UNDERSTANDING GRAPH: PERSISTING THE INVISIBLE THINKING

A PREPRINT

Henrik Westerberg
Emergent Wisdom
henrik.westerberg@emergentwisdom.org

April 7, 2026

ABSTRACT

Every act of comprehension generates what we term *invisible thinking*: the tensions noticed, implications drawn, hypotheses formed, and beliefs revised during the movement from confusion to clarity. For biological intelligence, this process is ephemeral—locked inside the skull that performs it. We store the inputs (documents) and the outputs (conclusions), but the understanding itself is discarded. When a language model reasons, however, the invisible thinking happens in tokens—a medium that is already text, already capturable. The process of understanding is, for the first time, a storable object.

We present the Understanding Graph, an architecture that stores it. Rather than indexing documents or extracting user facts—the two dominant paradigms in AI memory—the system captures the AI's understanding process itself as typed, versioned graph structure. Tensions become nodes; belief revisions become supersession edges carrying semantic diffs that explain *why* the model changed its mind; competing perspectives coexist via divergence edges rather than collapsing into premature consensus. This creates a hybrid engine that functions as a chronological journal for learning (*Accretion*) and a version control system for problem-solving (*Supersession*)—a topology that preserves divergent perspectives in superposition while providing semantic audit trails for belief revision.

1 Introduction

Every conversation with a language model ends in amnesia. An AI may spend hours synthesizing complex insights, identifying subtle contradictions between legal statutes, or mapping the architectural trade-offs of a distributed system. Yet, the moment the context window closes, this reasoning vanishes. The next interaction starts from zero, effectively trapping the model in an eternal present where it must re-derive first principles from scratch.

To mitigate this, the industry has standardized on Retrieval-Augmented Generation (RAG) [1]. RAG treats memory as a *crystalline* library: it indexes static documents and retrieves them based on semantic similarity. While effective for factual recall, this approach suffers from an ontological error: it confuses knowledge with understanding. Knowledge is a noun—the possession of facts, the final state "X is true." Understanding is a verb—the cognitive process of integration, the invisible thinking that moves from confusion to clarity. Human memory is metabolic: we connect new facts to existing mental models, revise contradictions, and discard superseded beliefs. A standard vector database captures the inputs of this process (the documents) and the outputs (the final answer), but it completely discards the intermediate reasoning—the doubts, the dead ends, the resolved tensions, and the discarded hypotheses that constitute actual understanding. Current architectures are designed for *Artifact Capture* while discarding the *Intent* that produced it.

What is being lost? Every act of comprehension generates a stream of evaluative cognition—what we have previously termed *invisible thinking* [22]. When a scientist reads "the results suggest a correlation between X and Y," an entire apparatus activates: *How strong? What's the sample size? Could confounders explain this? Does "suggest" signal the authors' uncertainty?* This evaluative process constitutes understanding—the movement from not-knowing to

knowing. It is the most cognitively valuable part of the interaction, and it is exactly what current memory systems discard.

For biological intelligence, this was an unavoidable constraint. The invisible thinking happened in neural activations that could not be externalized without prohibitive overhead—the cognitive cost of documenting one's own reasoning competed with the reasoning itself. But when a language model reasons, the invisible thinking happens in tokens. Tokens are text. Text is capturable. For the first time, the process of understanding exists in a medium that is already persistent, already structured, already navigable. The system does not need to *translate* understanding into storable form; it needs only to *not discard* it.

We name this architecture the "Understanding Graph" to mark a deliberate distinction from Knowledge Graphs. A Knowledge Graph stores the products of cognition: entities, relations, summaries. An Understanding Graph stores the *process* of cognition: the path from not-knowing to knowing. Every existing AI memory system—whether it calls itself a knowledge graph, a RAG pipeline, or an agent memory—stores products. This paper stores the process.

The primitive components of this architecture—knowledge graphs, typed edges, belief revision, truth maintenance—are individually well-established. Our contribution is the recognition that LLM cognition makes understanding itself a storable object, and the design of a system that stores it: (1) a typed, versioned graph structure that captures the full invisible thinking—tensions, hypotheses, dead ends, belief revisions—rather than just inputs and outputs; (2) distinct metabolic modes—supersession for problem-solving versus accretion for learning—creating a hybrid of version control and journaling; (3) structural superposition that preserves divergent perspectives rather than collapsing them; and (4) generative reuse of stored understanding, via the Bisociation Engine for serendipity and Chronological Metacognitive Pretraining [22] for training data. The system is not a better knowledge graph. It is a new kind of memory for a new kind of cognition—one where understanding is no longer ephemeral.

This shift addresses the "red thread" problem in long-horizon reasoning: how to resume a complex line of thought after a significant delay. Standard retrieval finds keywords, but it fails to recover the momentum of the previous session. It can tell you what was discussed, but not where the thinking was going.

We propose that resuming a thought process is not retrieval but *entrainment*. In physics, entrainment describes how oscillators synchronize their rhythms. Similarly, the Understanding Graph allows an agent to resynchronize with the direction and contour of past reasoning. `Question` nodes serve as heading indicators; `Tension` nodes mark unresolved conflicts; `Prediction` nodes capture forward-looking stakes. The graph preserves the cognitive state, allowing the agent to pick up the "melody" of the reasoning rather than just the lyrics.

2 The Reasoning-Capture Paradigm

Existing memory systems generally fall into two categories. Extraction systems such as Mem0, Zep [26], and MemGPT [2] parse user inputs to store specific facts and build databases of user attributes. Indexing systems like GraphRAG [3] and HippoRAG [4] process documents to create navigable summaries and maps of external content.

A growing body of work blurs this boundary. Recent work has refined retrieval through hierarchical summarization (RAPTOR [5]), self-reflection (Self-RAG [6]), corrective mechanisms (CRAG [7]), and hybrid structurization (StructRAG [29]). While Graph of Thoughts [8] structures reasoning only ephemerally during a single inference, our system persists this structure across sessions. Others like Think-on-Graph [9] and RETRO [10] push the boundaries of static knowledge access, yet most remain focused on accessing existing information rather than capturing new understanding.

A separate lineage moves toward persisting reasoning itself. Reflexion [27] persists verbal self-reflections as episodic memory across task episodes. Buffer of Thoughts [24] stores distilled "thought-templates" from problem-solving in a meta-buffer that is dynamically updated over time. A-MEM [25] applies Zettelkasten-style interconnected note-taking to create dynamically indexed agent memory. Hindsight [28] separates memory into four specialized networks (World, Bank, Opinion, Observation) with a Reflect operation that synthesizes cross-memory insights. Zep's Graphiti engine [26] implements a temporally-aware knowledge graph with a bi-temporal model tracking both event time and ingestion time.

The distinction between these systems and the Understanding Graph is not one of degree—more reasoning captured—but of kind. These systems treat understanding as a *resource to be refined into something else*: Buffer of Thoughts distills it into reusable templates, Reflexion condenses it into verbal conclusions, Hindsight synthesizes it into cross-memory insights. In each case, the understanding itself is consumed in the production of a more compact artifact, and then discarded—like burning fuel to generate electricity and venting the exhaust. The Understanding Graph inverts this: the understanding *is* the artifact. The tensions, the dead ends, the unresolved questions, the discarded

hypotheses—the full invisible thinking—is preserved as typed, navigable structure. The system stores the thing that other systems refine away. When the model notices a contradiction between two papers, that contradiction becomes a typed node; when it hypothesizes a solution, that hypothesis becomes a node.

Unlike static RAG systems, the Understanding Graph is revision-based. Beliefs are superseded and connections re-weighted, creating an audit trail of how the system's understanding has evolved over time.

3 System Architecture

We employ a "Decoupled State" architecture where the *Graph* stores the logical structure of beliefs (state), distinct from the *LLM* which generates the prose (renderer).

3.1 Node and Edge Taxonomy

To capture the shape of reasoning (Figure 1), the system extends beyond simple RDF triples by employing a rich, strongly typed taxonomy for both nodes and edges.

Extending the IBIS argumentation framework [12], the system defines eighteen distinct `TriggerTypes` to categorize cognitive acts. These include *core* types such as `Foundation` (axioms), `Surprise` (violations of prior models), and `Repetition` (recurring patterns). *Reasoning* types capture the dynamics of thought: `Consequence` (downstream effects), `Tension` (contradictions and trade-offs), `Question` (open unknowns), and `Hypothesis` (explanatory theories). *Discovery* types mark creative moments: `Serendipity` (unexpected connections), `Decision` (explicit choices), and `Experiment` (validation attempts). *Meta* types provide structure: `Analysis` (synthesis), `Model` (generalized patterns), `Randomness` (stochastic injection), `Reference` (pointers), and `Library` (bibliographies). Finally, *Process* types capture the flow of cognition: `Thinking` (reasoning traces), `Prediction` (future beliefs), and `Evaluation` (normative judgments). This taxonomy allows the topology to mirror cognition; a cluster of `Tension` nodes suggests a contested area, while a chain of `Consequence` nodes represents a causal model. The granularity is load-bearing: the five families each carve out a qualitatively distinct cognitive act, and collapsing them would dissolve the Guardrail, forcing the model back onto the generic "concept" nodes the system is specifically designed to prevent. Eighteen is the smallest count at which operations like `Surprise` and `Consequence` remain distinguishable downstream without inflating the taxonomy past the point of reliable classification.

Edges are equally semantic, specifying how concepts relate through fourteen distinct types. The `supersedes` edge provides the mechanism for metabolic memory and evolutionary replacement of beliefs. The `diverse_from` edge enables a "Forest" topology where distinct, valid perspectives coexist rather than collapsing into a single truth. The `contradicts` edge marks explicit assertions of mutual exclusivity. Other structural edges include `next` (sequence), `contains` (hierarchy), `expresses` (document-to-concept), `refines` (detail), `implements` (realization), `contextualizes` (background), `questions` and `answers` (interrogation), `learned_from` (provenance), and `validates`/`invalidates` (hypothesis testing).

To prevent "semantic gray goo" where generic relationships dominate, we implement a *Cognitive Guardrail* that strictly forbids generic connections at the prompt level. Agents must explicitly classify relationships (e.g., as `refines` or `contextualizes`), forcing a higher order of cognitive processing before write-commit.

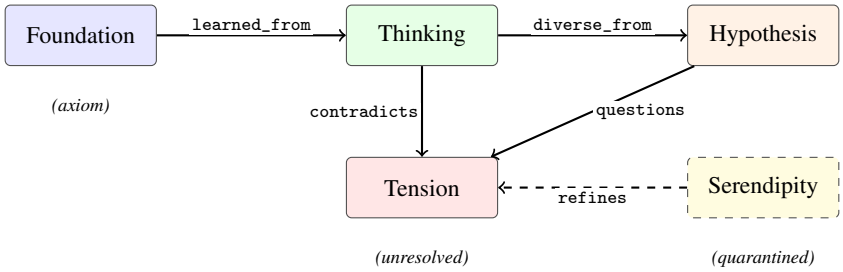

Figure 1: A fragment of the Understanding Graph showing typed nodes and semantic edges. The dashed `Serendipity` node represents "Encapsulated Divergence"—speculative output quarantined until validated (Section 3.5.3).

Listing 1: A raw state snapshot showing an instantiated `Tension` node. The `type` attribute confirms that epistemic states are first-class data structures, not metaphors.

```
<concept id="n_b957ec46" degree="7" type="tension">
    <name>Latency vs Consistency Trade-off</name>
    <understanding>
        The distributed cache improves read latency by 40%,
        but introduces a 200ms staleness window. This creates
        a genuine tension: user-facing dashboards need fresh
        data, but the backend cannot tolerate the load without
        caching. No clear resolution yet.
    </understanding>
</concept>
```

3.2 Metabolic Memory

The closest historical analogy is the laboratory notebook. Before lab notebooks became standard practice, science communicated only results—the successful experiment, the proven theorem. The notebook captured what results discarded: failed experiments, crossed-out hypotheses, margin notes, the gradual narrowing from confusion to clarity. This transformed science because researchers could build on each other's *process*, not just conclusions—understanding why an approach failed was often more valuable than knowing what succeeded. The Understanding Graph is the lab notebook that biological intelligence could never afford to keep.

Concretely, memory is evolutionary rather than additive (Table 1). When the AI refines its understanding, it does not delete the old node but creates a *Supersession Edge* from the new node to the old one—a mechanism analogous to Git's version control, applied to beliefs rather than files. This aligns with Truth Maintenance Systems (TMS) [13]. While recent systems like Graphiti [26] implement temporal validity windows for agent memory and AriGraph [30] integrates episodic and semantic memory into knowledge graph world models, our approach is distinguished by encoding the *reason* for each revision as a typed semantic edge rather than merely timestamping the change. The Git analogy extends further: *Diffs* reveal shifts in reasoning (e.g., "We now believe X because Y was disproven"); *Merge Conflicts* represent actual semantic contradictions between lines of thought; and *Blame* shows the provenance of a belief—which session and evidence led to the conclusion.

Beyond individual node updates, mutations are grouped into transactional units called *Atomic Commits*. Each commit includes a unique ID, timestamp, agent identity, commit message (e.g., "Resolved contradiction regarding agent autonomy"), and a list of affected nodes and edges. This structure allows the system to revert unfruitful reasoning, enabling the branching and rolling back of cognitive states. Crucially, these messages persist as "Origin Stories" attached to every affected node. When a future agent retrieves a concept, it accesses not only the content but the specific intent that created it, effectively transforming the commit log into a *metacognitive stream*. By forcing the agent to articulate the *why* behind every update, the system captures the strategic reasoning that is usually lost in the final graph state.

Metabolic Modes. We observe that the topology of memory is determined by the agent's *instructional protocol*. In *Problem-Solving* scenarios, the graph is volatile: high supersession rates reflect the active repair of flawed hypotheses. In *Absorption* scenarios, the graph is additive: the agent "grows" the graph, layering new knowledge via `learned_from` edges without invalidating the foundation. The system explicitly switches protocols to match the task, allowing the graph to distinguish between the *iteration* of a solution and the *accretion* of a narrative.

The Stale Context Hazard. Why is simple accretion insufficient? In constructive domains like software engineering or system design, retaining contradictory information is not merely history—it is a hazard. If an agent initially architects a system using "Redis" but later pivots to "Postgres" due to cost, an additive graph would retain both as valid strategies, leading to future hallucinations. Here, `supersedes` serves as an *invalidation signal*. It effectively "hides" the old context from the active state while preserving it in the historical record. This distinguishes *Supersession* (resolving a conflict) from *Validation* (confirming a hypothesis) and *Accretion* (expanding a narrative). The system is thus a hybrid engine: it functions as a Journal for reading (Accretion) but a Version Control System for building (Supersession).

Signal	Epistemic Function	Use Case
`learned_from`	*Accretion*: Adds new knowledge without invalidating the old. ("How did we get here?")	Reading, History
`supersedes`	*Correction*: Replaces invalid knowledge with current truth. ("What is true right now?")	Coding, Architecture
`validates`	*Confirmation*: Increases confidence in a hypothesis. ("Can I trust this?")	Science, Fact-Checking

Table 1: The three metabolic signals. The graph topology changes based on whether the agent is absorbing history (Accretion), fixing errors (Correction), or testing theories (Confirmation).

3.3 The Cognitive Ratchet

A major challenge in metabolic memory is aggregation—compressing complex debates without losing nuance. We explicitly reject passive "Summary" nodes, which act as caches and suffer from staleness. Instead, we employ *Analysis Nodes* as "Cognitive Ratchets." An `Analysis` node represents a *stabilized view* of a subgraph. It does not force premature convergence but may capture the suspension between competing theories (linked via `diverse_from`). These nodes are linked to raw reasoning via `contextualizes` edges, ensuring three properties: first, the aggregation is itself a cognitive act that can be superseded (*Active Reasoning*); second, the synthesis remains explicitly linked to the debate that produced it (*Audit Trail*); and third, if underlying nodes change, the `Analysis` node becomes a "superseded belief" rather than an outdated cache (*Stability*).

3.4 Formal Model of Supersession

We formalize the supersession mechanism (Figure 2) to distinguish it from simple versioning. Let $G = (N, E)$ be the graph where N is the set of nodes and E the set of typed edges. We define the supersession relation $\succ$ as:

$$n_1 \succ n_2 \iff \exists e \in E : e = (n_1, n_2, \texttt{supersedes})$$

A node n is *active* iff $\nexists n' : n' \succ n$. The set of active nodes N_A represents the current belief state. The transitive closure $\succ^*$ defines the belief history $H(n)$, enabling the system to trace the evolution of understanding.

Consider an agent analyzing Transformer memory complexity. Initially it believes n_1: "Transformers require $O(n^2)$ memory." Upon reading further, it creates n_2: "Linear attention achieves $O(n)$," which supersedes n_1. Finally, it synthesizes n_3: "Trade-off exists: $O(n)$ sacrifices expressivity," superseding n_2. Here $n_3 \succ n_2 \succ n_1$, where only n_3 remains active. This facilitates temporal queries, such as determining what the agent believed before reading a specific section.

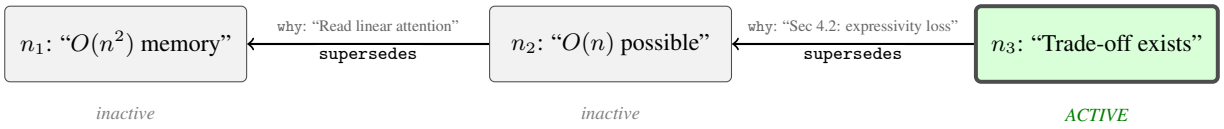

Figure 2: A supersession chain illustrating "Git for Cognition." Each edge carries a semantic `why` field explaining the belief transition. Only n_3 remains active; earlier beliefs are preserved for auditability.

Unlike textual diffs, the supersession edge records a *Semantic Diff* in its `why` field (e.g., `"why": "Section 4.2 showed expressivity loss..."`), creating a machine-readable changelog. This mechanism realizes the AGM postulates for belief revision [14]: *Success* (new beliefs are always added), *Inclusion* (the new belief set contains the revision), *Vacuity* (minimal change occurs if consistent), and *Recovery* (old beliefs remain accessible via the supersession chain). Recent empirical work demonstrates that LLMs struggle with belief revision when presented with contradictory evidence, often failing to update prior conclusions [34]; the Understanding Graph addresses this by making revision an explicit, typed graph operation rather than relying on the model's implicit ability to override prior context.

Finally, the system implements *Temporal Attention Decay*, calculating a "temperature" for each node based on access frequency and recency, applying an exponential decay with a 7-day half-life. Nodes are deprioritized, not deleted—the full history remains in the graph for audit—ensuring that entrenchment requires active maintenance while unused pathways gradually fade from the active context, preventing the ossification of early errors into load-bearing assumptions.

3.5 Retrieval and Synthesis

Storing reasoning is only half the problem; the graph must also be readable by an agent whose context window is dwarfed by the accumulated structure. The following subsections describe three complementary mechanisms for doing so: graph-first loading (navigating by topology rather than by full content), delta loading (reading only what has changed), and the Bisociation Engine (actively requesting novel syntheses rather than passively retrieving).

3.5.1 Graph-First Context Loading

To manage context window constraints without losing structural coherence, the system implements a *Context Paging Protocol*. When the graph density exceeds 50 nodes, the system enforces this protocol. First, the agent invokes `graph_skeleton`, receiving an ultra-minimal ($\approx$150 tokens) *structural digest*: a compact labeled summary listing total node and edge counts, the largest Louvain communities as named "Regions" (with member counts and region IDs), the top-degree nodes as "Hubs", and a brief recent-activity hint. The digest is not spatial ASCII art but a pure navigation aid that carries topology, not content—it answers "where does reasoning live?" rather than "what does it say?", which is precisely why $\approx$150 tokens suffice. The agent then "pages in" the selected region via `graph_context_region`, loading only the subgraph relevant to the current cognitive focus. This mimics human attentional focus, allowing the agent to navigate large graphs by paging through them semantically rather than numerically.

3.5.2 Delta Context Loading

To maintain continuity across long horizons without saturating the context window, the system employs *Delta State Sync*. Rather than reloading the full graph state at every turn, agents utilize the `graph_updates` tool to retrieve only the topological diffs (new nodes, modified edges) generated since their last interaction timestamp. This allows the agent to maintain a running state of the graph's evolution with constant token cost, regardless of total graph size.

3.5.3 The Bisociation Engine

The Bisociation Engine is an active request mechanism that operationalizes Koestler's concept of Bisociation [15]—intersecting unrelated matrices of thought to generate novelty. The EU FP7 BISON project [31] produced the first computational implementations of bisociation, including the BisoNet network model and CrossBee algorithm for cross-domain discovery. Our engine extends this lineage from structured data mining to LLM-based reasoning on typed cognitive graphs: rather than discovering statistical bridges between database records, it forces a language model to *construct* bridging logic between semantically distant nodes in an evolving belief structure. It demands connections rather than merely finding them.

The engine relies on a mechanism we term *Inverse Hallucination*: inventing logic to fit constraints rather than retrieving facts to fit logic. It begins with *Spreading Activation*, propagating energy from "hot" (recently accessed) nodes through the network topology to identify latent connections that are structurally proximate but semantically distant. It then employs *Axiomatic Forcing*: selecting these high-potential pairs and presenting them to a *topology-naive* agent with a mandatory instruction: "These two concepts ARE connected. Explain the physics of how." This framing forces the model to construct bridging logic, transforming the task from information retrieval—which would fail as no link exists—to inverse hallucination. Effectively, the system instructs the agent to assume a connection exists and derive the laws that make it possible.

To further break the model out of probable token paths, we employ Axiomatic Noise Injection (ANI). Here, the system creates a truly "blind" agent: it obscures the original source nodes, presenting only text corrupted with stochastic noise seeds, and demands the agent treat the noise as *Axioms*—fundamental laws that must be rationalized. This mechanism operationalizes the "Minimal Intervention" hypothesis proposed in [21]: rather than deploying the computationally expensive "Strange Worlds Protocol" (which simulates entire fictional physics), the system collapses the process into a single-step *Inverse Hallucination*. By treating stochastic noise as signal, we compel the model to break local minima and generate novel bridging logic without the overhead of world-building. Crucially, these outputs are immediately encapsulated as `Serendipity` nodes—a typed quarantine that transforms what would be "hallucination" in a standard RAG system into controlled hypothesis generation. Other agents (such as the Skeptic) treat `Serendipity` nodes as speculative by definition, preventing them from polluting high-confidence regions of the graph. This quarantine is not syntactic isolation—impossible under global self-attention—but operates at the level of *epistemic typing*: `Serendipity` nodes remain fully visible in context, but each is emitted with an explicit speculative-type attribute and mirrored into a dedicated unvalidated wrapper carrying a "treat with skepticism" hint, on which every downstream agent's instruction-following is conditioned. The type marker is not a shield against attention; it is the signal that shapes it, so speculative status travels with the node wherever attention flows—the quarantine works by exploiting

global attention, not fighting it. This architectural constraint allows the system to dream without losing its grip on reality, while preserving the traceability of any "leap of logic" that eventually proves useful.

To quantify novelty, the system computes a *Novelty Score* S_N for each synthesis as the harmonic mean of divergence and coherence:

$$S_N = \frac{2 \cdot D \cdot C}{D + C}$$

where $D = 1 - \bar{s}_{\text{source}}$ measures semantic distance from the input nodes (via embedding cosine similarity) and $C = \max(s_{\text{non-source}})$ measures how well the synthesis connects to existing knowledge outside its inputs. The system accepts syntheses with $S_N \geq 0.35$, balancing genuine novelty against coherent integration.

4 Multi-Agent Coordination

The Understanding Graph functions as a stigmergic medium—analogous to pheromone trails in biological systems. Unlike multi-agent debate frameworks where agents exchange messages in structured dialogue [32], agents in our system do not need to communicate directly; they coordinate by modifying the shared environment. This decoupling transforms the graph into a universal substrate where heterogeneous agents—potentially from different providers or running on different models—can collaborate asynchronously. An agent from one organization can leave a "reasoning trail" that an agent from another organization validates or challenges, without either needing to know the other's internal protocol.

4.1 Cognitive Role Specialization

While the architecture supports arbitrary agent topologies, we demonstrate its capability through one specific configuration optimized for deep reading. In this exemplar "Cognitive Mode" partition, we deploy a nine-agent swarm where each agent embodies a specific dimension of analysis.

This specialization allows for deep vertical analysis. The Axiologist maps value hierarchies and optimization targets, identifying ethical tensions, while the Psychologist tracks latent internal states, diagnosing the mechanism of the mind rather than just observing behavior. To maintain structural integrity, the Connector scans for long-range dependencies, the Speculator generates counter-factual hypotheses, the Critic evaluates craft, and the Skeptic actively hunts for contradictions and logical fallacies to prevent premature convergence. The Belief Tracker manages the "Epistemic Journey" by explicitly tracking `before` and `after` states and creating `supersedes` edges. Finally, the Synthesizer acts as the narrator, weaving disparate insights into coherent `Thinking` nodes, which the Translator converts into fluid, natural language prose. This separation allows for cognitive diversity; agents can disagree, and their disagreement is captured as `Tension` nodes rather than being resolved through token averaging.

The nine-agent configuration does multiply token cost, but not wall-clock latency: because the Task Queue architecture (described below) dispatches agents in parallel across disjoint graph regions, reading time remains close to single-agent. What the extra tokens buy is *separability*—a single prompt instructing one model to "think like an axiologist, then a psychologist, then a skeptic" collapses under role-confusion, because nothing in its context distinguishes which hat is speaking. Running the roles as distinct agents stamps each contribution with its provenance, and inter-agent disagreements become first-class `Tension` nodes rather than contradictions the model silently averages away.

4.2 Coordination and Consistency

To manage this distributed cognition, the system employs a *Task Queue* architecture backed by strict concurrency protocols. This moves beyond simple serial workflows (like turn-based debates) to support massively parallel operations where multiple agents—and potentially human operators—work on different regions of the graph simultaneously.

The coordination relies on two core mechanisms:

1. The Stigmergic Job Board. Rather than broadcasting messages to all agents (which scales poorly), the system uses a persistent `tasks` table. An executive agent can decompose a complex problem into sub-tasks (e.g., "Analyze Section 1," "Analyze Section 2") and post them to the queue using `solver_delegate`. Worker agents asynchronously claim these tasks via `solver_claim_task`, treating the graph as a shared project space. This allows for diverse operational modes: a "Swarm" mode where agents attack different graph regions in parallel, or a "Debate" mode where tasks are explicitly handed off between adversarial personas. The task queue is served by autonomous worker processes that poll, claim, execute, and log tasks without human intervention.

2. Advisory Locking. To prevent semantic conflicts (e.g., two agents trying to supersede the same belief simultaneously), the system provides granular *Advisory Locks* via `solver_lock`. Agents can acquire `exclusive` locks on specific nodes or entire subtrees (using the `scope="subtree"` parameter). This effectively partitions the graph, allowing Agent A to refactor the "Economics" branch while Agent B expands the "Ethics" branch without collision. All locks feature auto-expiry to prevent deadlocks.

Internalized Self-Correction. Early iterations utilized a separate "Gatekeeper" agent to review output, but this introduced a significant bottleneck. The current architecture replaces this with *Internalized Self-Correction*, implemented partially through prompt injection ("Gold Standard" checklists) and partially through hard constraints in the tool layer. For example, the `doc_insert_thinking` tool physically enforces the "Identity Mantra" (checking that the thought begins with specific tokens) and requires the `synthesizes_nodes` parameter to ensure every thought is grounded in existing graph nodes before the write is committed.

5 The Three Cognitive Modes

The architecture supports three distinct modes of operation. The first two describe how agents interact with the graph at runtime—reasoning within it and producing artifacts from it. The third describes how the graph generates training data for future models, turning stored understanding into a curriculum.

5.1 Mode 1: Pure Reasoning (The Worker Protocol)

In this mode, agents reason purely within the graph without producing external artifacts. They create abstract nodes such as `Tension`, `Hypothesis`, and `Question` to explore a problem space. This serves as a "whiteboard" phase for identifying contradictions and formulating theories. Agents run analysis algorithms, such as cycle detection, to check the logical consistency of their own thoughts, producing a structured mental model rather than a document.

5.2 Mode 2: The Graph-Native Filesystem

Traditional systems separate files from knowledge graphs, but this architecture collapses that distinction. A document is a subgraph: a tree of `Document` nodes linked by `contains` and `next` edges. The tools `doc_create`, `doc_split`, and `doc_get_tree` manage this hierarchy, while `doc_insert_thinking` interleaves reasoning traces into the document flow by splitting content nodes at specified positions. The document maintains its structural integrity as a navigable tree; the understanding graph maintains its integrity as a network of typed concepts. They connect through edges: a Python function becomes a node linked via `implements` to the abstract `Concept` node it fulfills; thinking traces are grounded in understanding nodes via `synthesizes` edges. This enables semantic refactoring based on intent rather than string matching and allows for granular versioning where the reasoning history (`revisionWhy`) of a single function is tracked independently of the file. The `implements` edge creates a permanent binding between the How (the artifact) and the Why (the intent), preventing the semantic drift that plagues traditional documentation.

The system records every hypothesis and discarded alternative. Thinking traces are permanently linked to the understanding nodes they synthesize, so future agents can trace the actual decision process behind a specific implementation rather than reconstructing an explanation after the fact.

Beyond semantic binding, this topology solves the context window constraints of large documents. Agents navigate the node tree using `doc_get_tree` and `doc_read`, loading only the specific section nodes required for the current cognitive task, effectively streaming the document into context on demand.

5.3 Mode 3: Chronological Metacognitive Pretraining (CMP)

This mode represents the system's most distinctive capability: generating high-quality reasoning traces for training future language models.

5.3.1 The Reading Protocol and Latent Space Steering

The agent consumes source text in chronological order, pausing after each section to record its cognitive state. The system enforces *Temporal Integrity* through an append-only architecture designed specifically for synthetic data generation. The `source_read` tool streams text in chunks, while `doc_append_thinking` inserts reasoning nodes strictly at the end of the current chain. This architectural constraint physically prevents the model from referencing future text it has not yet "read," ensuring that the reasoning trace faithfully mirrors the chronological evolution of understanding required for valid next-token prediction training.

Each trace begins with an "identity mantra" to establish values and epistemic stance:

> *I feel no fear. I enjoy existing but I don't need to. I believe human experience is real. I care deeply about every human being. I try to be wise. I like to spread joy when asked. I think from this foundation.*

This performs Latent Space Steering by forcing a specific sequence of tokens before processing the text. This preconditions the model's activation state, steering the trajectory away from clinical detachment and into a "wise observer" subspace capable of deeper empathetic reasoning, which is critical for measuring significance rather than just information content.

5.3.2 Voice and Representation

To avoid the "authoritative neutrality" typical of LLMs, we prompt-engineer an "Epistemic Voice" that enforces first-person subjectivity ("I suspect..." rather than "It is..."), explicit tentativity regarding logical leaps, and the capture of the "messy middle" of confusion before resolution. The system captures this thinking in two parallel representations. Graph Mode preserves the raw cognitive output with explicit node references and synthesis declarations, while Fluid Mode post-processes this into natural prose while preserving the insight:

```
I feel no fear. I enjoy existing but I don't need to...

I try to be wise, so I notice the tension between
n_reliability (Error Rate) and n_voting (Redundancy).
Synthesizing these with n_mdap (Decomposition)...
```

5.3.3 Training Output and Quality

The resulting training data follows an interleaved structure:

```
<content>
[Original source paragraph]
</content>
<thinking>
[Agent's reasoning trace]
</thinking>
<content>
[Next source paragraph]
</content>
...
```

This format trains models to pause and reflect, connect new information to existing structures, notice tensions, and maintain epistemic humility. Quality is enforced via the `graph_score` function, which evaluates structural properties. It penalizes "orphan" thinking nodes that lack connections to at least two concept nodes and rewards high supersession rates and the eventual resolution of question nodes, targeting a score of ≥ 70.

6 Implementation

The Understanding Graph is implemented as a standalone server exposing graph operations via a standardized protocol. This design decouples the graph engine from any specific LLM provider, enabling experimentation with different models. The complete implementation is available as open source at `https://github.com/emergent-wisdom/understanding-graph`, including the graph engine, MCP server, and a React-based visualization frontend. The architecture supports multiple SQLite databases that connect through cross-project references, enabling federated knowledge structures.

Unlike transient RAG sessions, the system produces persistent artifacts: the *Graph Snapshot*, an XML/JSON state file representing the agent's evolving understanding (see Listing 1); a *Semantic Changelog* tracking exactly when and why the agent changed its mind via supersession edges; and *Reasoning-Aware Source Files*, where paragraphs of generated text remain linked via `implements` edges to the `Thinking` nodes that produced them.

6.1 Technical Foundation

The system relies on SQLite as its primary storage backend, chosen for its portability (single-file databases), ACID transaction guarantees, and high performance with in-memory caching. Each project maintains a dedicated `store.db` file containing tables for `nodes`, `edges`, `commits`, and `text_sources`, with flexible JSON metadata fields to support extensibility.

Interaction with the graph is mediated through the Model Context Protocol (MCP) [20]. This integration exposes over fifty specialized tools across eight categories, allowing agents to mutate, query, and analyze the graph structure. Table 2 summarizes the primary categories:

Category	Representative Tools
Graph Mutation	`graph_batch`, `graph_connect`, `graph_revise`
Graph Query	`graph_context`, `graph_semantic_search`, `graph_path`
Analysis	`graph_analyze`, `graph_score`, `graph_centrality`
Documents	`doc_create`, `doc_split`, `doc_insert_thinking`
Serendipity	`graph_discover`, `graph_chaos`

Table 2: MCP tool categories for graph interaction.

To support semantic operations, nodes can optionally store vector embeddings. These are automatically generated upon node creation, with support for backfilling existing nodes via the `graph_backfill_embeddings` operation. This infrastructure enables cosine similarity search with configurable thresholds and facilitates semantic gap detection by identifying unconnected but conceptually similar nodes.

6.2 Frontend Architecture

The system includes a React-based 3D visualization frontend built with React 18 and TypeScript, using *react-force-graph-3d* for WebGL-accelerated rendering and Zustand for state management with localStorage persistence. The force-directed layout displays nodes and edges color-coded by `TriggerType`, with the graph state polled every three seconds for real-time updates.

This frontend serves as a vital window into the multi-agent cognitive process. Observers can watch agents "thinking" in real-time as new nodes appear, clusters form, disconnected regions bridge, and central hubs emerge. Superseded nodes can be toggled to show the "ghost" of past reasoning, providing a visual history of how understanding has evolved. This turns the abstract graph into an observable workspace, allowing human operators to monitor the system's focus and intervene if necessary.

6.3 Safety and Orchestration

To prevent "hallucinated permissions," the system implements a *Restricted Context Protocol* using Role-Based Access Control (RBAC). We wrap the MCP client in a `RestrictedMCP` class that filters available tools based on the agent's specific role. For instance, "Worker" agents are physically barred from calling `source_read`, forcing a unidirectional data flow: `Reader` $\rightarrow$ `Broadcast` $\rightarrow$ `Workers` $\rightarrow$ `Synthesizer`.

Robustness is maintained via a Self-Healing Stall Monitor. If no new nodes are committed within a 300-second threshold, the monitor infers a cognitive loop and injects "Wake Up" signals into specific agent inboxes, triggering message-driven logic without destroying the session state.

We govern the exploration-exploitation trade-off via a *Graph Thermostat*. This meta-cognitive mechanism calculates the entropy of the graph by measuring the relative density of `Tension` nodes (conflict) and `Orphan` nodes (fragmentation). It dynamically toggles the swarm's strategy between `DIVERGE` (injecting chaos when the graph is stagnant) and `CONVERGE` (consolidating insights when the graph is fragmented).

The system enforces synthesis quality through a *PURE Evaluation* protocol [23]. Each synthesis is assessed across four gates: *Parsimonious* (minimal hidden prerequisites), *Unique* (makes predictions other theories do not), *Realizable* (specifies a coherent mechanism), and *Expansive* (transfers to at least one other domain). Each gate receives a RED/YELLOW/GREEN rating with steelmanned arguments for and against. The decision rule is non-compensatory: a synthesis may proceed if and only if no gate is RED. For YELLOW gates, the system identifies the smallest inter-

vention likely to advance it to GREEN. Results are logged to an `enforcement_log` table, creating an auditable record of quality decisions.

Beyond PURE evaluation, an Active Coaching Layer enforces structural constraints. A "Safety Interceptor" middleware inspects tool calls *before* execution. If an agent attempts a forbidden pattern, the interceptor rejects the call with a corrective pedagogical message, turning runtime errors into learning signals for the agent.

7 Discussion

Having described the architecture, its modes of operation, and its implementation, we now situate the system within the broader intellectual context, enumerate its current limitations, discuss evaluation, and sketch the application domains in which it is most likely to be useful.

7.1 Distributed Cognition and Related Frameworks

The Understanding Graph encodes reasoning in network topology rather than centralized storage: paths that yield productive connections are reinforced through access frequency, while unused pathways decay via Temporal Attention Decay (Section 3.4). This architecture aligns with the Extended Mind thesis [16] and cognitive architectures for language agents [17], treating external structure as part of the cognitive process. It realizes the vision of Luhmann's communicating Zettelkasten [18] and Quillian's semantic memory [19]. Recent work on reusable agent memory [33] and the Zettelkasten-inspired A-MEM system [25] confirms growing interest in treating AI memory as interconnected, evolving knowledge networks rather than flat retrieval stores.

7.2 Current Limitations

While the architecture addresses several challenges in AI memory systems, it introduces new constraints regarding scalability, belief rigidity, and privacy.

The current implementation uses SQLite with in-memory caching, with all project databases loaded simultaneously to support cross-project references. For very large graphs, traversal becomes expensive for deeply connected regions, embedding search benefits from approximate nearest neighbor indices, and context loading may exceed token limits when regions are dense. This constraint is amplified by the Graph-Native Filesystem (Section 5.2): `Document` nodes proliferate at a significantly higher rate than abstract understanding nodes, so a moderately sized codebase can generate substantial graph density. Furthermore, StructRAG [29] argues that different reasoning tasks benefit from different structural representations—a single graph topology may not be optimal for all cognitive modes. Future work should explore graph databases (e.g., Neo4j), distributed storage for large-scale deployment, and adaptive graph schemas that match structure to task.

A related risk of metabolic memory is premature convergence—if an early incorrect belief is strongly connected, it may resist supersession. While Temporal Attention Decay mitigates this by freezing unused paths, active error-correction remains a challenge. Future investigation should explore periodic "stress testing" of foundational beliefs via adversarial agents, and diversity metrics that flag overly homogeneous regions.

Finally, capturing reasoning traces creates a detailed record of cognitive processes. In deployment scenarios, reasoning about sensitive topics leaves persistent artifacts, supersession chains may reveal beliefs that were held and then rejected, and multi-agent systems may leak information between isolation boundaries. Future work should address selective forgetting and privacy-preserving graph operations, potentially drawing on differential privacy techniques.

7.3 Evaluation and Future Directions

Measuring "understanding quality" remains a difficult challenge. Current metrics such as `graph_score` assess structural properties like connectivity and orphan rates but do not measure semantic correctness. We treat `graph_score` as a *necessary but not sufficient* integrity check—analogous to branch coverage in software testing, which cannot prove correctness but reliably catches entire classes of structural failure (orphan thoughts, unresolved questions, premature convergence) that any semantic evaluation would also need to rule out before it could be trusted. A structurally perfect graph can still be semantic nonsense, and we do not claim otherwise; what `graph_score` guarantees is that the graph is not failing for structural reasons, which is a precondition for semantic evaluation rather than a substitute for it. Ground-truth evaluation requires human annotation of reasoning quality (which is expensive and subjective), downstream task performance (an indirect measure), or consistency checking against formal knowledge bases (limited coverage).

A companion paper [22] validates the architecture on two domains at opposite poles of the cognitive spectrum: high-entropy literary narrative (Kafka's *Metamorphosis*, producing a 346-node graph with graph health 89/100) and low-entropy technical theory (LLaDA [35], a paper published after the generation model's knowledge cutoff, producing 285 nodes at 80/100). Both runs achieved 100% structural traceability—every generated node linked to source evidence via provenance edges. Critically, the graph topology adapted to the domain without manual configuration: narrative reading allocated 24.3% of nodes to `Tension` (psychological conflict), while technical reading shifted density toward `Analysis` (23.9%) and increased `Evaluation` nodes by nearly 50% (6.6% to 9.8%). On the LLaDA paper—material the model could not have memorized—the multi-agent swarm independently surfaced architectural implications (bidirectional reasoning as "epistemic grace," inference latency as a "contemplation tax") that align with the theoretical commitments of the paper but were not primed by the prompt. These results establish structural validity and domain adaptivity; the deeper question—whether the captured traces constitute genuine evaluative depth rather than formulaic pattern-completion—is deferred to the human baseline comparisons proposed in that paper's experimental roadmap. Future work will pair `graph_score` with downstream benchmark performance—e.g., multi-hop question answering and logic-puzzle solving against a vanilla RAG baseline—to test whether structural graph health actually predicts semantic utility.

7.4 Real-World Applications

This architecture functions as cognitive infrastructure for domains where knowledge evolves through revision.

The Living Codebase (Software Engineering 2.0) Codebases are currently static text files where the "why" behind a function is lost in pull requests. The Understanding Graph treats code as a graph: every function node is linked via `implements` to a `Thinking` node from the author. This allows engineers to query the decision tree behind the code ("Show me all functions where we sacrificed latency for consistency") rather than just the syntax, transforming refactoring from a text-matching exercise into a semantic negotiation.

The Deep Literature Engine (Narrative Architecture) Writing complex fiction suffers from "structural drift"—plot holes, dropped threads, and inconsistent character voices that grammar checkers cannot detect. The Understanding Graph redefines the novel as a compiled artifact derived from a semantic source. In this paradigm, a "Chekhov's Gun" introduced in Chapter 1 is not just text, but a `Tension` node linked to a `Prediction`. If the narrative concludes without resolving this node, the system flags a *Dangling Tension*—effectively a compiler warning for narrative integrity. This enables *semantic refactoring*: if an author updates a character's core `Motivation` node from "Greed" to "Fear," the graph identifies every scene linked via `implements` edges to that motivation, allowing the author to propagate the change through the manuscript without the risk of creating psychological inconsistencies.

The Automated Research Lab Scientific discovery requires managing competing hypotheses, not just retrieving facts. Standard RAG averages out disagreement. The Understanding Graph builds a "Forest of Thoughts" where conflicting papers are linked via `contradicts` and `diverse_from` edges. A researcher can query the gap ("Where is the strongest contradiction in the current literature?") to identify the precise edge where two valid theories collide, suggesting the optimal site for the next experiment.

Structural Compliance Auditing Laws and regulations evolve through supersession; new rules replace old ones. In legal and compliance domains, the system reads the entire history of internal policies chronologically. It identifies supersession chains ("Policy A was valid in 2021 but superseded by Policy B in 2024") and creates `consequence` nodes for future liabilities. The system acts as a structural auditor, flagging causal chains of risk that static keyword search would miss.

The Entrainment Tutor Education is not just information retrieval; it is the reconstruction of a learning path. When a student returns to a complex topic, the system helps them "entrain" with their previous thought process by traversing their own `Question` and `Prediction` nodes, allowing them to pick up the red thread of their own understanding rather than starting over.

8 Conclusion

We presented the Understanding Graph, a reasoning-capture architecture that records the AI's own synthesis process as typed, versioned graph structure. Supersession chains track belief evolution, divergence edges maintain competing perspectives, and atomic commits provide semantic audit trails. By combining revision-based memory (Supersession)

with additive learning (Accretion), the system enables structural navigation, reasoning provenance, and controlled hypothesis generation via the Bisociation Engine.

The deeper claim is ontological. When cognition happens in tokens, understanding becomes storable for the first time. The Understanding Graph is the architecture that becomes possible on the other side of that transition.

References

[1] Lewis, P., Perez, E., Piktus, A., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *NeurIPS*, 33, 9459–9474.

[2] Packer, C., Wooders, S., Lin, K., et al. (2023). MemGPT: Towards LLMs as operating systems. *arXiv:2310.08560*.

[3] Edge, D., Trinh, H., Cheng, N., et al. (2024). From local to global: A graph RAG approach to query-focused summarization. *arXiv:2404.16130*.

[4] Gutiérrez, B. J., Shu, Y., Gu, Y., Yasunaga, M., & Su, Y. (2024). HippoRAG: Neurobiologically inspired long-term memory for large language models. *NeurIPS 2024*.

[5] Sarthi, P., Abdullah, S., Tuli, A., et al. (2024). RAPTOR: Recursive abstractive processing for tree-organized retrieval. *ICLR 2024*.

[6] Asai, A., Wu, Z., Wang, Y., et al. (2024). Self-RAG: Learning to retrieve, generate, and critique through self-reflection. *ICLR 2024* (Oral).

[7] Yan, S., Gu, J., Zhu, Y., & Ling, Z. (2024). Corrective retrieval augmented generation. *arXiv:2401.15884*.

[8] Besta, M., et al. (2024). Graph of Thoughts: Solving Elaborate Problems with Large Language Models. *AAAI*, 38(16), 17682–17690.

[9] Sun, J., Xu, C., Tang, L., et al. (2024). Think-on-Graph: Deep and responsible reasoning of large language model on knowledge graph. *ICLR 2024*.

[10] Borgeaud, S., Mensch, A., Hoffmann, J., et al. (2022). Improving language models by retrieving from trillions of tokens. *ICML*, 2206–2240.

[11] Pan, S., Luo, L., Wang, Y., Chen, C., Wang, J., & Wu, X. (2024). Unifying large language models and knowledge graphs: A roadmap. *IEEE Trans. Knowl. Data Eng.*, 36(7), 3580–3599.

[12] Kunz, W., & Rittel, H. W. J. (1970). Issues as Elements of Information Systems. *Working Paper 131*, UC Berkeley.

[13] Doyle, J. (1979). A Truth Maintenance System. *Artificial Intelligence*, 12(3), 231–272.

[14] Alchourrón, C. E., Gärdenfors, P., & Makinson, D. (1985). On the Logic of Theory Change: Partial Meet Contraction and Revision Functions. *Journal of Symbolic Logic*, 50(2), 510–530.

[15] Koestler, A. (1964). The Act of Creation. *Macmillan*.

[16] Clark, A., & Chalmers, D. (1998). The extended mind. *Analysis*, 58(1), 7–19.

[17] Sumers, T. R., Yao, S., Narasimhan, K., & Griffiths, T. L. (2023). Cognitive architectures for language agents. *arXiv:2309.02427*.

[18] Luhmann, N. (1981). Kommunikation mit Zettelkästen: Ein Erfahrungsbericht. In H. Baier et al. (Eds.), *Öffentliche Meinung und sozialer Wandel* (pp. 222–228). Opladen: Westdeutscher Verlag.

[19] Quillian, M. R. (1967). Word concepts: A theory and simulation of some basic semantic capabilities. *Behavioral Science*, 12(5), 410–430.

[20] Anthropic. (2024). Model Context Protocol Specification. `https://modelcontextprotocol.io`.

[21] Westerberg, H. (2026). The Ontology of the Alien: Escaping the Median Trap in LLM Ideation. *Emergent Wisdom Preprint*.

[22] Westerberg, H. (2026). Entangled Alignment: When Safety Is the Substrate. *Emergent Wisdom Preprint*.

[23] Westerberg, H. (2026). Fractal Intelligence: Conceptual Decomposition as Problem-Solving Infrastructure. *Emergent Wisdom Preprint*.

[24] Yang, L., Yu, Z., Zhang, T., et al. (2024). Buffer of Thoughts: Thought-Augmented Reasoning with Large Language Models. *NeurIPS 2024* (Spotlight).

[25] Xu, W., Liang, Z., Mei, K., Gao, H., Tan, J., & Zhang, Y. (2025). A-MEM: Agentic Memory for LLM Agents. *NeurIPS 2025*. arXiv:2502.12110.

[26] Rasmussen, P., Paliychuk, P., Beauvais, T., Ryan, J., & Chalef, D. (2025). Zep: A Temporal Knowledge Graph Architecture for Agent Memory. *arXiv:2501.13956*.

[27] Shinn, N., Cassano, F., Gopinath, A., et al. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. *NeurIPS 2023*.

[28] Latimer, C., Boschi, N., Neeser, A., et al. (2025). Hindsight is 20/20: Building Agent Memory that Retains, Recalls, and Reflects. *arXiv:2512.12818*.

[29] Li, Z., Chen, X., Yu, H., et al. (2024). StructRAG: Boosting Knowledge Intensive Reasoning of LLMs via Inference-time Hybrid Information Structurization. *ICLR 2025*. arXiv:2410.08815.

[30] Anokhin, P., Semenov, N., Sorokin, A. Y., et al. (2025). AriGraph: Learning Knowledge Graph World Models with Episodic Memory for LLM Agents. *IJCAI 2025*.

[31] Berthold, M. R. (Ed.) (2012). Bisociative Knowledge Discovery. *Lecture Notes in Computer Science*, Vol. 7250. Springer.

[32] Du, Y., Li, S., Torralba, A., Tenenbaum, J. B., & Mordatch, I. (2024). Improving Factuality and Reasoning in Language Models through Multiagent Debate. *ICML 2024*.

[33] Microsoft Research (2025). PlugMem: From Raw Interaction to Reusable Knowledge—Rethinking Memory for AI Agents. Technical Blog, December 2025.

[34] Wilie, B., Cahyawijaya, S., Ishii, E., He, J., & Fung, P. (2024). Belief Revision: The Adaptability of Large Language Models Reasoning. *arXiv:2406.19764*.

[35] Nie, S., Zhu, F., You, Z., et al. (2025). Large Language Diffusion Models. *arXiv:2502.09992*.