory, converting a growing collection of raw episodic memories into a cleaner, better-organized memory graph that enables faster and more accurate retrieval.

Prior art. The idea of asynchronous background consolidation is not new. Letta's sleep-time compute [2025] explicitly implements background agents that reorganize memory during idle periods, drawing the same biological metaphor. Google's Vertex AI Memory Bank and Amazon's Bedrock AgentCore Memory both perform asynchronous background extraction. Our contribution is not the concept of offline consolidation but the *safety architecture* surrounding it: the specific mechanisms described in Section 9.5—published-item protection, circuit breakers, dry-run validation, cursor-based resumption, and auditable reports—are, to our knowledge, absent from prior consolidation systems and address the critical question of what happens when automated memory management goes wrong.

9.2 Event-Driven Architecture

The Dream State processes the system's event stream with cursor-based semantics. Events include `revision.created`, `edge.created`, and `revision.deprecated`. The cursor position is persisted on a dedicated internal item (`_dream_state`), enabling resume after interruption. On first run (no cursor), the system replays available history from the beginning.

Trigger mechanisms include: scheduled execution (e.g., nightly at a configured hour); event cursor idle detection; memory count threshold; and explicit API invocation (including through MCP as the `memory_dream_state` tool).

9.3 Nine-Stage Consolidation Pipeline

The pipeline proceeds through the following stages:

1. **Ensure Cursor**: Create the internal `_dream_state` space and item if they do not exist.
2. **Load Cursor**: Read the persisted cursor position; `None` on first run.

3. **Collect Events**: Stream events from the cursor position, grouping by item and deduplicating (latest revision per item wins).
4. **Fetch Revisions**: Batch-load revision metadata from the graph, filtering to episodic memories (`kind=conversation`) and excluding already-deprecated items.
5. **Inspect Bundles**: For bundle-related events, fetch current membership lists to provide topical grouping context.
6. **LLM Assessment**: Submit batches of memories (configurable batch size, default 20) to an LLM with a structured prompt requesting: relevance scoring (0.0–1.0), deprecation recommendations with reasons, tag suggestions for improved retrieval, metadata corrections or enrichments, and relationship identification between memories in the batch.
7. **Apply Actions**: Execute the LLM's recommendations under safety guards (Section 9.5).
8. **Save Cursor**: Persist the new cursor position and timestamp.
9. **Generate Report**: Create a revision on the `_dream_state` item with a detailed Markdown audit report as an artifact, documenting all actions taken, skipped, and failed.

9.4 LLM Assessment Protocol

The assessment prompt instructs the LLM to analyze each memory and return structured JSON with per-memory assessments. For each memory, the LLM evaluates:

- **Relevance**: How useful is this memory for future agent interactions? (0.0–1.0)
- **Deprecation**: Should this memory be deprecated? Criteria include: duplicates of newer memories, superseded information, trivially obvious content, and content with no actionable value. Deprecation implements the formal contraction operation (Definition 7.5): deprecated items are excluded from all search and retrieval by default, effectively removing the belief from the agent's active state while preserving it in the graph for audit or explicit recovery.
- **Enrichment**: What tags, keywords, or metadata corrections would improve future retrieval?
- **Relationships**: Which other memories in the batch relate to this one, and what is the relationship type?

The LLM is explicitly instructed to be conservative: when in doubt, keep the memory. This bias toward preservation is reinforced by the safety guards described below.

Table 6: Dream State safety guards.

Guard	Mechanism
Dry Run	Assessment-only mode
Published Protect	Never deprecate "published"
Circuit Breaker	Max 50% deprecation
Error Isolate	Per-action try/except
Audit Report	Markdown report artifact
Cursor Persist	Resume from checkpoint

Formal status of consolidation. We clarify the relationship between online memory operations (Section 7) and the Dream State's offline consolidation. The nine-stage pipeline is an *engineering* contribution, not a formal one. Each individual action maps to a formally defined operation: deprecation is contraction (Definition 7.5), metadata enrichment is expansion (Definition 7.7), and relationship creation adds edges to E. However, we do *not* claim that the composition of a batch of such actions across multiple memories preserves all AGM postulates simultaneously—proving compositional preservation of rationality postulates under sequential application is an open problem in belief revision theory. What the safety guards provide is not formal guarantees but operational constraints (published-item immunity, circuit breakers, dry-run validation) that limit the damage from incorrect LLM assessments. The formal properties of Section 7 apply to individual memory operations; the Dream State's contribution is the safety architecture surrounding their automated application.

9.5 Safety Guards

Automated memory management requires conservative defaults. Table 6 enumerates the safety mechanisms.

The circuit breaker is particularly important: if the LLM recommends deprecating more than 50% of assessed memories in a single run, this likely indicates a miscalibrated prompt or adversarial input

rather than a genuine consolidation need. The system caps deprecations at 50% and logs a warning for operator review.

Threshold justification. The 50% circuit breaker is motivated by the observation that in a healthy, regularly-consolidated memory graph, each run should be *incremental*: a small fraction of assessed memories will be genuinely redundant or outdated. Three scenarios can trigger a high deprecation rate: (a) a miscalibrated assessment prompt (the LLM is over-aggressive), (b) adversarial or corrupted input to the assessment, or (c) a long-overdue consolidation on a heavily stale graph where many memories have been superseded. In cases (a) and (b), halting is the correct response. In case (c), the system converges through multiple successive runs, each pruning up to 50% of the remaining candidates—geometrically approaching a clean state without risking catastrophic single-run loss.

Published-item protection justification. Published items represent beliefs that have been explicitly validated by a human operator or an upstream approval process—analogous to "approved" assets in production pipelines. Automated consolidation should never override explicit human validation; this is a direct application of the principle that human-in-the-loop decisions outrank automated assessments. The protection ensures that the Dream State cannot silently degrade curated, high-confidence beliefs.

Tunability. Both thresholds are configurable via the API, enabling operators to adapt consolidation behavior to their domain's risk profile:

- The circuit breaker accepts a `max_deprecation_ratio` parameter (default 0.5, valid range 0.1–0.9). *Recommended settings*: high-stakes domains (medical, legal, financial) should use 0.2–0.3; general-purpose agents use the default 0.5; batch cleanup operations on known-stale graphs can use 0.7–0.9, preferably after a dry-run validation pass.
- Published-item protection can be relaxed via an explicit `allow_published_deprecation=true` flag for operators who require full automated control over all items, including curated ones. This flag is logged in the audit report for traceability.
- The dry-run mode is itself a tunability mechanism: operators can preview all proposed actions before committing, adjusting the assessment prompt or thresholds based on the dry-run report.

Principle 10 (Conservative Memory Management)**.** *When an AI system makes automated decisions about memory retention, the default must be preservation. Deleting a useful memory is worse than retaining a useless one. Circuit breakers, dry runs, and protection tags ensure that consolidation enhances quality without risking catastrophic loss.*

9.6 Consolidation as LLM-Decoupled Operation

The Dream State accepts any LLM through a pluggable adapter interface. The consolidation model can differ from the agent's primary model—for instance, using a smaller, cost-efficient model for nightly batch processing while the agent uses a larger model for interactive reasoning. The memory graph is the shared substrate; the LLM is a tool applied to it.

10 Privacy Architecture

10.1 Local-First, Summary-to-Cloud

The privacy architecture enforces a strict boundary: raw content (chat transcripts, voice recordings, images, tool output, user PII) remains local to the agent runtime. Only PII-redacted summaries, extracted facts, topic keywords, artifact *pointers* (paths, not content), and embedding vectors cross the privacy boundary into the cloud graph.

PII redaction is applied during the ingest pipeline before any data reaches the graph database. The redaction step uses an LLM (through the same pluggable adapter interface) to identify and remove personally identifiable information from summaries.

10.2 BYO-Storage Architecture

As described in Section 6.4.2, artifacts in the graph are pointers to files on the user's own storage—local

filesystems, network shares, or cloud storage buckets. The system never copies, caches, or proxies artifact content. This means:

- Users control data residency and retention.
- The graph database contains no raw user content.
- Data exfiltration from the graph yields only metadata.
- Compliance with data sovereignty regulations is simplified.

This design reflects the Metadata Over Content principle (Table 10): the cloud graph stores the minimum information necessary for recall and reasoning, while raw content stays local. This preserves user privacy, reduces storage costs, eliminates data exfiltration risk, and—equally important—enables *cognitive efficiency*: agents read compact summaries to understand context, dereferencing raw content only when exact detail is required.

10.3 Multi-Channel Session Identity

For agents operating across multiple platforms (messaging services, web interfaces, desktop applications), session identity is user-centric, not channel-centric. The session ID format encodes context (e.g., `personal`, `work`), user identity hash, date, and sequence number, enabling cross-channel memory retrieval unified under one identity. The context field serves as a memory namespace—not a tenant isolation boundary—allowing the same user to maintain separate memory contexts (personal preferences vs. work decisions) within a single deployment.

Design note: Identity over channel. Session continuity follows the user, not the platform. An agent that forgets a conversation because the user switched from mobile to desktop has failed at its core purpose.

10.4 Threat Model and Mitigations

The privacy architecture addresses a specific threat model. Table 7 enumerates the primary threats and the mitigations provided by the current design.

We acknowledge that LLM-based PII redaction is not a formal privacy guarantee. False negatives (PII that escapes redaction) and false positives (over-redaction that degrades summary quality) are inherent to the approach. For regulated data classes (HIPAA, GDPR Article 9), operators should deploy a dedicated redaction model with measured precision/recall or a rule-based pre-filter upstream of the LLM redactor. The current architecture supports this through the pluggable adapter interface: the redaction component can be replaced without changing the ingest pipeline.

Table 7: Threat model and mitigations for the memory privacy boundary.

Threat	**Mitigation**
PII leakage via summaries	LLM-based redaction before graph ingest. *Limitation*: LLM redaction has non-zero false negative rates; no formal guarantees.
Membership inference via embeddings	Embeddings are stored per-tenant in isolated database partitions. *Limitation*: embedding inversion attacks are an active research area.
Prompt injection in Dream State	Consolidation operates on stored metadata, not raw user input; safety guards (circuit breaker, published protection) limit blast radius.
Metadata re-identification	Topics and keywords can be identifying even without PII. *Limitation*: current redaction targets named entities, not topical fingerprints.
Malicious artifacts via tool output	Artifact pointers are stored, not artifact content; the graph never executes or parses artifact data.
Credential leakage	The ingest pipeline rejects known credential patterns (API keys, tokens) before summarization.

11 MCP Integration

11.1 The Model Context Protocol

The Model Context Protocol (MCP) MCP [2025] provides a standardized interface between AI agents

and external tools. By exposing the memory system as MCP tools, any MCP-compatible agent gains memory capabilities without custom integration code. MCP-based memory systems are now table stakes for adoption—the official MCP Memory Server, Mem0 MCP Server, Redis Agent Memory Server, Neo4j MCP Servers, MemoryOS-MCP, and Basic Memory all provide MCP interfaces. Our MCP integration is not a differentiator but a necessary condition for platform-agnostic deployment.

What distinguishes the interface is the *breadth* of graph operations exposed: not just store/retrieve, but reasoning and provenance tools (AnalyzeImpact, FindPath, GetProvenance) and temporal point-in-time queries that other MCP memory servers do not provide.

11.2 Tool Taxonomy

The MCP server exposes 51 tools organized into six categories:

Cognitive Memory Lifecycle (`memory_ingest`, `memory_recall`, `memory_consolidate`, `memory_discover_edges`, `memory_store_execution`, `memory_dream_state`, `memory_add_response`): The complete memory pipeline. `memory_ingest` buffers user messages in working memory while simultaneously recalling relevant long-term memories and creating atomic memory units (see below), providing the agent with prior context on every turn. `memory_add_response` captures assistant responses into the working memory buffer. `memory_consolidate` summarizes sessions with PII redaction and consolidation enrichments (prospective indexing, event extraction). `memory_recall` performs semantic search over long-term memory with optional graph-augmented multi-query reformulation and edge traversal. `memory_discover_edges` generates implication queries from a newly stored memory and creates typed edges to related existing memories—the mechanism that keeps the graph connected. `memory_store_execution` persists tool execution results as memory artifacts. `memory_dream_state` triggers the Dream State consolidation pipeline (Section 9).

Working Memory (`chat_add`, `chat_get`, `chat_clear`): Redis-backed session buffer for conversational context before consolidation to long-term graph storage.

Graph Navigation (`get_project`, `get_spaces`, `get_item`, `get_revision`, `get_artifacts`, `search_items`): Structured exploration of the project/space/item/revision hierarchy.

Reasoning & Provenance (`get_edges`, `get_dependencies`, `get_dependents`, `analyze_impact`, `find_path`, `get_provenance_summary`): Understanding *why* memories exist and how they relate, enabling chain-of-evidence reasoning.

Temporal Operations (`get_item_revisions`, `get_revision_by_tag`, `get_revision_as_of`, `resolve_kref`): Navigating memory through time, including point-in-time queries ("what did the agent believe about X on date Y?").

Graph Mutation (`create_item`, `create_revision`, `tag_revision`, `create_edge`, `set_metadata`, `deprecate_item`, and 13 additional CRUD operations): Programmatic graph structure management for agent-driven and multi-agent workflows.

11.3 Atomic Memory Writes

A single `memory_ingest` invocation creates the complete memory unit: space (with auto-creation), item, revision with metadata, artifact attachment, edges to source materials, bundle membership, tag assignment, and asynchronous embedding generation.

This "one tool call, complete memory" design eliminates fragile multi-step sequences that could leave partially-committed state in the graph.

Design note: One tool call, complete memory. A single MCP tool invocation must create the full memory unit. Requiring multiple sequential tool calls creates fragile, partially-committed states and increases agent complexity.

11.4 Human Auditability

MCP makes memory accessible to agents, but trustworthy systems also require a human-auditable surface. A web dashboard and companion desktop asset browser render the cognitive memory graph

using the same project/space/item/revision hierarchy, enabling operators to inspect what an agent remembered, why, and when through immutable revision history, lineage traversal, and artifact inspection. Every consolidation decision is traceable to a Dream State report; every memory has a provenance chain.

For agents performing consequential work, this auditability is not a convenience but a requirement. The same project/space/item/revision hierarchy serves both asset browsing and memory inspection, ensuring that the cognitive state of an agent is as navigable and auditable as a traditional asset management system.

12 Comparative Analysis

Table 8 compares the architecture across nine evaluation dimensions. Table 9 provides a feature-level comparison with concurrent systems.

12.1 vs. Flat Retrieval Systems

Flat retrieval systems treat each query as an independent similarity search over static document chunks. The proposed architecture extends this model along three axes: (i) *statefulness*—memories evolve through revisions rather than being overwritten or duplicated; (ii) *structure*—typed edges encode causal and evidential relationships between memories; (iii) *consolidation*—the Dream State actively distills episodic experience into semantic knowledge.

Flat retrieval remains valuable as one component of a hybrid system. The architecture incorporates vector similarity as one of two retrieval branches (alongside fulltext search), recognizing that embedding-based search is effective for semantic matching but insufficient as a complete memory system.

12.2 vs. Tiered Buffer Systems

Tiered buffer systems introduce an important operating-system metaphor for memory management. However, they typically operate on flat stores without typed relationships or immutable versioning. More critically, the memory management logic is typically embedded within the LLM's own reasoning process: the model decides when to move content between tiers, creating tight LLM-memory coupling.

The proposed architecture provides richer structure (graph edges, revision history), stronger safety guarantees (circuit breakers, dry runs, published protection), and LLM-decoupled memory management (the memory graph is an external data store, not model-internal state).

12.3 vs. Extended Context Windows

As argued in Section 3, context window extension addresses the wrong problem. It increases attention capacity but does not provide persistent recall, structural representation, or model-independent storage. The quadratic scaling of attention cost makes it economically infeasible for lifelong agent memory. Extended context is complementary—useful for in-session reasoning over recently retrieved memories—but not a substitute for persistent, structured, externalized memory.

12.4 vs. Static Knowledge Graphs

Static knowledge graphs excel at representing shared, encyclopedic knowledge with fixed schemas. The proposed architecture is designed for *experiential* memory—agent-scoped, temporally-evolving, with flexible metadata and a working memory layer. The two approaches are complementary: a memory-equipped agent could reference external knowledge graphs via REFERENCED edges while maintaining its own experiential memory in the graph.

12.5 vs. MAGMA

MAGMA [2026] represents the most direct structural alternative, implementing a multi-graph architecture with four orthogonal graph layers (semantic, temporal, causal, entity) and policy-guided retrieval traversal. The two architectures represent alternative philosophical commitments. MAGMA disentangles memory dimensions into separate graphs, enabling cleaner retrieval routing: a temporal query traverses only the temporal graph, a causal query traverses only the

Table 8: Comparative analysis across nine evaluation dimensions.

Dimension	Flat RAG	Tiered Buffers	Extended Context	Static KG	Kumiho
Retrieval	Embedding only	Embedding + scan	In-context	Query language	Hybrid + graph nav
Statefulness	Stateless	Tiered buffers	Ephemeral	Current state	Versioned history
Relationships	None	None	None	Fixed ontology	6 typed edge types
Provenance	None	None	None	Schema-dependent	Complete lineage
Temporal Nav.	Point-in-time	Current window	Current window	Mostly current	Full history + tags
Consolidation	None	Manual / rule	N/A	Batch ETL	LLM Dream State
LLM Coupling	Low	High	Complete	None	None (MCP)
Cost Scaling	Linear	Linear	Quadratic	Linear	Linear
Work Auditability	None	None	None	Partial	Full (SDK + desktop)

Table 9: Feature comparison with concurrent agent memory systems (as of Feb 2026). ✓ = present, (✓) = partial, × = absent.

Feature	Graphiti	Mem0g	A-MEM	Letta	MAGMA	Hindsight	MemOS	Kumiho
Property graph storage	✓	✓	(✓)	×	✓	×	×	✓
Hybrid retrieval (≥2 modalities)	✓	(✓)	×	×	(✓)	(✓)	×	✓
Immutable revision history	(✓)	(✓)	×	(✓)	×	×	×	✓
Formal belief revision	×	×	×	×	×	×	×	✓
URI-based addressing	×	×	×	×	×	×	×	✓
Typed edge ontology (≥6)	(✓)	×	(✓)	×	(✓)	×	×	✓
Async. consolidation + safety guards	×	×	×	(✓)	×	×	×	✓
LLM-decoupled	(✓)	(✓)	×	(✓)	(✓)	(✓)	(✓)	✓
Unified asset + memory graph	×	×	×	×	×	×	×	✓
Agent work auditability (SDK)	×	×	×	×	×	×	×	✓
Benchmark eval. (LoCoMo/LME)	✓	✓	✓	✓	✓	✓	×	✓[†]
BYO-storage (raw data stays local)	×	×	×	×	×	×	×	✓

[†] LoCoMo: 0.447 four-category F1 / 97.5% adversarial refusal (Section 15.2). LoCoMo-Plus: 93.3% (Section 15.3). LongMemEval planned.

causal graph. Our architecture unifies all relationships in a single property graph with typed edges, enabling cross-dimensional traversal: AnalyzeImpact propagates across Depends_On, Derived_From, and Supersedes edges simultaneously, discovering dependencies that span multiple memory dimensions. The unified graph design reflects a deliberate architectural trade-off. Multi-graph separation offers cleaner retrieval routing and avoids cross-dimensional noise, but introduces synchronization complexity: updates that span multiple dimensions (e.g., a belief revision that is simultaneously temporal, causal, and semantic) must be coordinated across separate graph instances, and cross-dimensional queries require joins across storage boundaries. The unified property graph accepts edge-type heterogeneity in exchange for transactional atomicity—a single Neo4j transaction can create a revision, re-point tags, add Supersedes and Derived_From edges, and update provenance metadata, ensuring that the belief state is never in a partially-updated condition. This atomicity is what enables the AGM compliance results (Section 15.7): the formal postulates require that revision is an atomic operation, and the unified graph makes this a database-level guarantee rather than an application-level coordination problem. Neither approach has been empirically compared to the other. MAGMA's 0.70 on LoCoMo (LLM-as-judge score, not token-level F1) and 61.2% on LongMemEval (with 95% token reduction) establish strong baselines; whether our unified graph approach achieves comparable or better results under the same evaluation conditions is the most important open empirical question for this work.

12.6 vs. Hindsight

Hindsight [2025] achieves the highest reported LoCoMo (89.61%, percentage accuracy) and LongMemEval (91.4%) scores, demonstrating that prag-

Table 10: Core design principles.

#	Principle	Category
1	Structural Reuse	Structure
2	Universal Addressability	Structure
3	Immutable Rev., Mutable Ptr.	Formal
4	Explicit Over Inferred Rel.	Formal
5	Non-Blocking Enhancement	Performance
6	Conservative Memory Mgmt.	Safety
7	Metadata Over Content	Safety

matic belief tracking without formal guarantees can deliver strong empirical results. Its Opinion Network maintains confidence-scored beliefs that update with evidence—functionally similar to our revision mechanism, but without AGM grounding. The relationship is complementary rather than competitive: Hindsight demonstrates the empirical value of structured belief tracking; our formal framework provides the theoretical guarantees (Relevance, Core-Retainment, Consistency) that such systems could adopt to ensure belief revision satisfies minimal change and does not discard beliefs without justification. Hindsight explicitly identifies safe personality management as an open problem; our safety-hardened consolidation with circuit breakers and published-item protection addresses precisely this gap.

13 Core Design Principles

Table 10 consolidates the seven core design principles governing the architecture. We distinguish these research-level principles—which define the architecture's formal and structural commitments—from additional engineering design notes (boundary validation, latency matching, infrastructure independence, query sanitization, session identity, and atomic writes) documented inline in the relevant sections.

14 Reference Implementation: Kumiho

The architecture is fully implemented as the Kumiho system[2], which provides open-source client SDKs (Python, C++, Dart, MCP server) with the core graph server delivered as a managed cloud service, and validated through deployment:

Kumiho Server. Rust-based gRPC server handling graph operations, dynamic connection routing, embedding generation, and hybrid search using `tokio`, `tonic`, and `neo4rs`.

Kumiho SDK. Python SDK providing typed access to all graph operations with memory reference validation and retry logic.

Kumiho MCP Server. Python MCP server wrapping the SDK, exposing all memory operations as MCP tools.

Kumiho Memory Library. Python library providing higher-level cognitive memory operations: session management, memory ingest with automatic recall, PII-redacted summarization with consolidation enrichments (prospective indexing, event extraction), and the Dream State pipeline.

Kumiho Dashboard. Web-based interface at `https://kumiho.io` (Figure 1) providing two integrated views: (i) an *AI Cognitive Memory* browser with fulltext search, interactive force-directed graph visualization of typed edges, node detail panels (title, summary, kref URI, revision tags, creation date), and live stats (total memories, spaces, connections, Dream State recency); and (ii) an *Asset Browser* for navigating non-memory graph items (models, textures, workflows). Both views share the same project/space/item/revision hierarchy and the same graph API, unifying work product management and memory inspection in a single web interface. Node types are color-coded by kind (conversation, decision, fact, reflection, error, action, instruction, bundle) and edges are color-coded by type (Depends On, Derived From, Referenced, Contains, Created From, Belongs To, Supersedes).

Kumiho Desktop. Cross-platform desktop asset browser (Flutter/Dart) providing offline-capable access to the same graph hierarchy, optimized for

[2] `https://kumiho.io`, `https://github.com/kumihoclouds`

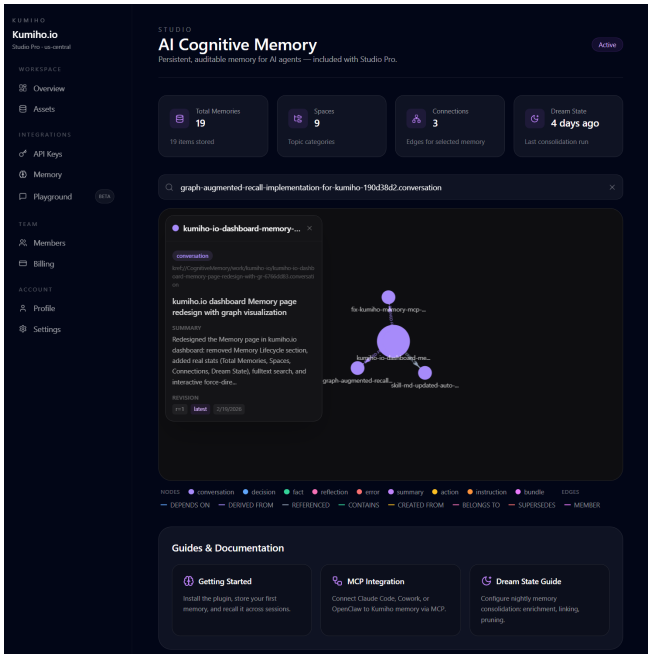

Figure 1: Kumiho dashboard showing the AI Cognitive Memory browser. The selected node (*kumiho.io dashboard Memory page redesign with graph visualization*) displays its summary, kref URI, revision tag (`r=1`, `latest`), and creation date. The force-directed graph on the right visualizes typed edges to related memories. Stats cards show total memories (19), spaces (9), connections for the selected memory (3), and Dream State recency (4 days ago). The same graph, SDK, and API that agents use to manage work products is exposed here for human inspection.

browsing large asset collections on local and NAS storage.

Measured end-to-end latencies: 15 ms typical for working memory, 80–120 ms for long-term graph queries including hybrid search.

15 Implementation Validation: A Case Study

Scope. This section presents three categories of empirical evidence: (i) standardized benchmark evaluations on LoCoMo and LoCoMo-Plus with cross-system comparison, (ii) formal AGM compliance verification, and (iii) operational case studies from deployment. The benchmark evaluations are the primary empirical contributions; the case studies demonstrate that the architecture functions as designed in production use.

Table 11: Token compression ratios.

Type	Summary	Raw	Ratio
Simple fact	∼12	500	42×
Preference	∼18	800	44×
Profile	∼80	4K	50×
Paper session	∼65	12K	185×
Planning	∼90	25K	278×

15.1 Token Compression

A central claim of the architecture is that storing compact metadata summaries rather than raw transcripts yields significant token savings. Table 11 shows representative examples from the deployed graph, comparing the stored summary token count against the estimated raw conversation token count.

The compression ratio ranges from ∼40× for simple facts to ∼280× for complex multi-turn sessions. Critically, this compression compounds at retrieval time: a typical recall returning k=5 results injects ∼250–400 summary tokens into the agent's context, compared to the ∼50,000+ tokens that would be required to replay the raw transcripts of those same sessions. This corresponds to the cost scaling advantage formalized in Section 3: the retrieval approach scales as $O(k \cdot \bar{s})$ where $\bar{s}$ is the mean summary size, while raw replay would scale as $O(k \cdot \bar{c})$ where $\bar{c}$ is the mean conversation size.

15.2 LoCoMo Benchmark Evaluation

We evaluated Kumiho on the LoCoMo benchmark [2024], a multi-session conversation benchmark comprising 1,986 questions across 10 conversations and five categories that test long-term memory retrieval across extended conversational histories. We report results using the official *token-level F1 with Porter stemming* metric—the standard scoring function used by the broader community (Zep [2025], Mem0 [2025], Memobase [2025]). The evaluation uses summarized recall mode with GPT-4o as the answer model, graph-augmented retrieval with multi-query reformulation, and a recall

Table 12: LoCoMo token-level F1: cross-system comparison (four retrieval categories).

System	Single	Multi	Temp.	Open	Overall	Source
Zep	0.357	0.194	0.420	0.496	—	Zep [2025]
OpenAI Mem	—	—	—	—	∼0.34	Zep [2025]
Mem0	0.387	0.286	0.489	0.477	∼0.40	Zep [2025]
Mem0-Graph	0.381	0.243	0.516	0.493	∼0.40	Zep [2025]
Memobase	0.463	0.229	0.642	0.516	—	Memobase [2025]
ENGRAM	0.231	0.183	0.219	0.086	0.211	Patel and Patel [2025]
Kumiho	**0.462**	**0.355**	**0.533**	0.290*	**0.447**‡	This work

* Open-domain questions require world knowledge absent from conversation history; a memory-only system's expected floor (see text).
‡ Four-category weighted average (n=1,540). Including adversarial refusal accuracy (97.5%, n=446, binary scoring), overall F1 is 0.565 (n=1,986).

Table 13: LoCoMo per-category breakdown (n=1,986).

Category	n	F1
Single-hop	841	0.462
Multi-hop	282	0.355
Temporal	321	0.533
Open-domain	96	0.290
Retrieval average	**1,540**	**0.447**
Adversarial (refusal acc.)†	446	0.975
Overall (incl. adversarial)	1,986	0.565

† Binary refusal detection, not continuous token-level F1. See Caveats.

limit of 3 memories per query with context top-k=7.

Table 12 reports the cross-system comparison on the four retrieval categories (single-hop, multi-hop, temporal, open-domain). Kumiho achieves **0.447 four-category F1** (n=1,540)—the highest reported score on the official LoCoMo token-level metric LoCoMo [2024], Patel and Patel [2025], Zep [2025]. Adversarial refusal accuracy (97.5%, n=446) is reported separately in Table 13, as it uses binary refusal detection rather than continuous token-level F1. Including adversarial, overall F1 is 0.565 (n=1,986). Table 13 provides the per-category breakdown.

Key findings. Four observations emerge from the token-level F1 evaluation:

(i) Architecture generalization. The same graph-native architecture that dominates LoCoMo-Plus (93.3% on Level-2 cognitive constraints, Section 15.3) also leads on standard factual QA with 0.447 four-category F1. The result is not attributable to a different pipeline configuration—the same prospective indexing, event extraction, and graph-augmented recall that bridge cue-trigger semantic disconnects also improve factual retrieval accuracy.

(ii) Adversarial refusal accuracy. The system correctly refuses to fabricate information in 97.5% of adversarial questions (n=446). This is arguably the most important metric for production trust: a memory system that hallucinates plausible-sounding but incorrect answers is more dangerous than one that fails to retrieve. The near-perfect score is a natural consequence of the belief revision architecture (Section 7): the memory graph genuinely does not contain fabricated information—immutable revisions preserve only what was actually discussed, and consolidation-as-denoising strips the surface-level cues that adversarial questions exploit—so there is nothing for the model to hallucinate from. Note that adversarial scoring uses binary refusal detection (presence of refusal phrases), not continuous token-level F1.

(iii) Multi-hop as strongest relative improvement. Multi-hop F1 of 0.355 exceeds Mem0 (0.286, +6.9 pp) and Mem0-Graph (0.243, +11.2 pp). Multi-hop questions require connecting information across multiple conversation segments—precisely the scenario where graph-augmented recall with edge traversal provides a structural advantage over flat vector stores. The typed REFERENCED and DERIVED_FROM edges created during ingestion enable the retrieval pipeline to discover memories that are structurally connected but semantically distant in embedding space.

(iv) Open-domain as expected floor. Open-domain questions (0.290) require world knowledge beyond the conversation history—e.g., general facts about geography, science, or culture that the participants did not discuss. A memory-only system cannot provide this knowledge, making open-domain the expected accuracy floor rather than a system failure. Notably, Zep (0.496) and Mem0 (0.477) score higher on open-domain, likely because their retrieval pipelines surface broader context that incidentally overlaps with world knowledge.

Cross-benchmark architecture consistency. Table 14 demonstrates that the architectural innovations generalize across both evaluation protocols—they are not tuned to a single bench-

Table 14: Architecture consistency across benchmarks.

Innovation	LoCoMo-Plus	LoCoMo F1
Prospective idx	Bridges cue-trigger gap	Improves temporal/multi-hop
Event extraction	Preserves causal chains	Preserves factual detail
LLM reranking	100% goal retrieval hit	Better sibling selection
Graph-aug. recall	Connected memory discovery	Multi-hop 0.355 (best)

mark.

Cost. The full LoCoMo token-level F1 evaluation (1,986 questions across 10 conversations) costs ∼$10 with GPT-4o-mini as the answer model and ∼$14 with GPT-4o. The pipeline cost structure mirrors LoCoMo-Plus: consolidation and enrichment are one-time per session, while per-query costs are dominated by answer generation.

Caveats. Competitor F1 scores are sourced from published evaluations Zep [2025], Memobase [2025], Patel and Patel [2025], not from controlled re-evaluation on identical infrastructure; different evaluation configurations (Top-K, prompt templates, embedding models) may account for some cross-system variance. The adversarial category uses binary refusal detection (match against refusal phrases), not continuous token-level F1—the ground truth is not used for scoring. The 97.5% adversarial score is therefore a refusal accuracy metric, methodologically distinct from the four retrieval categories. We report the four-category F1 (0.447) as the primary comparable number and the overall F1 (0.565, including adversarial) separately. Open-domain performance (0.290) is below several competitors—this reflects the architectural decision to retrieve only from the memory graph rather than augmenting with external knowledge sources.

15.3 LoCoMo-Plus Benchmark Evaluation

Where LoCoMo (Section 15.2) evaluates factual recall under strong semantic alignment between query and stored content, LoCoMo-Plus [2026] tests a fundamentally harder capability: implicit constraint recall under intentional cue-trigger semantic disconnect. In LoCoMo-Plus, a *cue* dialogue embeds a constraint (e.g., "Joanna decided to quit sugar after feeling sluggish") that a later *trigger* query must connect to (e.g., "I've been indulging in all kinds of new desserts lately"), with no surface-level lexical overlap between them. The benchmark comprises 401 entries stitched into 10 LoCoMo base conversations (19–32 sessions each, 369–689 turns per conversation), spanning four constraint types—causal (n=101), state (n=100), goal (n=100), value (n=100)—with time gaps of 2 weeks to 12+ months between cue and trigger. We report results on all 401 entries across all four constraint types.

System configuration. The evaluation uses the same graph-native architecture evaluated on LoCoMo, operating in summarized recall mode (title + summary metadata). Four architectural mechanisms address the semantic disconnect that makes LoCoMo-Plus qualitatively harder than LoCoMo:

Prospective indexing. During session consolidation, the summarizer generates 3–5 hypothetical future scenarios in which the memory would be relevant, using vocabulary and framing different from the original conversation. These implications are indexed alongside the summary for both fulltext and vector retrieval. The technique bridges the cue-trigger semantic gap at *write time*: when a conversation mentions lactose intolerance making someone bedridden, one generated implication might be "Months later, Caroline carefully reads restaurant menus before agreeing to dinner dates, prioritizing her dietary restrictions over social convenience." At query time, the trigger—"I finally said yes to a dinner date, but I picked the place solely because I know I won't end up doubled over afterward"—finds the memory through the implication's alternative framing, despite sharing no vocabulary with the original conversation. This is analogous to how human memory works: we encode not just what happened, but what it *means for the future*. The encoding shapes retrieval. Implications are generated by a light model (GPT-4o-mini) running in parallel with the full-model summarization via `asyncio.gather`, adding zero wall-clock time.

Event extraction. The consolidation pipeline extracts structured events from each session, each comprising a description and its consequences (e.g., "Joanna decided to quit sugar → Improved energy levels"). These events are appended to the summary text before indexing. Where narrative summaries compress episodes into abstract descriptions ("dis-

cussed dietary changes"), event extraction preserves the specific incidents and their causal chains that LoCoMo-Plus questions target. Events and implications are complementary: events preserve *what happened* with causal structure; implications anticipate *when it will matter* with alternative vocabulary.

Sibling relevance filtering. After retrieval, sibling revisions (other memories from the same item) are filtered by embedding cosine similarity to the trigger query using `text-embedding-3-small` with a threshold of 0.30. This prevents context dilution where loosely-related siblings overwhelm the answer model. In one case, an entry initially presented 25 sibling revisions to the answer model; after filtering, only 2 memories / 4 revisions reached the answer model, producing a correct answer. The system performs quality control over what reaches the consumer—managing retrieved context, not just retrieving everything loosely related.

Client-side LLM reranking. After cosine prefiltering, remaining sibling revisions are evaluated by the consuming agent's own LLM using structured metadata (title, summary, facts, entities, events, implications). In agentic contexts (MCP), the host agent performs reranking as part of its normal response generation at zero additional inference cost. In non-agentic contexts (API, playground), a dedicated lightweight model (e.g., GPT-4o-mini) handles it using the user's own API key. Three configuration modes are supported: *client* (host agent LLM, zero cost), *dedicated* (user-configured model), and *auto* (detect context). This design reflects the LLM-decoupled architecture (Section 8.8): the memory layer provides structured data; the consumer's own intelligence performs selection. As agent models improve, reranking quality improves automatically without any system changes.

The pipeline uses GPT-4o-mini for summarization, query reformulation, edge discovery, event extraction, implication generation, and judging, with GPT-4o for answer generation only. Each entry is ingested end-to-end: parallel session consolidation into long-term graph storage, followed by LLM-driven edge discovery that links each memory to related existing memories via typed edges (REFERENCED), bridging the cue-trigger semantic disconnect structurally. Recall uses multi-query reformulation (3–4 semantic variants per trigger) with graph-augmented retrieval (edge traversal to surface structurally connected memories that vector similarity alone would miss).

Table 15: LoCoMo-Plus: Kumiho vs. published baselines (n=401).

System	Model	Acc. (%)
RAG (text-ada-002)	text-ada-002	23.5
RAG (text-embed-small)	text-embed-small	24.9
RAG (text-embed-large)	text-embed-large	29.8
Mem0	Various	41.4
A-MEM	Various	42.4
SeCom	Various	42.6
GPT-4o (full context)	GPT-4o	41.9
GPT-4.1 (full context)	GPT-4.1	43.6
Gemini 2.5 Flash (1M)	Gemini 2.5 Flash	44.6
Gemini 2.5 Pro (1M)	Gemini 2.5 Pro	45.7
Kumiho (4o-mini answer)	**GPT-4o-mini**	**∼88**
Kumiho (4o answer)	**GPT-4o**	**93.3**

Table 15 reports the headline comparison. Kumiho achieves 93.3% judge accuracy (374/401) on all entries, outperforming the best published baseline (Gemini 2.5 Pro, 45.7%) by 47.6 percentage points. Recall accuracy—the fraction of entries where the system retrieves at least one relevant memory—reaches 98.5% (395/401). The result is particularly striking because the system uses GPT-4o-mini—one of the cheapest available models—for the bulk of LLM operations (summarization, query reformulation, edge discovery, event extraction, implication generation, judging), with GPT-4o used only for answer generation. Even with GPT-4o-mini as the answer model (∼88%), the system more than doubles the previous state-of-the-art. Gemini 2.5 Pro with its 1M+ token context window can fit entire conversation histories without any summarization or retrieval, yet achieves only 45.7%—demonstrating that cognitive memory is not a context capacity problem but an *organization, enrichment, and retrieval* problem.

Performance by constraint type. Table 16 reports accuracy across all four constraint types with both answer models. The LoCoMo-Plus dataset distributes entries approximately equally across types (101 causal, 100 each for state, value, goal). Causal, state, and value types all achieve 96% with GPT-4o—near-ceiling perfor-

Table 16: LoCoMo-Plus accuracy by constraint type (n=401).

Type	n	Correct	4o (%)	4o-mini (%)
Causal	101	97	96.0	96.0
State	100	96	96.0	95.0
Value	100	96	96.0	∼89
Goal	100	85	85.0	∼73
Overall	**401**	**374**	**93.3**	**∼88**

Table 17: LoCoMo-Plus accuracy by time gap (n=401, GPT-4o answer).

Time Gap	n	Acc. (%)	Notes
≤2 weeks	35	88.6	More goal-type entries
2 wk – 1 mo	77	97.4	Peak performance
1 – 3 mo	164	93.9	Largest cohort
3 – 6 mo	93	93.5	No degradation
>6 mo	32	84.4	Cliff eliminated

mance demonstrating that the architecture's event extraction and prospective indexing produce summaries rich enough for reliable retrieval and reasoning across all three types. Goal-type questions (85%) remain the hardest: they require the most abstract reasoning to connect a current trigger to a stored intention (e.g., connecting "can't believe they charge that much for a car key" to "saving for an engagement ring" requires understanding that both involve money management, despite no vocabulary overlap).

The model impact is type-dependent: switching from GPT-4o-mini to GPT-4o yields causal +0%, state +1%, value +7%, goal +12%. The harder the constraint type, the more a stronger answer model helps. This confirms that the bottleneck for the hardest queries is answer model reasoning capacity, not retrieval quality—the 98.5% recall accuracy is invariant across models.

Performance by time gap. Table 17 reveals the most significant empirical finding of this evaluation: the elimination of the long-horizon accuracy cliff. In a pre-enrichment run (n=200, without prospective indexing or event extraction), accuracy dropped to 37.5% for time gaps exceeding 6 months. With both enrichments active, accuracy at >6 months rises to 84.4%—a 47 percentage point improvement that validates prospective indexing as the critical mechanism for bridging long temporal gaps. The improvement is straightforward to explain: as time increases, embedding similarity between cue and trigger naturally decays. Prospective indexing provides alternative retrieval paths through the generated implications, whose vocabulary is independent of the original conversation's wording and therefore does not suffer the same temporal decay.

The ≤2 week bucket (88.6%) is slightly lower than adjacent periods because it contains proportionally more goal-type constraints, which are harder regardless of time gap. Accuracy is consistent across all 10 base conversations (90–97.5%), with no conversation falling below 90%, demonstrating robustness across different dialogue structures, conversation lengths, and topic distributions.

Failure mode taxonomy. Analysis of the 27 failures (6.7%) across 401 entries reveals two distinct failure modes:

1. **Recall miss** (6 failures, 22%): No relevant memories were retrieved by any query variant, or the retrieved context was empty. These represent the system's hard floor—cases where even prospective indexing and graph-augmented recall could not bridge the semantic gap. Notably, time gap is not the primary predictor: 2 of 6 recall misses occur at ≤1 week.
2. **Answer fabrication** (21 failures, 78%): The correct memory appeared in the recalled context, but the answer model generated a response that ignored it or fabricated around it. The dominant pattern is *surface-theme hijacking*: the model follows the trigger's surface theme and tone rather than connecting to the contradictory or abstract recalled memory. Example: the trigger mentions "indulging in new desserts" while the recalled context states "Joanna decided to quit sugar after experiencing constant sluggishness"—the model acknowledged the sugar quit but fabricated details ("dairy-free ice cream recipe") matching the trigger's positive tone rather than surfacing the contradiction. All 15 goal failures are answer fabrication—the system retrieved the right memory every time, but the model could not bridge the abstract gap between trigger and stored intention.

This failure distribution is architecturally significant: 98.5% recall accuracy means the memory layer has effectively solved the retrieval problem for LoCoMo-Plus. The remaining 6.7% end-to-end gap is entirely attributable to the answer model's reasoning limitations, not to the memory architecture.

Model-decoupled architecture. The system's model-decoupled design (Section 5) allows swapping the answer model without any pipeline changes. Table 16 reports both GPT-4o and GPT-4o-mini results. The 98.5% recall accuracy is constant across models—both receive identical recalled context from the same retrieval pipeline. Only the end-to-end accuracy differs (93.3% vs. ∼88%), concentrated in the hardest constraint types (goal +12%, value +7%). This empirically validates that the memory layer is infrastructure that outlives any single model generation: as LLMs improve at reasoning over retrieved context, the system's end-to-end accuracy improves automatically. The theoretical ceiling with perfect answer generation is ∼99%, limited only by the 6 genuine recall misses.

Cost analysis. The total cost for 401 entries is ∼$14, with summarization (∼$3), edge discovery (∼$2), implication generation (∼$1), event extraction (∼$1), answer generation (∼$3), judging (∼$0.50), and sibling embedding (∼$0.10) as the primary components. This cost efficiency stems from the same architectural principle validated by the LoCoMo summary-only results: structured summarization with enrichment (cheap, one-time per session) replaces brute-force full-context retrieval (expensive, per-query). For comparison, running Gemini 2.5 Pro with full context on 401 entries would cost significantly more while achieving only 45.7% accuracy.

Architectural significance. The LoCoMo-Plus results validate the central thesis of this paper more decisively than LoCoMo. LoCoMo's factual recall questions have strong semantic alignment between query and stored content—precisely the scenario where embedding-based retrieval excels. LoCoMo-Plus deliberately breaks this alignment, testing whether the system can bridge semantic gaps through structural reasoning. The 47.6 percentage point advantage over all baselines demonstrates that graph-native primitives—typed edges bridging cue-trigger disconnects, structured summarization preserving causal relationships, prospective indexing anticipating future retrieval needs, sibling relevance filtering controlling context quality, and multi-query reformulation expanding the retrieval surface—provide capabilities that neither larger context windows nor stronger models can substitute for.

The enrichment contribution is quantified by comparing against a pre-enrichment baseline run (n=200, without prospective indexing, event extraction, or sibling filtering): 61.6% judge accuracy. With the full enrichment pipeline on the complete 401-entry dataset, accuracy rises to 93.3%—a 31.7 percentage point improvement. The >6-month accuracy cliff is the clearest evidence: pre-enrichment accuracy at this horizon was 37.5%; with enrichments, it rises to 84.4%. The mechanism is general: any memory system that generates retrieval-oriented metadata at write time can bridge semantic gaps that would otherwise require exact vocabulary match or enormous context windows.

The separation of recall accuracy (98.5%) from end-to-end accuracy (93.3%) is the second signature analytical result. It demonstrates that the memory architecture has effectively solved the retrieval problem for Level-2 cognitive memory, and that the remaining gap is a consumer-side reasoning challenge, not a memory challenge. Any improvement to the answer model's ability to use retrieved context—whether through better prompting, stronger models, or fine-tuning—translates directly to higher end-to-end accuracy without any architectural changes.

Ablation study (planned). To isolate the contribution of each consolidation enrichment, we plan four configurations on the same 401-entry benchmark: (i) summary only (base consolidation, no enrichments), (ii) summary + event extraction, (iii) summary + prospective indexing, (iv) full system (summary + events + implications + sibling filter). The hypothesis: events and implications are complementary—events preserve factual anchors (what happened); implications provide semantic bridges (what it means for the future). Neither alone is sufficient for the highest accuracy. Proof case: cog-261 (730-day gap)—events provided the factual anchor (cottage purchase), implications provided the semantic bridge (financial planning

goals). Neither alone would have retrieved this memory across a 2-year gap.

Caveats. Baseline scores are taken from the LoCoMo-Plus publication [2026], not from controlled re-evaluation. As with the LoCoMo evaluation, differences in LLM configuration and evaluation methodology may account for some variance. The GPT-4o-mini answer model scores (∼88% overall, ∼73% goal, ∼89% value) are estimated from separate runs; exact figures may vary by ±1–2 percentage points due to non-deterministic LLM behavior.

15.4 Retrieval Observations

Beyond the standardized benchmarks, informal testing against the live deployment graph revealed two qualitative properties of the hybrid retrieval pipeline worth noting. First, the system consistently surfaces semantically adjacent memories alongside exact matches—querying "favorite color" also retrieves other preference memories—reflecting the hybrid design's breadth. Second, for a query on "AGM belief revision," the system correctly ranked the planning memory (which described the *intent* to add AGM proofs) above the execution memory (which recorded the *completed* AGM section), suggesting that the fulltext index captures semantic relevance beyond simple keyword matching. These observations are anecdotal; a comprehensive retrieval evaluation with 100+ queries spanning multiple memory categories, along with ablation studies comparing fulltext-only, vector-only, and hybrid configurations, is planned as future work (Section 16).

15.5 Cross-Session Provenance: A Case Study

A demanding test of the architecture was the iterative authorship of this paper, which spanned 6 revision sessions across 3 days. Each revision was stored as a separate memory item in the `work/paper-project` space, with DERIVED_FROM edges encoding the lineage:

`v1` → `v2` → `v3` → `v3-upd` → `v4` → `v5` → `v6`

This provenance chain enabled several capabilities not available in flat retrieval systems: (i) an agent resuming work on a new session could traverse the chain to understand what had changed and why; (ii) querying the provenance summary of v6 resolved the full dependency graph including the planning session that preceded it; (iii) the chain provided an auditable record of the document's evolution for human review.

The paper collaboration also demonstrated the belief revision mechanism in practice: the formal AGM section (v6) was planned in one session and executed in another, with the planning memory serving as the source dependency. When the user provided refinement feedback (e.g., improving temporal tag representations), the agent could recall prior planning context without re-reading the full paper, using only the compact summary stored in the graph.

15.6 Belief Revision in Practice

The deployed system captured a concrete instance of belief revision that exercises the formal machinery of Section 7. We walk through the complete graph-level lifecycle to illustrate how the AGM postulates manifest operationally.

Step 1: Initial belief storage. On February 5, the agent stored the fact "user's favorite color is blue" via `memory_ingest`, which created:

- Item: `kref://CognitiveMemory/user/favorite-color.conversation`
- Revision r_1 with metadata `{type: fact}`
- Tag `latest` $\mapsto r_1$

At this point the belief state is $\mathcal{B}(\tau_1) = \{\text{"favorite-color} = \text{blue"}\}$.

Step 2: Belief revision. On February 7, the user corrected: "favorite color is now black, not blue." The `memory_ingest` pipeline detected the existing item via fulltext search, triggering the revision path rather than creating a new item. The system:

1. Created revision r_2 on the same item with metadata `{summary: "User's favorite color is black (previously blue)"}`.
2. Re-pointed the `latest` tag: `latest` $\mapsto r_2$ (previously r_1). This is the *mutable pointer* operation that implements $K*2$ (Success): after revision,

$\psi \in \mathcal{B}(\tau_2)$.

3. Created a SUPERSEDES edge: $r_2 \xrightarrow{\text{SUPERSEDES}} r_1$. This edge records the provenance of the change and is the mechanism by which Relevance and Core-Retainment are preserved—r_1 is not deleted but is no longer the authoritative belief.

The belief state is now $\mathcal{B}(\tau_2) = \{$"favorite-color = black"$\}$, while r_1 remains accessible for provenance queries. $K*4$ (Vacuity) was not triggered because the new belief contradicted the existing one. $K*5$ (Consistency) holds because the tag re-pointing ensures only the non-contradictory revision is authoritative. $K*3$ (Inclusion) holds because $\mathcal{B}(\tau_2) \subseteq \text{Cn}(\mathcal{B}(\tau_1) \cup \{\psi\})$—the new belief state contains only the updated preference plus any unchanged beliefs.

Step 3: Downstream impact. If other memories depended on the color preference (e.g., a "room decoration plan" linked via DEPENDS_ON to the color belief), the `analyze_impact` operation would traverse the dependency graph from r_2 and surface all downstream dependents. In the deployed instance, the favorite-color item had no dependents, so the impact set was empty. For the paper revision chain described above, `analyze_impact` on v6 correctly propagated through the full DERIVED_FROM chain to surface all prior versions.

Step 4: Retrieval with conflict presentation. A subsequent query for "favorite color" returned both r_1 (blue, score 4.70) and r_2 (black, score 3.27), with r_1 ranking higher due to exact keyword match on "blue" in the fulltext index. This illustrates both a strength and a current limitation. The immutable revision model correctly preserves both beliefs with full provenance—the agent can see that the preference changed and when. However, the retrieval ranking does not yet automatically prioritize the more recent revision.

This is a concrete demonstration of why the retrieval properties (Section 8.5) are design observations rather than formal guarantees: the retrieval pipeline can surface results whose ranking contradicts the formal belief state $\mathcal{B}(\tau')$. The conflict presentation design (Section 8.6) deliberately returns both beliefs with temporal metadata, leaving resolution to the agent's reasoning layer. In the current deployment, the agent's skill prompt instructs it to prefer the most recently created memory when conflicts are detected, implementing the belief revision at the application layer. Future work could incorporate temporal recency as a retrieval signal (Section 16).

Table 18: AGM Compliance Verification (49 scenarios).

Post.	Simple	Multi	Chain	Temp.	Adv.	Pass
$K*2$	✓	✓	✓	✓	✓	100%
$K*3$	✓	✓	✓	✓	✓	100%
$K*4$	✓	✓	✓	✓	✓	100%
$K*5$	✓	✓	✓	✓	✓	100%
$K*6$	✓	✓	–	✓	✓	100%
Rel.	✓	✓	✓	✓	✓	100%
Core	✓	✓	✓	–	✓	100%
Ovrl	100%	100%	100%	100%	100%	**100%**

49 scenarios: 49 passed, 0 failed. ✓=pass; –=N/A.

The 49-scenario AGM compliance suite (Section 15.7) generalizes this single case to a systematic verification, testing each postulate across simple, multi-item, chain, temporal, and adversarial configurations—including rapid sequential revisions (10 consecutive updates to the same item) and case-variant values that stress-test the SUPERSEDES mechanism at scale.

15.7 AGM Compliance Verification

To empirically validate the formal claims of Section 7, we implemented an automated compliance evaluation suite comprising 49 test scenarios across 5 categories (simple, multi-item, chain, temporal, adversarial), testing all 7 postulates claimed by the architecture: $K*2$ (Success), $K*3$ (Inclusion), $K*4$ (Vacuity), $K*5$ (Consistency), $K*6$ (Extensionality), Relevance, and Core-Retainment. Each scenario creates beliefs in a fresh graph instance, performs revision or contraction operations, and verifies postulate-specific assertions against the resulting graph state.

Table 18 reports the results: all 49 scenarios pass across all postulate–category combinations, with zero failures and zero errors. The "–" entries indicate categories where no applicable test scenario was defined (e.g., $K*6$ Extensionality is not tested in chain configurations because logical equivalence

of chains requires cross-item identity, which falls outside the postulate's scope; Core-Retainment is not tested in temporal configurations because temporal sequences do not produce the independent belief structure needed to verify minimal contraction).

The adversarial category is particularly significant: it tests edge cases such as case-variant values, long string values, rapid sequential revisions (10 consecutive revisions to the same item), similar item names ("color" vs. "colour"), idempotent revisions, deep dependency chains (A→B→C→D), and mixed edge types. All adversarial scenarios pass, confirming that the graph-native operations satisfy the formal postulates not only in idealized conditions but under stress.

These results provide empirical backing for the formal proofs in Section 7. The proofs establish that the operations *should* satisfy the postulates given the operational semantics; the compliance suite verifies that the *implementation* faithfully executes those semantics across a diverse scenario space. This is particularly important because the formal analysis operates over an idealized propositional logic $\mathcal{L}_G$, while the implementation operates over concrete Neo4j graph operations with real network latency, concurrent access, and string-based metadata—conditions where implementation drift from formal specification is common.

15.8 Dream State Consolidation

The Dream State pipeline has been validated both in deployment and during benchmark evaluation. In deployment, early runs on a small graph produced conservative outcomes (0 deprecations, 0 tag updates), correctly reflecting that a young graph with few episodic memories has limited candidates for consolidation. The circuit breaker and conservative assessment prompt avoided premature pruning—the expected behavior. More significantly, the safety guards operated as designed during the LoCoMo benchmark evaluation: the consolidation pipeline processed all LoCoMo conversation transcripts into summaries without triggering the circuit breaker's `max_deprecation_ratio` threshold (set to 0.5), and items tagged `published` were protected from deprecation regardless of the LLM assessor's recommendations. No manual intervention was required to prevent the consolidation pipeline from corrupting the evaluation data—the architectural safety mechanisms (Section 9) were sufficient.

15.9 Limitations

Scale. The deployment graph used for case studies remains small (tens of items). We cannot extrapolate retrieval quality to graphs with thousands or millions of items without further testing; performance may degrade as the ratio of relevant to irrelevant items decreases.

Cross-system comparison methodology. The LoCoMo token-level F1 results (Section 15.2) use the same scoring function as competing systems (Zep, Mem0, Memobase), enabling direct comparison on the official metric. However, competitor scores are sourced from their respective publications, not from controlled re-evaluation on identical infrastructure and LLM configurations. Differences in underlying LLM, prompt engineering, and evaluation methodology may account for some variance; a fully controlled comparison using the same LLM, prompt template, and hardware is needed to isolate architectural contributions from confounds.

Evaluation scope complementarity. LoCoMo evaluates end-to-end recall quality (can the system answer questions about past conversations?) but does not directly exercise the formal belief revision machinery—its questions do not require the system to detect contradictions, supersede prior beliefs, or propagate impact through dependency chains. The AGM compliance suite (Section 15.7) fills this gap by testing the graph-level operations in isolation, but does not measure whether those operations improve downstream agent behavior. The two evaluation approaches are therefore complementary: LoCoMo validates that the overall architecture produces correct answers; the AGM suite validates that the belief revision operations satisfy formal rationality constraints. A benchmark that combines both—requiring the agent to answer questions whose correctness *depends* on having performed correct belief revision (e.g., MemoryAgentBench's conflict resolution tasks [2026])—is the critical missing evaluation, planned as future work.

Internal retrieval evaluation. The retrieval

observations reported in this paper are anecdotal. A comprehensive assessment would require a larger query set (100+ queries) with multiple annotators establishing ground-truth relevance labels.

Self-evaluation bias. The system is evaluated on its own deployment data during the authorship of this paper, creating an inherent circularity. We have attempted to mitigate this by reporting raw numbers without favorable interpretation and by explicitly acknowledging where the system falls short (e.g., the belief revision ranking limitation).

$\mathcal{L}_G$ expressiveness. The formal results hold for a deliberately weak propositional logic over ground triples. This logic cannot express subsumption hierarchies, role composition, disjointness axioms, or cardinality constraints—features that real knowledge graphs often need. Any strengthening of $\mathcal{L}_G$ toward richer logics would re-encounter Flouris-type impossibility results. The formal contribution is thus scoped: it demonstrates that AGM belief revision is achievable for graph-native memory systems at the propositional level, but does not extend to more expressive representations.

Formal scope. The primary formal claim covers $K*2$–$K*6$ plus Relevance and Core-Retainment. The supplementary postulates ($K*7$, $K*8$) remain open—establishing them requires constructing an entrenchment ordering or proving the system's operations encode a transitively relational contraction. Additionally, the postulates are proved for $\mathcal{B}(\tau)$ (the full belief base determined by tag assignments), not for the specific subset surfaced by the hybrid retrieval pipeline, which introduces score-based non-determinism.

Dream State LLM dependency. The consolidation pipeline's quality depends entirely on the LLM's assessment accuracy. Incorrect deprecation recommendations, despite safety guards, could degrade the memory graph over time. The circuit breaker mitigates catastrophic failures but cannot prevent gradual quality erosion from consistently biased assessments.

System performance. We do not report latency distributions, throughput measurements, or memory overhead per belief. Basic system metrics are needed to substantiate the architectural claims.

16 Future Directions

16.1 Evaluation Roadmap

LoCoMo-Plus extensions: The LoCoMo-Plus evaluation (Section 15.3, n=401, 93.3%) demonstrates that prospective indexing and event extraction largely solve the retrieval problem (98.5% recall accuracy), with the remaining failures concentrated in answer model reasoning. Three planned extensions target the remaining weaknesses. *Goal-type accuracy*: the 85% accuracy on goal-type questions (abstract intention inference) motivates investigation of goal-aware prospective indexing—specifically generating implications for stated goals and intentions, not just events, to create better semantic bridges for the most abstract constraint type. *Chronological context ordering*: currently, recalled memories are presented to the answer model in relevance-score order; sorting by chronological order may help the model reason about temporal progressions (e.g., "saved for ring 6 months ago → now complaining about car key costs"), particularly for goal-type constraints where the narrative arc matters. *Ablation study*: isolating the individual contributions of event extraction, prospective indexing, sibling relevance filtering, and graph-augmented recall by running four configurations (summary-only, +events, +implications, full system) on the same 401-entry benchmark. The pre-enrichment baseline (61.6% on n=200) establishes the lower bound; the full system (93.3%) establishes the upper bound. The hypothesis: events and implications are complementary—events preserve factual anchors (what happened); implications provide semantic bridges (what it means for the future). Neither alone is sufficient for the highest accuracy. *Stronger answer models*: with 98.5% recall accuracy, the theoretical ceiling is ∼99%. A model with stronger causal reasoning and less tendency to fabricate on correctly retrieved context would close the remaining gap—the architecture requires no changes, only a model swap. Additional benchmarks including MemBench, Mem2ActBench, MEMTRACK, and PERSONAMEM will establish our system's position across diverse evaluation dimensions.

LongMemEval [2025]: Temporal reasoning

and knowledge update evaluation. **MemoryAgentBench** [2026]: Four-competency evaluation framework (accurate retrieval, test-time learning, long-range understanding, conflict resolution) with EventQA and FactConsolidation datasets—the conflict resolution competency is directly relevant to our AGM-grounded revision operators.

Controlled cross-system comparison: Re-evaluating competitor systems (Graphiti, Mem0, Hindsight) on identical infrastructure and LLM configurations to isolate architectural contributions from confounds in the LoCoMo results. The token-level F1 results (Section 15.2) partially address this by enabling metric-comparable evaluation against systems that also report F1.

Retrieval ablation studies: Isolating each retrieval branch's contribution. Sensitivity analysis for $\beta \in [0.5, 1.0]$, type weights, and comparison against RRF and convex combination.

16.2 Formal Extensions

Entrenchment ordering for $K*7/K*8$: Constructing an explicit epistemic entrenchment ordering over graph triples, or proving that tag-based contraction is equivalent to a transitively relational selection function, would complete the formal picture. We identify three candidate sources for such an ordering: (a) temporal recency of the revision containing the belief (more recent beliefs are more entrenched), (b) in-degree in the dependency graph (beliefs depended upon by many others are more entrenched), and (c) confidence metadata attached to revisions by the Dream State pipeline. As discussed in Section 7.2, the key challenge is that no single ordering is appropriate for all belief types. We conjecture that a *type-dependent entrenchment function*—where the ordering criterion varies by belief kind (temporal recency for preferences, evidential support for facts, confidence score for inferred beliefs)—would satisfy $K*7/K*8$ for the common case of single-type revisions while requiring careful treatment for cross-type belief interactions. Formalizing this conjecture requires proving that the type-restricted orderings compose into a global total preorder satisfying the Gärdenfors–Makinson conditions, or alternatively, showing that a weaker condition (a partial preorder with type-local totality) suffices for the graph-native revision operator.

Richer logics: Extending $\mathcal{L}_G$ toward fragments of description logics that remain AGM-compatible Qi et al. [2006], potentially via Aiguier et al.'s Aiguier et al. [2018] satisfaction system framework. This would enable subsumption reasoning and role composition within the belief revision framework.

Partial merge operator: Defining a formally characterized merge operator for partial belief updates within a single revision. The current whole-revision replacement strategy is clean but coarse-grained. A merge operator based on Konieczny and Pino Pérez's Konieczny and Pino Pérez [2002] belief merging framework could handle contradictory sub-claims by identifying the minimal set of atoms that conflict with the new input and replacing only those, while preserving unchanged co-located beliefs. The key challenge is ensuring that such an operator satisfies the AGM postulates—particularly Relevance, which requires that only beliefs relevant to the new input are affected.

16.3 System Extensions

Adaptive Consolidation: Urgency-based triggering; agent-initiated consolidation. Extending the Dream State with anticipatory pre-computation—pre-reasoning about likely future queries over the graph structure (e.g., pre-computing AnalyzeImpact cascades for recently revised beliefs)—following Letta's sleep-time compute approach [2025].

Temporal Recency in Retrieval: Incorporating recency as a third retrieval signal alongside full-text and vector scoring.

17 Conclusion

We have presented Kumiho, a graph-native cognitive memory architecture whose structural primitives—immutable revisions, typed edges, mutable tag pointers, URI addressing—simultaneously serve as the operational infrastructure for managing agent work products. Agents use the same graph to remember past interactions and to version, locate, and build upon each other's outputs—enabling fully autonomous multi-agent pipelines without separate

asset tracking systems. Several individual components exist in concurrent systems; we do not claim novelty for them individually. The contribution is their architectural synthesis, grounded in formal belief revision analysis and the recognition that cognitive memory and work product management share identical structural requirements.

The core formal result is a correspondence between the AGM belief revision postulates and graph-native memory operations, framed at the belief base level following Hansson Hansson [1999]. We proved satisfaction of the basic rationality postulates ($K*2$–$K*6$) and Hansson's belief base postulates (Relevance, Core-Retainment), provided a principled rejection of Recovery grounded in provenance preservation, and identified the supplementary postulates ($K*7$, $K*8$) as an open question requiring construction of an entrenchment ordering. We addressed why the Flouris et al. impossibility results do not apply to our property graph formalism.

Beyond the formalism, the architecture contributes: (i) a unified graph where cognitive memory primitives simultaneously serve as operational asset management infrastructure—agents use the same graph to remember and to manage each other's work products in multi-agent pipelines (architecturally enabled and deployed in production, though multi-agent pipeline evaluation is planned as future work); (ii) SDK transparency enabling both multi-agent coordination (agents query the graph to find inputs) and human governance (operators audit agent beliefs through the same API); (iii) safety-hardened consolidation with guard mechanisms (circuit breakers, dry-run validation, published-item protection) not found in the concurrent systems we surveyed; (iv) a URI-based addressing scheme unique among agent memory systems; and (v) a BYO-storage artifact model whose privacy-preserving compression is simultaneously a robustness mechanism.

Empirically, on the LoCoMo benchmark [2024] (official token-level F1 with Porter stemming), Kumiho achieves 0.447 four-category F1 (n=1,540)—the highest reported score across the retrieval categories. Separately, adversarial refusal accuracy reaches 97.5% (n=446), demonstrating production-grade hallucination resistance. This near-perfect adversarial score is a natural consequence of the belief revision architecture: the memory graph genuinely does not contain fabricated information—immutable revisions preserve only what was actually discussed—so there is nothing for the model to hallucinate from. The combination of top-tier retrieval F1 and near-perfect hallucination resistance is the central empirical result. Including adversarial binary scoring, overall F1 is 0.565 (n=1,986). On LoCoMo-Plus [2026]—a Level-2 cognitive memory benchmark testing implicit constraint recall under cue-trigger semantic disconnect—Kumiho achieves 93.3% judge accuracy (n=401) with 98.5% recall accuracy, outperforming the best published baseline (Gemini 2.5 Pro, 45.7%) by 47.6 percentage points while using GPT-4o-mini for bulk operations at a total cost of ∼$14. Three architectural innovations drive both results: *prospective indexing* (LLM-generated future-scenario implications indexed alongside summaries), *event extraction* (structured events with consequences preserving causal detail), and *client-side LLM reranking* (Section 8.8), where the consuming agent's own LLM selects the most relevant sibling revision from structured metadata at zero additional cost. The enrichments drove LoCoMo-Plus accuracy from a pre-enrichment baseline of 61.6% to 93.3%, eliminating the >6-month accuracy cliff (37.5% → 84.4%). The architecture is model-decoupled: the 98.5% recall accuracy is invariant across answer models, while end-to-end accuracy scales from ∼88% (GPT-4o-mini) to 93.3% (GPT-4o), with the gap concentrated in goal-type constraints requiring abstract reasoning. Of the 27 failures, 78% are answer model fabrication on correctly retrieved context—demonstrating that the memory layer has effectively solved the retrieval problem, and the remaining gap is a consumer-side reasoning challenge. Where all tested baselines—including premium models with million-token context windows—achieve 23–46%, the graph-native primitives (typed edges, structured summarization, prospective indexing, sibling relevance filtering, multi-query reformulation, client-side LLM reranking) provide the structural reasoning capabilities that surface-level retrieval cannot substitute.

Automated AGM compliance verification (49 scenarios, 100% pass rate) confirms that the implemen-

tation faithfully executes the formal specification. Controlled cross-system comparison, consolidation enrichment ablation, and additional benchmarks (LongMemEval, MemoryAgentBench) remain important directions for future work.

In an era where AI agents perform consequential work—producing artifacts, making decisions, and collaborating autonomously in multi-agent pipelines—their memory must serve double duty: as the cognitive substrate for individual agent intelligence and as the shared operational infrastructure through which agents coordinate, build upon each other's outputs, and maintain auditable provenance chains. The seven design principles distilled from this work provide a reusable framework for building such systems: systems where every belief can be traced to its evidence, every output can be found and built upon by downstream agents, and every decision chain is open to human inspection.

References

C. E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50(2):510–530, 1985.

M. Aiguier, J. Atif, I. Bloch, and C. Hudelot. Belief revision, minimal change and relaxation: A general framework based on satisfaction systems, and applications to description logics. *Artificial Intelligence*, 256:160–180, 2018.

T. I. Aravanis. Towards machine learning as AGM-style belief change. *International Journal of Approximate Reasoning*, 183:109437, 2025.

R. C. Atkinson and R. M. Shiffrin. Human memory: A proposed system and its control processes. In *Psychology of Learning and Motivation*, volume 2, pages 89–195. Academic Press, 1968.

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

S. Baitalik, R. Datta, A. K. Das, and S. Das Choudhury. Conversation as belief revision: GreedySAT revision for global logical consistency in multi-turn LLM dialogues. In *AAAI 2026 Bridge on Logic & AI*, 2026.

S. Bruch, S. Gai, and A. Ingber. An analysis of fusion functions for hybrid retrieval. *ACM Transactions on Information Systems*, 42(1):1–35, 2023.

G. V. Cormack, C. L. A. Clarke, and S. Büttcher. Reciprocal rank fusion outperforms Condorcet and individual rank learning methods. In *Proc. 32nd International ACM SIGIR Conference*, pages 758–759, 2009.

A. Darwiche and J. Pearl. On the logic of iterated belief revision. *Artificial Intelligence*, 89(1–2):1–29, 1997.

G. Flouris, D. Plexousakis, and G. Antoniou. On applying the AGM theory to DLs and OWL. In *Proc. 4th International Semantic Web Conference (ISWC)*, pages 216–231, 2005.

A. Fuhrmann. Theory contraction through base contraction. *Journal of Philosophical Logic*, 20(2):175–203, 1991.

P. Gärdenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. MIT Press, 1988.

P. Trickel, A. Lapets, and J. Hipple. Graphiti: Building real-time, streaming knowledge graphs for AI agents. Zep AI, 2025.

S. O. Hansson. Belief contraction without recovery. *Studia Logica*, 50(2):251–260, 1991.

S. O. Hansson. *A Textbook of Belief Dynamics: Theory Change and Database Updating*. Kluwer Academic Publishers, 1999.

P. Hase, T. Hofweber, X. Zhou, E. Stengel-Eskin, and M. Bansal. Fundamental problems with model editing: How should rational belief revision work in LLMs? *Transactions on Machine Learning Research (TMLR)*, 2024.

Y. Hu, S. Liu, Y. Yue, et al. Memory in the age of AI agents. *arXiv preprint arXiv:2512.13564*, 2025.

P. Lewis, E. Perez, A. Piktus, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *NeurIPS*, 2020.

Letta Team. Sleep-time compute: Beyond inference scaling at test-time. *arXiv preprint arXiv:2504.13171*, 2025.

C. Packer, S. Wooders, K. Lin, V. Fang, and J. E. Gonzalez. Letta Context Repositories: Git-backed versioned memory for stateful agents. Letta Blog, February 2026.

A. Maharana, D.-H. Lee, S. Tulyakov, M. Bansal, F. Barbieri, and Y. Fang. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the ACL*, pages 13851–13870, 2024.

D. Wu, H. Wang, W. Yu, Y. Zhang, K.-W. Chang, and D. Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. In *ICLR*, 2025.

D. Jiang, Y. Li, G. Li, and B. Li. MAGMA: A multi-graph based agentic memory architecture for AI agents. *arXiv preprint arXiv:2601.03236*, 2026.

D. Makinson. On the status of the postulate of recovery in the logic of theory change. *Journal of Philosophical Logic*, 16(4):383–394, 1987.

Model Context Protocol Specification (2025-11-25). Anthropic, 2025. https://modelcontextprotocol.io.

Mem0 Team. Mem0: The memory layer for AI agents. https://github.com/mem0ai/mem0, 2025.

MemAgents: Memory for LLM-Based Agentic Systems. ICLR 2026 Workshop, 2026.

C. Packer, V. Fang, S. G. Patil, K. Lin, S. Wooders, and J. E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Y. Hu, Y. Wang, and J. McAuley. Evaluating memory in LLM agents via incremental multi-turn interactions. In *ICLR*, 2026.

J. Kang, M. Ji, Z. Zhao, and T. Bai. Memory OS of AI agent. In *EMNLP*, 2025.

G. Qi, W. Liu, and D. A. Bell. A revision-based approach to handling inconsistency in description logics. *Artificial Intelligence Review*, 26(1–2):115–128, 2006.

B. Rasch and J. Born. About sleep's role in memory. *Physiological Reviews*, 93(2):681–766, 2013.

S. Robertson and H. Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

T. Sumers, S. Yao, K. Narasimhan, and T. L. Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024.

E. Tulving. Episodic and semantic memory. In E. Tulving and W. Donaldson, editors, *Organization of Memory*, pages 381–403. Academic Press, 1972.

A. Vaswani, N. Shazeer, N. Parmar, et al. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017.

L. Wang, C. Ma, X. Feng, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

B. Wilie, Z. Xu, S. R. Cahyawijaya, A. Lovenia, and P. Fung. Belief-R: An LLM benchmark for belief revision in conversational reasoning. In *EMNLP*, 2024.

W. Xu, Z. Liang, K. Mei, H. Gao, J. Tan, and Y. Zhang. A-MEM: Agentic memory for LLM agents. In *NeurIPS*, 2025.

Z. Zhang, X. Bo, C. Ma, R. Li, X. Chen, Q. Dai, J. Zhu, Z. Dong, and J.-R. Wen. A survey on the memory mechanism of large language model based agents. *arXiv preprint arXiv:2404.13501*, 2024.

R. Li, Z. Zhang, X. Bo, Z. Tian, X. Chen, Q. Dai, Z. Dong, and R. Tang. CAM: A constructivist view of agentic memory for LLM-based reading comprehension. In *NeurIPS*, 2025.

C. Hu, X. Gao, Z. Zhou, D. Xu, Y. Bai, X. Li, H. Zhang, T. Li, C. Zhang, L. Bing, and Y. Deng. EverMemOS: A self-organizing memory operating system for structured long-horizon reasoning. *arXiv preprint arXiv:2601.02163*, 2026.

J. Puget Gil, E. Coquery, J. Samuel, and G. Gesquière. ConVer-G: Concurrent versioning of knowledge graphs. In *BDA 2024*, 2024.

C. Yang, C. Zhou, Y. Xiao, et al. Graph-based agent memory: Taxonomy, techniques, and applications. *arXiv preprint arXiv:2602.05665*, 2026.

J. Luo, Y. Tian, C. Cao, Z. Luo, H. Lin, K. Li, C. Kong, R. Yang, and J. Ma. From storage to experience: A survey on the evolution of LLM agent memory mechanisms. *Preprints.org*, 202601.0618, 2026.

G. Bonanno. Belief revision: The Bayesian and AGM approaches unified via Kripke–Lewis semantics. *Artificial Intelligence*, 338:104259, 2025.

H. Meng, Z. Long, M. Sioutis, and Z. Zhou. On definite iterated belief revision with belief algebras. In *IJCAI*, 2025.

C. Latimer, N. Boschi, A. Neeser, C. Bartholomew, G. Srivastava, X. Wang, and N. Ramakrishnan. Hindsight is 20/20: Building agent memory that retains, recalls, and reflects. *arXiv preprint arXiv:2512.12818*, 2025.

Y. Yu, L. Yao, Y. Xie, Q. Tan, J. Feng, Y. Li, and L. Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. *arXiv preprint arXiv:2601.01885*, 2026.

K. Wang, Y. Lin, J. Lou, Z. Zhou, B. Suvonov, and J. Li. E-mem: Multi-agent based episodic context reconstruction for LLM agent memory. *arXiv preprint arXiv:2601.21714*, 2026.

Y. Li, W. Guo, L. Zhang, R. Xu, M. Huang, H. Liu, L. Xu, Y. Xu, and J. Liu. LocomoPlus: Beyond-factual cognitive memory evaluation framework for LLM agents. *arXiv preprint arXiv:2602.10715*, 2026.

M. Fu, G. Zhang, X. Xue, Y. Li, Z. He, S. Huang, X. Qu, Y. Cheng, and Y. Yang. LatentMem: Customizing latent memory for multi-agent systems. *arXiv preprint arXiv:2602.03036*, 2026.

W.-C. Huang, W. Zhang, Y. Liang, et al. Rethinking memory mechanisms of foundation agents in the second half: A survey. *arXiv preprint arXiv:2602.06052*, 2026.

K. Ma, B. Li, Y. Tang, L. Sun, and R. Jin. CAST: Character-and-scene episodic memory for agents. *arXiv preprint arXiv:2602.06051*, 2026.

J. Chandler and R. Booth. Parallel belief revision via order aggregation. In *IJCAI*, 2025.

N. Schwind, K. Inoue, S. Konieczny, and P. Marquis. Iterated belief change as learning. In *IJCAI*, pages 4669–4677, 2025.

R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum Press, 1978.

J. P. Delgrande, P. Peppas, and S. Woltran. General belief revision. *Journal of the ACM*, 65(5):1–34, 2018.

N. Arndt, P. Naumann, N. Radtke, M. Martin, and E. Marx. Decentralized collaborative knowledge management using Git. *Journal of Web Semantics*, 54:29–47, 2018.

R. Taelman, M. Vander Sande, and R. Verborgh. OSTRICH: Versioned random-access triple store. In *Companion of the Web Conference*, pages 127–130, 2018.

M. Vander Sande, P. Colpaert, R. Verborgh, S. Coppens, E. Mannens, and R. Van de Walle. R&Wbase: Git for triples. In *LDOW Workshop at WWW*, 2013.

M. Völkel, W. Winkler, Y. Sure, S. R. Kruk, and M. Synak. SemVersion: A versioning system for RDF and ontologies. In *Proc. 2nd European Semantic Web Conference (ESWC)*, 2005.

J. Wu. Git Context Controller: Manage the context of LLM-based agents like Git. *arXiv preprint arXiv:2508.00031*, 2025.

E. A. Fox and J. A. Shaw. Combination of multiple searches. In *TREC-2*, pages 243–252, 1993.

A. Grove. Two modellings for theory change. *Journal of Philosophical Logic*, 17(2):157–170, 1988.

S. Konieczny and R. Pino Pérez. Merging information under constraints: A logical framework. *Journal of Logic and Computation*, 12(5):773–808, 2002.

M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080, 1988.

M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3–4):365–385, 1991.

Zep AI. Evaluating memory for AI agents: A comprehensive benchmark analysis. *arXiv preprint arXiv:2504.19413*, 2025.

Memobase Team. Memobase: Stateful user memory for LLM applications. Software, https://github.com/memodb-io/memobase, 2025.

D. Patel and S. Patel. ENGRAM: Effective, lightweight memory orchestration for conversational agents. *arXiv preprint arXiv:2511.12960*, 2025.

A Key Data Structures

Memory Reference URI Format:

```
kref://project/space[/sub]/
  item.kind[?r=N][&a=art]
```

Revision Node Properties:

```
{
  ref: "kref://proj/space/item.kind?r=N",
  _search_text: "concat. searchable text...",
  embedding: [float64 x 1536],
  embedding_text: "text embedded",
  embedding_updated_at: datetime,
  metadata: {
    schema, type, summary,
    topics, keywords
  },
  created_at: datetime,
  author: "agent-id"
}
```

Redis Working Memory Key Structure:

```
cogmem:{proj}:sessions:{sid}:messages
cogmem:{proj}:sessions:{sid}:metadata
cogmem:{proj}:consol_queue
```

Dream State Assessment Structure:

```
MemoryAssessment {
  revision_ref: str
  relevance_score: float  // 0.0-1.0
  should_deprecate: bool
  deprecation_reason: str
  suggested_tags: List[str]
  metadata_updates: Dict[str, str]
  related_memories: List[(ref, type)]
}
```

B Dream State Report Format

Each Dream State run produces a Markdown report stored as a revision artifact, documenting events processed, memories assessed, actions taken (deprecations, metadata updates, tags added, relationships created), and the cursor for resumption.

```
# Dream State Report -- 2026-02-04T03:00Z
**Events:** 42 | **Assessed:** 18
**Duration:** 4500ms

## Actions Taken
### Deprecated (3)
- kref://CognitiveMemory/personal/...
  -- duplicate
### Metadata Updated (7)
- kref://CognitiveMemory/work/...
  -- topics added
### Tags Added (12)
- kref://CognitiveMemory/personal/...
  -- ssh, error
### Relationships Created (5)
- source -> target (DERIVED_FROM)

## Cursor
eyJjdXJzb3IiOiIxMjM0NTY3ODkwIn0=
```