

# Bridging Semantic and Relational Spaces: A Dual-Layer Architecture for Neuro-Symbolic Agent Memory

Mustafa Arslan<sup>1</sup>

<sup>1</sup>Independent Researcher, Istanbul, Turkey

## Abstract

Large Language Models are fundamentally constrained by the quadratic cost of self-attention and the empirically observed degradation of reasoning as context windows expand, a failure mode known as the “Lost in the Middle” phenomenon. Existing Retrieval-Augmented Generation (RAG) architectures treat agent memory as an unstructured bag of embeddings, leading to *Vector Haze*: the retrieval of semantically similar yet episodically disconnected facts that confuse the agent rather than aid it. This paper proposes a dual-layer Cognitive Operating System that redefines memory as a managed, structured resource. **Layer 1**, the Micro-Graph Kernel (Aeon), formalizes episodic memory as a neuro-symbolic Directed Acyclic Graph (DAG) with formally defined temporal, causal, and referential edge families, and introduces the *Semantic Lookaside Buffer* (SLB), a predictive caching mechanism that exploits conversational locality to achieve  $O(K)$  retrieval complexity against a buffer of  $K$  centroids, bypassing the  $O(\log N)$  cost of global index traversal. **Layer 2**, the Macro-Graph Router (PandoraLM), implements a dual-process cognitive router inspired by Kahneman’s *Thinking, Fast and Slow*, dynamically selecting between rapid vector search and deliberate multi-hop graph reasoning. The paper provides formal complexity analyses for each retrieval mode, demonstrates through a concrete scenario how AST-aware graph retrieval resolves polymorphic inheritance chains that defeat flat chunking, details a deterministic ReBAC security model that eliminates Time-of-Check to Time-of-Use vulnerabilities in agentic pipelines, and proposes a biologically inspired *Memory Consolidation* process (the migration of transient episodic traces into the persistent spatial index, termed the *Memory Palace*) with explicit graph-pruning invariants for bridging episodic traces with enterprise knowledge graphs.

## Keywords

Neuro-symbolic AI, Agent memory, Graph-based retrieval, Cognitive routing, Episodic DAG, Semantic caching

## 1. Introduction: Navigating the Vector Haze

The current paradigm of Retrieval-Augmented Generation (RAG) is encountering a structural ceiling. While Large Language Models (LLMs) have demonstrated remarkable generative capabilities, their capacity to maintain a coherent “self” over thousands of interaction turns is fundamentally limited by how they access information. Standard retrieval mechanisms, here characterized as *Flat RAG*, treat an agent’s history as a featureless plane: a “bag of vectors” where every query is an independent event, disconnected from the temporal and causal lineage of the task [1].

This statelessness leads to a phenomenon defined herein as **Vector Haze**<sup>1</sup>: the retrieval of semantically similar but episodically disjointed facts that confuse the agent rather than aid it [2]. In long-horizon, high-stakes technical environments, Vector Haze manifests as the agent losing the “narrative arc” of a project, failing to remember why a specific architectural decision was made three sessions ago, or hallucinating outdated API signatures because they appear semantically “closer” in a high-dimensional space [3]. Research on context utilization has confirmed that LLM reasoning degrades significantly when relevant information is buried within an expanded context window, further underscoring that models cannot simply “attend to everything” but must *structurally select* context [3].

This failure is not merely an engineering inconvenience; it is a theoretical limitation of treating similarity as the sole proxy for relevance. Consider a software debugging session: a code snippet from

---

Workshop on Graphs Across AI, 2026

✉ mustafarslan35@gmail.com (M. Arslan)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

<sup>1</sup>The complete Aeon system, including the high-performance kernel implementation with hardware-specific optimizations (e.g., SIMD-accelerated search, zero-copy memory bridges), is described in the full technical paper (arXiv:2601.15311) [2]. This discussion paper explicitly isolates the neuro-symbolic graph topology and proposes its application-layer orchestration.

**Table 1**

Comparison of Retrieval Paradigms for Agent Memory

Type	Representation	Navigation	Fidelity
Flat Vector	Bag of Embeddings	Distance (ANN)	Low (Vector Haze)
Episodic DAG	Causal/Temporal Nodes	Edge Traversal	High (Causal)
Knowledge Graph	Relational Entities	Multi-hop Reasoning	High (Thematic)
AST Map	Structural Code Units	Hierarchical Nav.	Precise (Logical)

a different project may be semantically identical to the current file (sharing function names, variable patterns, and docstring vocabulary) yet bear zero causal relevance to the bug at hand. Without structural edges that encode *how* and *why* that snippet entered the agent’s history, the retrieval system cannot distinguish it from the genuinely relevant code. As recent work on continuum memory has argued, memory for autonomous agents must be a *continuously evolving substrate* that mutates and consolidates, rather than a stateless lookup table reconstructed from scratch at every turn [4].

This paper proposes a paradigm shift: the treatment of agent memory as a resource managed by a neuro-symbolic **Cognitive Operating System**. The proposed architecture is composed of two synergistic layers:

- **Layer 1, the Micro-Graph Kernel (Aeon):** Formalizes episodic memory as a Directed Acyclic Graph (DAG) with typed edges preserving temporal and causal structure [2]. It introduces the *Semantic Lookaside Buffer* (SLB), a predictive caching mechanism that exploits conversational locality to bypass retrieval latency.
- **Layer 2, the Macro-Graph Router (PandoraLM):** Implements a dual-process cognitive router inspired by Kahneman’s *Thinking, Fast and Slow* [5], dynamically determining when to rely on rapid System 1 vector retrieval versus slow, deliberate System 2 graph reasoning over global knowledge structures and structural codebase ASTs [6].

By bridging high-performance graph topologies with adaptive cognitive routing, this dual-layer OS provides a deterministic navigational overlay to probabilistic vector spaces. Table 1 summarizes the landscape of retrieval paradigms this architecture bridges.

The contributions of this paper are threefold: (1) a formal specification of the Trace episodic DAG with typed edge families and complexity analysis of graph-based versus vector-based retrieval; (2) the Semantic Lookaside Buffer with theoretical grounding in the Semantic Inertia hypothesis and an explicit  $O(K)$  versus  $O(\log N)$  complexity comparison; and (3) a dual-process cognitive routing architecture with concrete failure-mode analysis for polymorphic code resolution, deterministic ReBAC security guarantees, and a graph-pruning consolidation algorithm with formally stated invariants.

The remainder of this paper is organized as follows. Section 2 formalizes the Trace as an episodic graph memory. Section 3 presents the Semantic Lookaside Buffer as a predictive graph cache. Section 4 details the cognitive routing and enterprise graph integration of Layer 2. Section 5 discusses the synergy between layers and proposes a “Dreaming” consolidation process, and Section 6 concludes with open questions.

## 2. The Trace: Episodic Memory as a Directed Acyclic Graph

The transition from flat vector stores to graph-based memory is necessitated by the inherent structure of human and agentic experience. Human memory is not a random-access lookup table; it is a continuously evolving substrate that relies on time and causality as scaffolds for recall [4]. Replicating this capacity in AI agents necessitates a topological model of experience that preserves relational dependencies [7].

## 2.1. Formal Graph Definition

The *Trace* is the episodic memory subsystem of the Aeon Micro-Graph Kernel. It is structured as a Directed Acyclic Graph  $G = (V, E)$  [2], where the vertex set  $V$  is partitioned into three disjoint subsets:

$$V = V_{\text{user}} \cup V_{\text{system}} \cup V_{\text{concept}}, \quad V_{\text{user}} \cap V_{\text{system}} = \emptyset, \quad V_{\text{user}} \cap V_{\text{concept}} = \emptyset, \quad V_{\text{system}} \cap V_{\text{concept}} = \emptyset \quad (1)$$

The three vertex partitions are defined as follows.  $V_{\text{user}}$  contains human input events (utterances, commands, and feedback).  $V_{\text{system}}$  contains agent-generated responses (answers, actions, and tool invocations).  $V_{\text{concept}}$  contains abstract semantic cluster nodes that serve as stable reference anchors in the embedding space.

Each node  $v \in V$  carries a composite payload:

$$v = (\mathbf{e}_v, t_v, m_v) \quad (2)$$

where  $\mathbf{e}_v \in \mathbb{R}^d$  is a dense embedding vector encoding semantic content,  $t_v \in \mathbb{R}^+$  is a monotonically increasing timestamp, and  $m_v$  is a metadata record containing session identifiers, tool-call traces, and token counts. This hybrid design (neural embeddings at the nodes, symbolic structure at the edges) is the defining characteristic of the neuro-symbolic architecture.

## 2.2. Typed Edge Families

The edge set  $E$  is partitioned into three families, each formally defined using set-builder notation:

**Temporal edges.**

$$E_{\text{next}} = \{(v_i, v_{i+1}) \mid v_i, v_{i+1} \in V_{\text{user}} \cup V_{\text{system}}, t_{v_i} < t_{v_{i+1}}, \nexists v_k : t_{v_i} < t_{v_k} < t_{v_{i+1}}\} \quad (3)$$

These edges form a strict chronological chain. The constraint  $\nexists v_k$  ensures that the temporal backbone contains no “gaps”: every consecutive pair of events is directly connected, producing a total order over the episodic timeline.

**Causal edges.**

$$E_{\text{causal}} = \{(v_j, v_k) \mid v_j, v_k \in V, t_{v_j} < t_{v_k}, v_k \text{ was produced using information from } v_j \text{ as a premise}\} \quad (4)$$

Unlike temporal edges, causal edges may span arbitrary temporal distances. An edge  $(v_j, v_k) \in E_{\text{causal}}$  records that the action at  $v_k$  was *justified* by the information at  $v_j$ , even if dozens of turns intervene. These edges capture the “why” of agent behavior, enabling post-hoc auditability and root-cause analysis.

**Reference edges.**

$$E_{\text{ref}} = \{(v_i, c_j) \mid v_i \in V_{\text{user}} \cup V_{\text{system}}, c_j \in V_{\text{concept}}, \text{sim}(\mathbf{e}_{v_i}, \mathbf{e}_{c_j}) > \tau_{\text{ref}}\} \quad (5)$$

Reference edges anchor transient episodic events to stable concept nodes, preventing *semantic drift*, the gradual divergence of an episodic narrative from its grounding ontology. The threshold  $\tau_{\text{ref}}$  controls the specificity of grounding: a lower threshold creates denser cross-references, while a higher threshold ensures only strong semantic alignments are recorded.

The complete edge set is:

$$E = E_{\text{next}} \cup E_{\text{causal}} \cup E_{\text{ref}} \quad (6)$$

This tripartite edge structure allows the agent to traverse its history not merely by “what things look like” (vector similarity) but by “how they were reached” (structural traversal). This mirrors recent work on DAG-based reasoning, which demonstrates that *logical closeness* (adherence to structural rules during traversal) is a more reliable metric for reasoning fidelity than answer-level accuracy alone [8].

### 2.3. Backtracking: Complexity Analysis

A critical affordance of the DAG structure is the ability to *backtrack*. By traversing the inverse temporal edges  $E_{\text{next}}^{-1}$ , the agent can “rewind” its cognitive state to an earlier point in the conversation. This capability is essential for error recovery: if the agent discovers that a downstream decision was based on faulty premises, it can trace back through the causal chain to the originating node and re-evaluate from that point.

The complexity advantage of graph-based backtracking over flat retrieval is significant. In the Trace DAG, given a current node  $v_n$ , reaching the ancestor node  $v_{n-k}$  that is  $k$  temporal hops away requires following exactly  $k$  inverse edges. Assuming constant-time pointer dereferences within an active, in-memory Trace DAG:

$$\text{Cost}_{\text{DAG}}(\text{backtrack}, k) = O(k) \quad (7)$$

Each hop is a constant-time pointer dereference along a pre-computed edge. In contrast, in a flat vector store, recovering the “state  $k$  turns ago” requires issuing a similarity query against the entire corpus of  $N$  embeddings, yielding:

$$\text{Cost}_{\text{Flat}}(\text{backtrack}, k) = \begin{cases} O(N) & \text{exhaustive scan} \\ O(\log N) \text{ (expected)} & \text{with ANN indexing (e.g., HNSW)} \end{cases} \quad (8)$$

Crucially, even with ANN indexing (e.g., HNSW [9]), the flat store cannot guarantee that the retrieved node is the *causally correct* ancestor; it returns the *semantically nearest* node, which may be from an entirely different session. The DAG provides **O(k) exact causal backtracking** versus  $O(\log N)$  **approximate semantic retrieval**, a qualitative difference that is invisible to benchmark metrics but decisive in real-world error recovery.

### 2.4. Context Anchoring

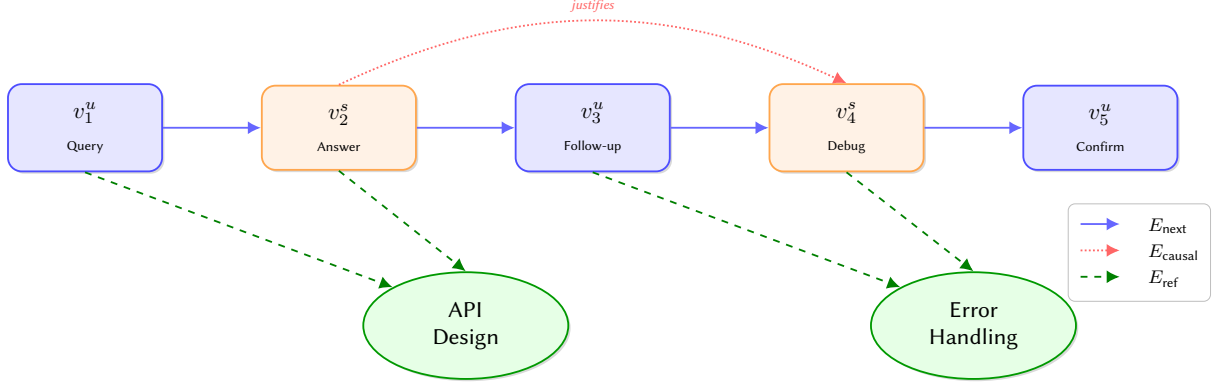
The reference edges  $E_{\text{ref}}$  provide *context anchoring*. In long, multi-session interactions, episodic nodes risk becoming semantically untethered as new information accumulates. By grounding episodic events to stable concept nodes, the Trace ensures that even distant memories remain navigable through a persistent conceptual lattice [10]. This echoes findings from episodic memory research showing that binding semantic structures into a persistent workspace enables tracking entities through evolving roles and spatiotemporal contexts [10].

Formally, for any episodic node  $v_i$  that occurred in session  $s$ , the set of its reference anchors  $\mathcal{A}(v_i) = \{c_j \mid (v_i, c_j) \in E_{\text{ref}}\}$  provides a session-invariant “fingerprint” that survives context shifts. A query seeking memories related to concept  $c_j$  can efficiently retrieve all episodes grounded to that concept by traversing  $E_{\text{ref}}^{-1}$ , regardless of temporal distance.

Figure 1 illustrates the structure of the Trace episodic DAG with its three edge families.

## 3. Overcoming Graph Latency: The Semantic Lookaside Buffer

While the Trace DAG provides structural fidelity, graph traversal introduces latency costs that can undermine real-time interaction. Global search over a growing graph of  $N$  nodes incurs  $O(\log N)$  complexity per query when using hierarchical index structures (e.g., HNSW [9]), or  $O(N)$  for exhaustive scans. For an agent managing tens of thousands of episodic nodes across hundreds of sessions, even logarithmic overhead compounds across the dozens of retrievals required per response generation. To address this, the Aeon Micro-Graph Kernel introduces the *Semantic Lookaside Buffer* (SLB), a predictive caching mechanism that exploits a fundamental property of human-agent dialogue: the locality of semantic topics [2].



**Figure 1:** The Trace Episodic DAG. User nodes (blue) and System nodes (orange) are connected by temporal edges  $E_{\text{next}}$ . Reference edges  $E_{\text{ref}}$  (dashed) anchor episodic events to stable concept nodes (green). A causal edge  $E_{\text{causal}}$  (dotted) links a justification chain across temporal distance. Each node carries an embedding vector  $\mathbf{e}_v$ , timestamp  $t_v$ , and metadata  $m_v$ .

### 3.1. The Semantic Inertia Hypothesis

We introduce the concept of *Semantic Inertia* to describe the spatial correlation inherent in continuous dialogue. In a multi-turn interaction, the topic vector  $\mathbf{q}_i$  at turn  $i$  is highly correlated with the topic vector  $\mathbf{q}_{i+1}$  at the subsequent turn:

$$P(\text{sim}(\mathbf{q}_{i+1}, \mathbf{q}_i) > \tau) \approx 1 \quad \text{for continuous dialogue} \quad (9)$$

where  $\text{sim}(\cdot, \cdot)$  denotes cosine similarity and  $\tau$  is a continuity threshold. The intuition is physical: just as a massive body resists changes to its velocity, a focused conversation resists abrupt shifts in its semantic trajectory. Topic transitions are overwhelmingly *incremental* (the next query is a small perturbation of the current one) rather than *discontinuous*.

This hypothesis is supported empirically. In realistic multi-turn conversational workloads, consecutive queries share greater than 85% cosine similarity more than 85% of the time [2]. Abrupt topic shifts (“hard pivots”) account for fewer than 15% of transitions, and even these pivots typically revisit a previously established concept node rather than introducing an entirely novel topic. This means the “active neighborhood” of the memory graph is highly localized at any given moment: if the system can predictively pre-fetch this neighborhood, it can bypass the  $O(\log N)$  costs of global search [2].

Research on temporal alignment in language models further supports this hypothesis. The “Capacity-Stability Trade-off” identified in recent work shows that models can better internalize temporal biases with minimal “alignment tax” [11], validating the use of a dedicated kernel to manage the stability of the local semantic context while the macro-router handles global knowledge capacity.

### 3.2. Architecture: A Predictive Graph Cache

The SLB draws its conceptual inspiration from the Translation Lookaside Buffer (TLB) in traditional operating systems, which accelerates virtual-to-physical address translation by caching recent mappings. Analogously, the SLB caches the “active neighborhood” of the semantic graph to achieve sub-millisecond retrieval [2].

The SLB is structured as a small, contiguous buffer of  $K$  recently accessed semantic centroids:

$$B = \{\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_K\}, \quad K \ll N \quad (10)$$

where  $K$  is tuned to be small enough (typically  $K \in [16, 64]$ ) for rapid exhaustive comparison. Unlike the global index, which requires hierarchical tree traversal, the SLB performs a *parallelized similarity comparison* across all  $K$  entries simultaneously. This design makes it a *software-managed* cache, distinct from traditional OS page caches, that is optimized specifically for semantic vector operations.

### 3.3. Complexity Comparison: $O(K)$ versus $O(\log N)$

The core complexity advantage is stark. For a query  $\mathbf{q}$ :

**SLB lookup (Cache Hit path):**

$$\text{Cost}_{\text{SLB}}(\mathbf{q}) = O(K) \cdot C_{\text{sim}} \quad (11)$$

where  $C_{\text{sim}}$  is the cost of a single cosine similarity computation ( $O(d)$  for  $d$ -dimensional vectors). Since  $K$  is a small constant (e.g., 32), this reduces to  $O(d)$ , effectively constant time with respect to corpus size  $N$ .

**Global index lookup (Cache Miss path):**

$$\text{Cost}_{\text{Global}}(\mathbf{q}) = O(\log N) \cdot C_{\text{sim}} + C_{\text{IO}} \quad (12)$$

where  $C_{\text{IO}}$  represents the I/O cost of loading index layers from persistent storage. For HNSW, the logarithmic factor hides a constant proportional to the number of layers  $M$ , and each layer hop involves random access patterns that are cache-unfriendly on modern memory hierarchies.

**Multi-hop graph reasoning (System 2 path):**

$$\text{Cost}_{\text{Graph}}(\mathbf{q}) = O(h \cdot b) \quad (13)$$

where  $h$  is the number of hops and  $b$  is the average branching factor. For complex thematic queries requiring 3–5 hops over a knowledge graph with branching factor 10–50, this yields  $10^3$ – $10^5$  node evaluations, orders of magnitude more expensive than either the SLB or global index path.

The SLB’s  $O(K)$  cost is *independent* of corpus size, making it asymptotically optimal for the common case of high semantic locality. When the Semantic Inertia hypothesis holds (Equation 9), the SLB intercepts the vast majority of queries before they ever reach the global index, amortizing the expensive  $O(\log N)$  path over the rare cache-miss events.

### 3.4. The Speculative Fetch Algorithm

The lookup procedure proceeds as follows: given a query  $\mathbf{q} \in \mathbb{R}^d$ , the system computes the similarity  $s_j = \text{sim}(\mathbf{q}, \mathbf{c}_j)$  for every cached centroid  $\mathbf{c}_j \in B$ , selects the best match  $s_{\text{best}} = \max_j s_j$ , and evaluates against the hit threshold  $\tau_{\text{hit}}$ :

- If  $s_{\text{best}} > \tau_{\text{hit}}$  (typically 0.85): a **Cache Hit** is declared, and the system returns the direct pointer to the corresponding Atlas graph node, bypassing the global index entirely [2].
- If  $s_{\text{best}} \leq \tau_{\text{hit}}$ : a **Cache Miss** triggers an authoritative fallback to the full Atlas index, and the result populates the SLB via a Least Recently Used (LRU) eviction policy [2].

Algorithm 1 formalizes this speculative fetch procedure. In production environments, the buffer  $B$  requires a thread-safe ring buffer or read-copy-update (RCU) mechanism to prevent read-write race conditions during the ASYNC\_PREFETCH routine, and  $\tau_{\text{hit}}$  is treated as an empirically tuned hyperparameter.

### 3.5. Asynchronous Prefetching and Hierarchical Loading

A key property of the SLB is its ability to perform *asynchronous prefetching* during natural pauses in user interaction (typing delays, reading time, or inter-turn gaps). When a Cache Hit occurs on node  $p^*$  at depth  $d$  in the Atlas hierarchy, the system speculatively loads the children of  $p^*$  from the persistent index into the SLB [2]. This leverages the hierarchical topology of the Atlas spatial index: if a user is querying a sub-topic (e.g., “vector database benchmarks”) and the SLB hits on the broader “Database” concept node  $c_{\text{db}}$ , the system proactively stages the “Vector ANN” and “Query Optimization” child nodes into the buffer:

$$B' = B \cup \{c \mid (c_{\text{db}}, c) \in E_{\text{Atlas}}\} \quad (14)$$

---

**Algorithm 1** Semantic Lookaside Buffer: Speculative Fetch

---

**Require:** Query vector  $\mathbf{q}$ , Buffer  $B = \{\mathbf{c}_1, \dots, \mathbf{c}_K\}$ , Threshold  $\tau_{\text{hit}}$ **Ensure:** Retrieved node pointer  $p^*$ 

```
1:  $s_{\text{best}} \leftarrow -\infty, j^* \leftarrow \text{null}$ 
2: for  $j = 1$  to  $K$  do ▷ Parallelized comparison
3:    $s_j \leftarrow \text{sim}(\mathbf{q}, \mathbf{c}_j)$ 
4:   if  $s_j > s_{\text{best}}$  then
5:      $s_{\text{best}} \leftarrow s_j, j^* \leftarrow j$ 
6:   end if
7: end for
8: if  $s_{\text{best}} > \tau_{\text{hit}}$  then ▷ Cache Hit
9:    $p^* \leftarrow \text{pointer}(\mathbf{c}_{j^*})$ 
10:   $\text{ASYNCPREFETCH}(\text{children of } p^*)$  ▷ Speculative loading
11:  return  $p^*$ 
12: else ▷ Cache Miss
13:    $p^* \leftarrow \text{AtlasFullSearch}(\mathbf{q})$ 
14:    $\text{LRU-INSERT}(B, \mathbf{q}, p^*)$ 
15:   return  $p^*$ 
16: end if
```

---

This effectively pipelines the semantic navigation, hiding the memory latency of the global graph behind the user’s cognitive processing time.

Empirical evaluation on realistic conversational workloads demonstrates that the SLB achieves hit rates exceeding 85%, reducing the effective retrieval latency by approximately  $6\times$  compared to full graph traversal [2]. This aligns with findings from the knowledge-graph-enhanced semantic cache literature, which reports significant latency improvements from caching semantically similar prompts [12], and with hierarchical memory architectures that show top-down routing reduces search complexity from exhaustive enumeration to targeted traversal [13].

## 4. Cognitive Routing and Enterprise Graph Integration

While the Micro-Graph Kernel manages the episodic state (Layer 1), the Macro-Graph Router (PandoraLM) handles the “System 2” reasoning over global structures (Layer 2). This dual-process routing ensures that the agent can navigate complex relational data and enforce enterprise-grade security without compromising interaction speed.

### 4.1. Dual-Process Strategy Selection

PandoraLM functions as a meta-cognitive layer that analyzes query characteristics to determine the optimal reasoning depth [6]. Inspired by Kahneman’s dual-process theory of cognition [5], the router classifies every incoming query into one of four categories and dispatches it along the appropriate execution path:

- **System 1 (Fast, Intuitive):** Queries classified as **FACTUAL** (direct lookups, definitions, simple retrievals) are routed to a high-speed vector store. This path optimizes for latency, returning results through approximate nearest neighbor search over dense embeddings. System 1 processing mirrors the automatic, low-effort cognitive responses described by Kahneman, where the cost of deliberation would exceed the value of the answer.
- **System 2 (Slow, Deliberate):** Queries classified as **THEMATIC** or **RESEARCH** (those requiring relational reasoning, cross-document synthesis, or multi-hop traversal) are routed to a knowledge graph store. This path employs Cypher queries over a Neo4j graph database, enabling the agent

to reason about entity relationships, temporal evolution, and thematic conflicts [14]. System 2 processing corresponds to the effortful, analytical cognition reserved for problems that demand structural understanding.

- **Code Path:** Queries classified as CODE (those involving source code understanding, debugging, or implementation) are dispatched to the Code Brain pipeline (Section 4.2), which operates over structural AST representations rather than text chunks.

Preliminary internal evaluation of the PandoraLM implementation yielded up to a 70% reduction in API inference costs compared to an always-on reasoning model baseline. The dual-process framing also aligns with Reasoning RAG architectures that distinguish between predefined (System 1) and agentic (System 2) reasoning modes [15], and with research on routing collapse prevention that addresses the risk of underutilizing specialized retrieval pipelines [16].

## 4.2. Structural Code Representation: The Code Brain

Standard text chunking is a primary source of Vector Haze in coding tasks. Traditional methods break code into fixed-size segments, often severing a function signature from its body or an import statement from its usage site [17]. PandoraLM addresses this through AST-aware structural chunking.

Using the Tree-sitter parser, source code is parsed into a hierarchical tree of typed syntax nodes (e.g., `FunctionDefinition`, `ClassDeclaration`). A recursive “split-then-merge” algorithm ensures that code is divided into segments that respect program logic and preserve structural boundaries [17]. Each chunk retains “breadcrumb context” (metadata headers specifying its parent class, inherited methods, and invoked dependencies) so that when a snippet is retrieved, the LLM understands its position within the larger codebase architecture.

These structural nodes are then mapped into the knowledge graph using typed relationships:

- **CONTAINS:** A module contains classes and functions.
- **CALLS:** Function *A* invokes function *B*.
- **INHERITS:** Class *C* extends class *D*.
- **IMPORTS:** Module *X* depends on module *Y*.
- **HAS\_METHOD:** Class *C* defines method *m*.

Empirical evaluation of AST-based retrieval shows significant gains, with a 5.5-point improvement on code retrieval benchmarks compared to line-based chunking [17].

### 4.2.1. Failure Scenario: Polymorphic Method Resolution

To illustrate why structural graph retrieval is necessary, consider a concrete failure scenario involving polymorphic inheritance.

**Setup.** A large enterprise codebase contains a base class `PaymentProcessor` with a method `authorize()`, and two subclasses: `CreditCardProcessor` (which overrides `authorize()` to add fraud-check logic) and `CryptoProcessor` (which overrides `authorize()` to call a blockchain validation service). A developer asks the agent: “*Why does `authorize()` sometimes call the fraud-check service and sometimes the blockchain service?*”

**Flat RAG Failure.** A traditional text-chunking pipeline indexes each method body as an independent text chunk. The query “`authorize()`” returns the three highest-similarity chunks: (1) `PaymentProcessor.authorize()` (the base method), (2) `CreditCardProcessor.authorize()`, and (3) an unrelated `AuthService.authorize_user()` from a different module that happens to share similar vocabulary. The flat retriever *cannot* represent the inheritance relationship, so it presents three disconnected code fragments. The LLM, seeing no structural link between them, may hallucinate an incorrect explanation or fail to identify that `CryptoProcessor` even exists, because its `authorize()` method is ranked below the irrelevant `AuthService`.

**AST Graph Resolution.** The Code Brain pipeline parses the codebase into typed AST nodes and maps them into the knowledge graph. Upon receiving the query, the system:

1. Retrieves the node for `PaymentProcessor.authorize()` via semantic search.
2. Traverses `INHERITS` edges to discover all subclasses: `CreditCardProcessor` and `CryptoProcessor`.
3. For each subclass, traverses `HAS_METHOD` edges to find their respective `authorize()` overrides.
4. Traverses `CALLS` edges from each override to discover the downstream services: `FraudCheckService` and `BlockchainValidator`.

The result is a complete, structurally verified inheritance tree with full call-chain context. The LLM receives *all* relevant methods in their hierarchical relationship, enabling a precise explanation of polymorphic dispatch. This multi-hop traversal (`INHERITS`  $\rightarrow$  `HAS_METHOD`  $\rightarrow$  `CALLS`) is computationally intractable to reconstruct in a flat vector store without maintaining costly, out-of-band relational indices.

### 4.3. ReBAC Security: Eliminating TOCTOU Vulnerabilities

In enterprise environments, the Macro-Graph also serves as the enforcement layer for *Relationship-Based Access Control* (ReBAC). Data is partitioned into hierarchical **Knowledge Layers** (`SYSTEM`, `ORGANIZATION`, `TEAM`, and `USER`), each representing a scope of visibility and control.

#### 4.3.1. The TOCTOU Problem in Standard RAG

Standard RAG pipelines suffer from a subtle but critical *Time-of-Check to Time-of-Use* (TOCTOU) vulnerability. The typical pattern is:

1. The system checks the user's permissions (Time-of-Check).
2. The retriever queries the vector store and returns candidate documents.
3. A post-hoc filter removes documents the user is not authorized to see (Time-of-Use).

Between steps 1 and 3, the unauthorized documents have already been *loaded into memory*, their embeddings have participated in similarity calculations, and, critically, they may have influenced the ranking of authorized documents. In adversarial settings, carefully crafted prompts can exploit this window to extract information about the *existence* of unauthorized documents (via ranking perturbation attacks) even if their content is filtered. The unauthorized data has already entered the reasoning context before the filter is applied.

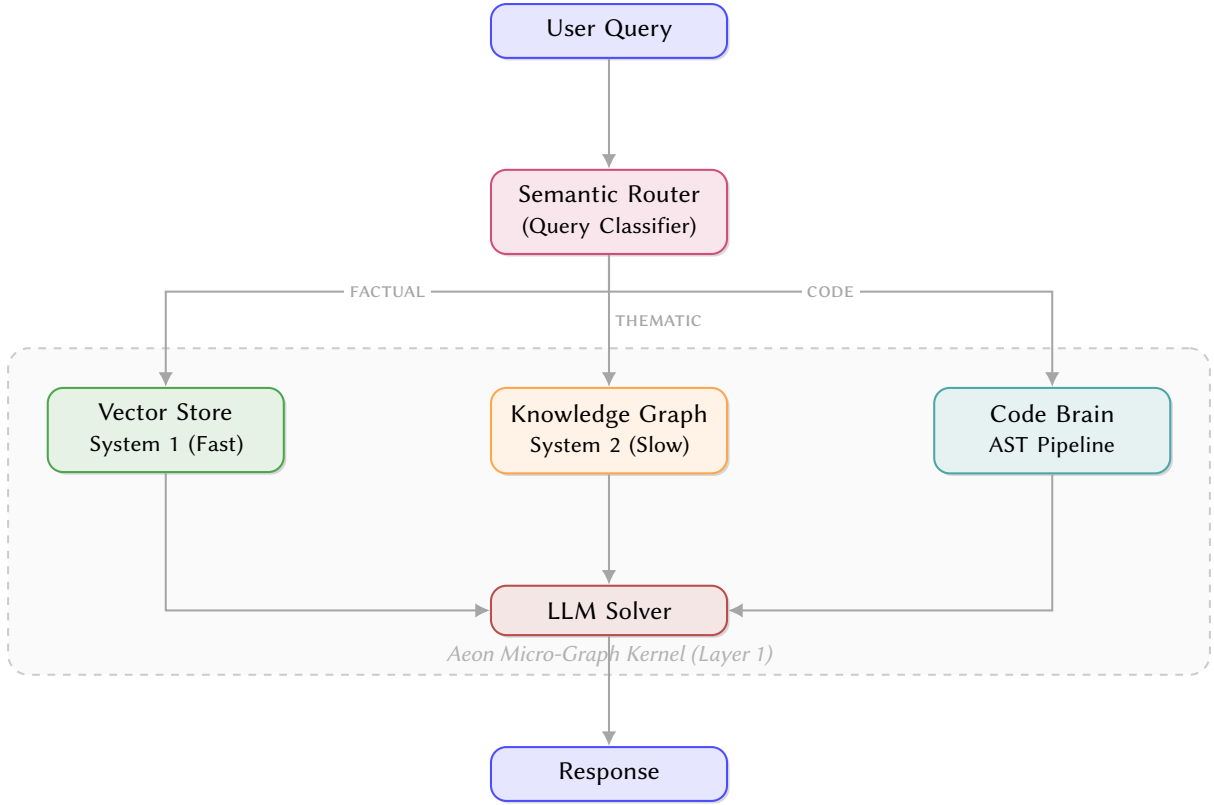
#### 4.3.2. Deterministic Pre-Filtering via Layer Injection

PandoraLM systematically mitigates this vulnerability through *deterministic pre-filtering*. Every incoming request carries an `X-Pandora-Layer-ID` header derived from the user's Identity Passport (e.g., Keycloak group memberships). This header is not merely a metadata annotation; it is injected directly into the Cypher execution plan as a `WHERE` clause constraint:

```
MATCH (n:KnowledgeNode)
WHERE n.layer_id IN $authorized_layers
AND n.embedding <-> $query_vec < $threshold
RETURN n
```

The `layer_id IN $authorized_layers` constraint is evaluated by the Neo4j query planner *before* any embedding similarity calculations occur. Unauthorized nodes are physically excluded from the traversal set: they are never loaded, never compared, and never ranked. This transforms the security model from *filter-after-retrieval* (TOCTOU-vulnerable) to *restrict-before-traversal* (TOCTOU-resistant). Assuming the database query planner strictly enforces pre-filtering prior to vector index traversal, this deterministic approach minimizes the window of time during which unauthorized data can influence reasoning pipelines, though timing side-channels remain an open challenge.

Figure 2 illustrates the cognitive routing architecture and its interaction with the underlying Aeon kernel.



**Figure 2:** PandoraLM Cognitive Routing Architecture. The Semantic Router classifies incoming queries and dispatches them to System 1 (Vector Store), System 2 (Knowledge Graph), or the Code Brain (AST). All paths converge at the LLM Solver using orthogonal routing. The dashed boundary indicates the Aeon Micro-Graph Kernel underlying all retrieval operations.

## 5. Discussion: Synergizing the Kernel and the Router

The fundamental architectural challenge of a dual-layer cognitive OS is the tension between the high-speed, verbose episodic DAG (Layer 1) and the consolidated, global knowledge graph (Layer 2). If every conversation turn were directly injected into the global knowledge graph, it would quickly become cluttered with transient noise, leading to retrieval degradation. Conversely, if the episodic DAG remains isolated, the agent cannot build cumulative, cross-session understanding. This paper proposes a theoretical *Memory Consolidation* process, a “Dreaming” phase, as the bridge between layers [2]. In this framework, *Memory Consolidation* refers to the biological/system *process* of migrating transient episodic data from the Trace DAG into long-term storage, whereas the *Memory Palace* denotes the *spatial Atlas index* where consolidated long-term episodic concepts permanently reside. While Layer 1 and Layer 2 have been implemented as independent reference systems, their real-time fusion via the Dreaming consolidation pipeline remains an active area of architectural research.

### 5.1. The Serialization Challenge

In the current architecture, Layer 1 and Layer 2 operate independently. The Aeon Micro-Graph Kernel manages within-session episodic memory with sub-millisecond responsiveness, while PandoraLM’s knowledge graph handles cross-session structural reasoning. Bridging these layers in real-time creates a serialization bottleneck: the raw, verbose trace data generated during high-speed interaction cannot be efficiently transmitted to the distributed graph router without incurring prohibitive overhead. A single 30-minute interaction session may generate hundreds of episodic nodes with full embedding payloads, metadata records, and edge sets, far too much data to inject into a production Neo4j instance synchronously without degrading query performance for concurrent users.

**Table 2**

Memory Consolidation Pipeline

Stage	Biological Parallel	Implementation
Clustering	Failure Analysis	LLM-based grouping
Abstraction	Gist Extraction	Candidate rule synthesis
Active Dream	Counterfactual Test	Scenario simulation
Integration	Synaptic Adjustment	KG commit + prune

The Dreaming process solves this by *decoupling the ingestion of experience from the consolidation of knowledge*. While the agent is “awake” and interacting, the kernel stores every raw detail in the episodic DAG. During idle time (between sessions or during scheduled maintenance windows) the Dreaming background process activates to transform fragile episodic traces into stable, long-term semantic knowledge within the Memory Palace [18].

## 5.2. The Sleep-Inspired Consolidation Pipeline

The consolidation process mirrors the biological interaction between the hippocampus and the neocortex, where memories are replayed and integrated into the existing knowledge network. Table 2 outlines the four-stage pipeline.

**Clustering.** The first stage groups related episodic traces; for example, clustering all debugging sessions that involved the same API endpoint or all conversations that referenced a particular architectural decision. The clustering uses the reference edges  $E_{\text{ref}}$ : nodes sharing common concept anchors  $\mathcal{A}(v_i) \cap \mathcal{A}(v_j) \neq \emptyset$  are candidates for the same cluster.

**Abstraction.** From each cluster, the system synthesizes “candidate rules,” generalized knowledge statements that abstract away the specific details of individual episodes. A cluster of five debugging conversations about null pointer exceptions in the authentication module, for instance, might yield the candidate rule: “The OAuth token refresh path does not handle concurrent requests safely.”

**Active Dreaming.** This stage goes beyond passive replay. Using the LLM, the system generates counterfactual simulations: “If I apply this new rule to a different but related situation, does it still hold?” [18]. Only rules that survive synthetic verification in a sandboxed environment proceed to the next stage.

**Integration.** Verified rules are committed to the PandoraLM knowledge graph as new entity nodes with structural edges linking them to relevant concepts, projects, and teams. These consolidated nodes reside permanently within the Memory Palace spatial index.

## 5.3. Graph Pruning: Deciding What to Forget

After integration, the system must decide which episodic nodes to prune. Aggressive pruning risks losing causal navigability; insufficient pruning allows unbounded graph growth. We define a pruning eligibility function  $\phi(v)$  that determines whether an episodic node  $v$  can safely transition to *dormant* status:

$$\phi(v) = \begin{cases} \text{DORMANT} & \text{if } \forall (v, c) \in E_{\text{ref}} : c \in V_{\text{committed}} \\ & \wedge |\{(v, w) \in E_{\text{causal}}\}| = 0 \\ & \wedge \text{age}(v) > T_{\text{decay}} \\ \text{ACTIVE} & \text{otherwise} \end{cases} \quad (15)$$

A node is eligible for dormancy if and only if: (1) all concept nodes it references have already been committed to the global KG ( $V_{\text{committed}}$ ), meaning no information is lost; (2) it has no outgoing causal edges, meaning no downstream decisions depend on it as a premise; and (3) it has exceeded the decay threshold  $T_{\text{decay}}$ , ensuring recent nodes remain accessible regardless of their structural properties.

### 5.3.1. Pruning Invariants

Two graph invariants must be preserved after pruning to guarantee continued correctness of traversal operations:

**Invariant 1: Temporal Chain Integrity.** If a node  $v_i$  is pruned, the temporal edges  $E_{\text{next}}$  must be repaired to maintain a connected backbone:

$$\text{If } v_i \text{ is pruned and } (v_{i-1}, v_i), (v_i, v_{i+1}) \in E_{\text{next}}, \text{ then add } (v_{i-1}, v_{i+1}) \text{ to } E_{\text{next}} \quad (16)$$

This “edge contraction” preserves the total ordering of the timeline, ensuring that backtracking via  $E_{\text{next}}^{-1}$  never encounters a broken chain.

**Invariant 2: Causal Ancestor Preservation.** No node  $v$  may be pruned if there exists any active node  $w$  such that  $(v, w) \in E_{\text{causal}}$ . This ensures that every active decision retains its full justification chain, enabling root-cause analysis at any time.

Dormant nodes are not deleted; they are moved to a compressed archival store with their embeddings and metadata intact. They can be “reawakened” if a future query strongly matches their embedding (i.e., a Cache Miss that resolves to an archival node), at which point they are restored to the active Trace and their edges are rehydrated.

## 5.4. Strategic Benefits

This synergy yields three core advantages:

- **Knowledge Retention:** Stress tests on self-modifying memory consolidation report 95% knowledge retention after 500 episodic cycles, effectively mitigating catastrophic forgetting [18].
- **Storage Efficiency:** Consolidation reduces storage growth by abstracting verbose transcripts into concise gists, allowing raw details to slip into dormancy unless specifically recalled [4].
- **Associative Routing:** By building structural edges between abstracted nodes (linking a technology to a project and its team) the system enables “spreading activation”: the ability to perform multi-hop recall even when exact query terms are absent [4].

## 6. Conclusion and Open Questions

This paper has presented a dual-layer Cognitive Operating System that addresses the Vector Haze problem by replacing flat vector retrieval with structured, navigable graph topologies. Layer 1 (the Aeon Micro-Graph Kernel) formalizes episodic memory as a neuro-symbolic DAG with formally defined temporal, causal, and referential edge families (Equations 3–6), providing  $O(k)$  exact causal backtracking versus  $O(\log N)$  approximate semantic retrieval. The Semantic Lookaside Buffer achieves  $O(K)$  retrieval complexity through predictive graph caching, exploiting the Semantic Inertia of continuous dialogue to intercept 85%+ of queries before they reach the global index.

Layer 2 (the PandoraLM Macro-Graph Router) implements Kahneman-inspired cognitive routing that eliminates computational waste through query classification, AST-aware code representation that resolves polymorphic inheritance chains impossible to navigate via flat chunking, and deterministic ReBAC security that eliminates TOCTOU vulnerabilities by physically excluding unauthorized nodes from the traversal set before any similarity computation. The proposed Dreaming consolidation process bridges the two layers through a biologically inspired four-stage pipeline with formally defined pruning invariants (temporal chain integrity and causal ancestor preservation) that guarantee correctness under graph contraction.

Three open questions warrant collaborative investigation:

1. **Symbolic Edge Maintenance Under Neural Concept Drift.** The Trace DAG’s symbolic edges ( $E_{\text{causal}}$ ,  $E_{\text{ref}}$ ) are defined relative to specific neural embedding models. When the underlying embedding model is upgraded or fine-tuned, the geometric relationships between node vectors

shift, potentially invalidating reference edges that were assigned under the previous space’s similarity metric (Equation 5). How should the system detect and repair stale symbolic edges without full  $O(|V| \times |V_{\text{concept}}|)$  re-computation of the reference edge set?

2. **Dynamic Pruning of Episodic DAGs.** The pruning eligibility function  $\phi(v)$  (Equation 15) provides a conservative baseline: nodes are only pruned when all their semantic content has been committed to the global KG. However, in high-throughput environments where an agent processes thousands of turns per day, even this conservative policy may leave an unmanageable residual graph. What relaxed invariants can be defined that allow more aggressive pruning (e.g., pruning causal ancestors if their *effects* have been independently verified) while bounding the probability of information loss?
3. **Vector Dimensionality and Multi-Modality.** Current spatial memory kernels often hard-code embedding dimensions (e.g.,  $d = 768$  for BERT-family models). As agent architectures increasingly integrate heterogeneous data sources, a critical question arises: how can graph-based memory OS kernels dynamically adapt to variable-length or multi-modal embeddings (e.g., integrating 1536-dimensional text embeddings with high-dimensional CLIP image embeddings) within the same episodic DAG, without incurring catastrophic spatial fragmentation of the Atlas index or prohibitive re-indexing costs across the Memory Palace?

The workshop community is invited to engage with these questions and contribute toward a scalable, principled framework for neuro-symbolic agent memory.

## Code Availability

Reference implementations of the architectures discussed in this paper are available as open-source repositories. The Macro-Graph Router (PandoraLM) can be accessed at <https://github.com/mustafarlan/pandoralm>, and the Micro-Graph Kernel (Aeon) at <https://github.com/mustafarlan/aeon>. Empirical metrics reported for the reference implementations (e.g., 70% cost reduction, 85% cache hit rates) were derived from internal benchmarks on simulated multi-turn technical workloads using local LLMs (e.g., 1B parameter router, 8B parameter solver). Full methodological details are available within the respective repositories.

## References

- [1] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, W.-t. Yih, Dense passage retrieval for open-domain question answering, arXiv preprint arXiv:2004.04906 (2020).
- [2] M. Arslan, Aeon: High-performance neuro-symbolic memory management for long-horizon LLM agents, arXiv preprint arXiv:2601.15311 (2026).
- [3] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, P. Liang, Lost in the middle: How language models use long contexts, arXiv preprint arXiv:2307.03172 (2023).
- [4] Y. Zhang, et al., Continuum memory architectures for long-horizon LLM agents, arXiv preprint arXiv:2601.09913 (2026).
- [5] D. Kahneman, Thinking, Fast and Slow, Farrar, Straus and Giroux, New York, 2011.
- [6] S. Kim, et al., Cognitive decision routing in large language models: When to think fast, when to think slow, arXiv preprint arXiv:2508.16636 (2025).
- [7] C. Liu, et al., Graph-based agent memory: Taxonomy, techniques, and applications, arXiv preprint arXiv:2602.05665 (2025).
- [8] Y. Zhang, I. Kuzborskij, J. D. Lee, C. Leng, F. Liu, DAG-Math: Graph-guided mathematical reasoning in LLMs, in: Advances in Neural Information Processing Systems (NeurIPS), 2025.
- [9] Y. A. Malkov, D. A. Yashunin, Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs, IEEE Transactions on Pattern Analysis and Machine Intelligence 42 (2018) 824–836.

- [10] S. Rajesh, P. Holur, C. Duan, D. Chong, V. Roychowdhury, Beyond fact retrieval: Episodic memory for RAG with generative semantic workspaces, arXiv preprint arXiv:2511.07587 (2025).
- [11] Y. Liao, et al., DZ-TDPO: Non-destructive temporal alignment for mutable state tracking in long-context dialogue, arXiv preprint arXiv:2512.03704 (2025).
- [12] X. Li, et al., Knowledge graph-enhanced semantic cache for low-latency and cost-effective inference in large language models, in: IEEE International Conference on Big Data, IEEE, 2024.
- [13] H. Sun, S. Zeng, Hierarchical memory for high-efficiency long-term reasoning in LLM agents, arXiv preprint arXiv:2507.22925 (2025).
- [14] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Ben-David, C. Larson, From local to global: A graph RAG approach to query-focused summarization, arXiv preprint arXiv:2404.16130 (2024).
- [15] J. Huang, et al., Reasoning RAG via system 1 or system 2, in: Findings of the Association for Computational Linguistics: ACL-IJCNLP, 2025.
- [16] Y. Zhang, et al., When routing collapses: On the degenerate convergence of LLM routers, arXiv preprint arXiv:2602.03478 (2026).
- [17] S. Wang, et al., cAST: Enhancing code retrieval-augmented generation with structural chunking via abstract syntax tree, arXiv preprint arXiv:2506.15655 (2025).
- [18] Y. Chen, et al., Language models need sleep: Learning to self modify and consolidate memories, OpenReview (2025). <https://openreview.net/pdf?id=iiZy6xyVVE>.