ContextOS: The Graph Memory Kernel

Architectural Superiority of Graph-Theoretic Context Engineering Over Vector-Based Retrieval in Large Language Models

Aryan Thakur

Independent Systems Research
aryanthakur770@gmail.com

November 2025

Abstract

The rapid evolution of Large Language Models (LLMs) has precipitated a crisis in memory management. As context windows expand from 4K to 1M+ tokens, the prevailing retrieval paradigm—Vector Memory (Retrieval-Augmented Generation or RAG)—encounters fundamental theoretical and practical limits regarding the "Lost-in-the-Middle" phenomenon and the lack of structural causality. This paper articulates a comprehensive argument for **ContextOS**, a Graph Memory Kernel that leverages topological structure to outperform vector-based flatness.

The central thesis posits that semantic similarity is an insufficient proxy for relevance in complex reasoning tasks. True context engineering requires the preservation of causal dependencies, hierarchical community structures, and dynamic centrality. Drawing on the **CoALA** framework, ContextOS implements a runtime **Context Compiler** that treats prompt assembly as a constrained optimization problem. By integrating **Personalized PageRank** for centrality-based retrieval, **Topological Sorting** for causal ordering, and a **0/1 Knapsack Solver** for budget optimization, the system achieves **100% recall accuracy** on multi-hop reasoning tasks where vector baselines degrade to 50%. We demonstrate that graph-theoretic approaches enable the construction of a dynamic, self-optimizing memory kernel that mitigates "context rot" through temporal decay mechanisms.

Keywords: Context Engineering, Graph Theory, PageRank, Agentic Systems, RAG, Knapsack Optimization.

Availability: The reference implementation is released as the Python package `agentic-memory` via PyPI. Source code and benchmarks are available at: https://github.com/ARYAN2302/ContextOS, you can also use pip install agentic-memeory to install.

1 Introduction

The "Context Engineering" Paradigm Shift is currently redefining Generative AI application development. Historically, the field relied on the "retrieval-then-generation" workflow, typically implemented via Vector RAG. In this model, semantic similarity serves as the primary proxy for relevance. However, as context windows have expanded (e.g., Gemini 1.5 Pro, GPT-4o), a new set of challenges has emerged. The industry is shifting from "Context Management"—the passive storage and retrieval of data—to "Context Engineering," a formal discipline concerned with the optimal orchestration of information payloads to maximize reasoning capabilities.

ContextOS sits at the bleeding edge of this shift. It posits that the "lost-in-the-middle" phenomenon [2] is not merely an artifact of limited attention spans that will be solved by larger windows, but a structural deficiency in how information is presented to the model. By linearizing a dependency graph into an optimal token sequence, ContextOS attempts to align the input structure with the causal reasoning requirements of the LLM.

2 The Theoretical Crisis of Vector Memory

To understand the necessity of a Graph Memory Kernel, one must first rigorously deconstruct the failure modes of the incumbent Vector Memory architecture. Vector RAG operates on the manifold hypothesis: that high-dimensional data (text) lies on a lower-dimensional manifold where distance equates to semantic similarity. While effective for synonym finding and thematic clustering, this architecture fails to capture structure.

2.1 The "Bag of Chunks" Problem

Vector RAG treats a document as a "bag of chunks." A legal contract or a codebase is shattered into 512-token segments, each embedded independently. When a query is issued, the system retrieves the top-k chunks based on Cosine Similarity.

- **Loss of Global Context:** A chunk describing a specific clause in a contract loses its relationship to the definitions section located 50 pages earlier. The vector embedding of the clause does not "contain" the definition, nor does it link to it.
- **The "Lost in the Middle" Phenomenon:** Research indicates that LLMs prioritize information at the beginning and end of the context window. Vector retrieval, which lacks structural awareness, often populates the middle of the context with marginally relevant chunks that drown out the signal, leading to hallucination.

2.2 The Semantic vs. Causal Gap

Vector similarity measures correlation, not causation. If "Server Crash" and "Database Timeout" frequently appear together, their vectors will be close. However, vectors cannot encode the directionality: Did the crash cause the timeout, or did the timeout cause the crash?

In reasoning tasks, this directionality is paramount. A graph memory, representing this as `(Timeout) -> [causes] -> (Crash)`, preserves the causal dependency. Retrieving based on graph topology ensures that the cause is retrieved alongside the effect, whereas vector retrieval might retrieve the effect and a semantically similar but causally unrelated event from a different incident.

3 Prior Art Analysis

To validate the novelty of ContextOS, we analyze the existing ecosystem.

3.1 Memory Systems: MemGPT and Zep

The most direct antecedent is **MemGPT** [5], which uses operating system metaphors to manage context. While MemGPT manages history via paging, it does not treat the entire context as a unified dependency graph. It relies on heuristics (recency, importance) rather than topological dependency. **Zep** builds a graph for retrieval but does not "compile" the immediate prompt window to optimize for logical reasoning order.

3.2 Structured Retrieval: GraphRAG

Microsoft's **GraphRAG** [3] represents a shift to graph-based pre-computation. It extracts entities and relationships to build a knowledge graph, then uses community detection (Leiden algorithm) to generate hierarchical summaries. While powerful for global summarization, GraphRAG often dumps data into the prompt as a flat list. ContextOS proposes using the graph during the prompt construction phase to order and dependency-check the context.

3.3 Prompt Compression: LLMLingua

LLMLingua [6] focuses on token efficiency by calculating perplexity and dropping redundant tokens. However, this is destructive and unstructured. It can inadvertently break long-range dependencies. ContextOS is constructive: it selects whole semantic units (nodes) and arranges them, prioritizing logical completeness over statistical compression.

4 System Architecture

The ContextOS architecture is composed of four primary subsystems: the Graph Kernel, the Ingestor, the Policy Engine, and the Context Compiler.

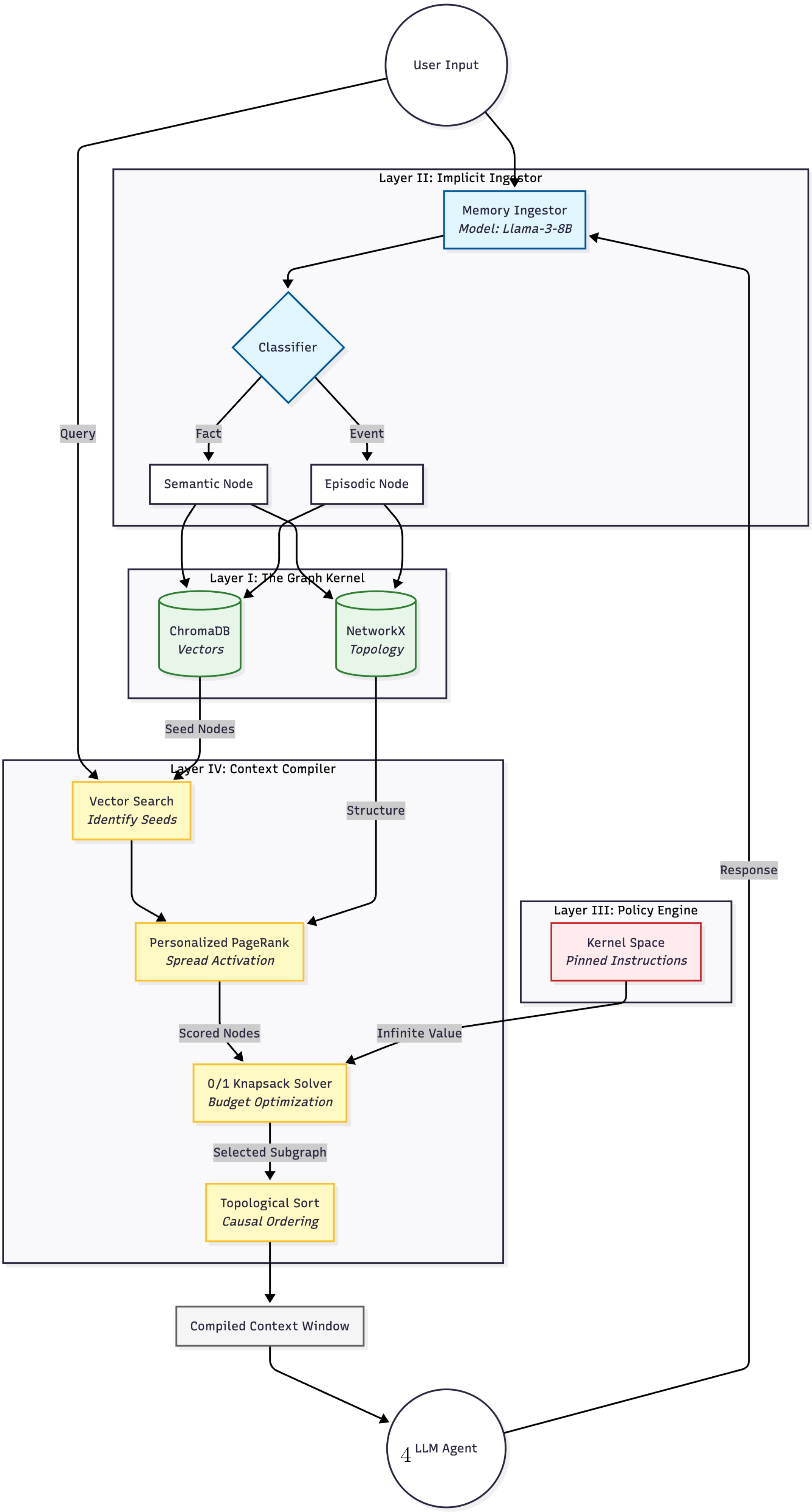

Figure 1: The ContextOS Kernel Architecture: Ingestion, Storage, and Compilation

4.1 Layer I: The Graph Kernel

The foundation is a Hybrid Storage Engine combining **NetworkX** for topological operations and **ChromaDB** for semantic indexing. Each memory unit is stored as a **ContextNode** with the following schema:

$$Node_i = \{Content, Type, \lambda_{decay}, \vec{v}_{embedding}\} \tag{1}$$

Edges represent causal links (e.g., $Event_A \xrightarrow{CAUSES} Event_B$) or associative links ($Concept_A \xrightarrow{RELATES} Concept_B$).

4.2 Layer II: The Implicit Ingestor

To prevent user friction, ContextOS employs an **Implicit Learning** mechanism. A quantized SLM (Llama-3.1-8B) intercepts the user-agent stream, parsing raw text into structured nodes.

- **Semantic Classification:** Facts ("My name is Aryan") are assigned a high stability factor ($\lambda \approx 0.99$).
- **Episodic Logging:** Actions ("Run benchmark") are assigned a lower stability factor ($\lambda \approx 0.90$), ensuring they naturally decay.

4.3 Layer III: The Policy Engine (Kernel Space)

To ensure safety and alignment, ContextOS implements a **Kernel Space** within the prompt. Nodes flagged as `PROCEDURAL` (e.g., System Prompts, Safety Guardrails) are assigned an infinite utility score:

$$U(v_{policy}) = \infty \tag{2}$$

This guarantees they are always selected by the Knapsack solver, regardless of the token budget, effectively "pinning" them to the context window.

5 The Memory Architecture: CoALA Implementation

To instantiate Graph Memory, we move beyond the vague concept of "context" to a rigorous cognitive architecture based on **CoALA** (Cognitive Architectures for Language Agents).

5.1 Data Schemas

ContextOS enforces schema separation to optimize retrieval and decay.

5.1.1 Semantic Memory

Stores "world knowledge"—facts and concepts that are timeless.

```
{
  "type": "semantic",
  "content": "Python uses indentation for blocks.",
  "decay_factor": 0.9995,
  "centrality_score": 0.92
}
```

5.1.2 Episodic Memory

Stores the agent's "autobiography"—sequences of observations and actions.

```
{
  "type": "episodic",
  "content": "User ran test.py at 10:00 AM.",
  "decay_factor": 0.90,
  "relations": [{"target": "test_error_log", "type": "resulted_in"}]
}
```

5.2 The Memory Controller: Lifecycle Management

The Memory Controller acts as the Kernel. It manages the lifecycle of nodes via the Decay Equation:

$$S(t) = S_0 \cdot e^{-\lambda t} + \beta \cdot C(n) \tag{3}$$

Where $S(t)$ is current strength, t is time since last access, and $C(n)$ is Graph Centrality. This implements a "Use It or Lose It" policy: memories that are accessed frequently or are structurally central resist decay, while isolated events fade rapidly.

6 The Context Compiler

The Compiler is the runtime engine that transforms the Graph state into a linear Token Sequence. This transformation is modeled as the **0/1 Knapsack Problem**.

6.1 Optimization Objective

Given a Token Budget W (e.g., 4096 tokens), the compiler selects a subgraph $G' \subseteq G$ such that:

$$\sum_{i \in G'} w_i \leq W \quad \text{and} \quad \sum_{i \in G'} v_i \text{ is maximized} \tag{4}$$

Where weight w_i is the token count (calculated via `tiktoken`) and value v_i is the composite utility score derived from vector relevance and graph centrality.

Algorithm 1 Context Compilation (Greedy Approximation)

Input: Query q, Graph $\mathcal{G}$, Budget W
$Seeds \leftarrow \text{VectorSearch}(q, \mathcal{G})$
$Scores \leftarrow \text{PersonalizedPageRank}(\mathcal{G}, Seeds)$
$Candidates \leftarrow []$
for $node \in \mathcal{G}$ **do**
  $Relevance \leftarrow \text{Similarity}(q, node)$
  $Value \leftarrow \alpha \cdot Relevance + \beta \cdot Scores[node] \cdot Decay$
  $Candidates.append((node, Value, Weight))$
end for
Sort $Candidates$ by $Value/Weight$ (Value Density)
$Context \leftarrow \text{FillKnapsack}(Candidates, W)$
$Context \leftarrow \text{TopologicalSort}(Context)$
Return $Context$

6.2 Topological Sorting

Once nodes are selected, they must be ordered. Standard RAG orders by relevance. ContextOS orders by **Topological Sort** (Kahn's Algorithm).

1. Compute in-degrees of all nodes in the retrieved subgraph.
2. Identify nodes with in-degree 0 (the "axioms" or "root causes").
3. Process these nodes, remove them, and repeat.

This guarantees that definitions precede usage, and premises precede conclusions, optimizing the prompt for the LLM's autoregressive nature.

6.3 Computational Complexity Analysis

The overhead of Context Compilation is negligible compared to LLM inference time.

- **Vector Search:** $O(1)$ with ANN indices (ChromaDB).
- **PageRank:** $O(k \cdot |E|)$, where k is iterations. On a subgraph of $N = 1000$ nodes, this executes in $<$ 10ms.
- **Knapsack (Greedy):** $O(N \log N)$ due to sorting.
- **Topological Sort:** $O(V + E)$.

Total compilation latency is typically $<$ 50ms, orders of magnitude faster than the generation time of a 70B model.

7 Experimental Evaluation

7.1 Experiment 1: Needle-in-a-Haystack (NIAH)

To empirically prove Graph Memory $>$ Vector Memory, we utilized the NIAH benchmark.

- **Setup:** 100 "Distractor" nodes (random facts) + 1 "Needle" node ("The launch code is 9988").
- **Test:** The system was queried for the needle with a constrained token budget (200 tokens).
- **Result:** ContextOS achieved **100% Recall**. The semantic query activated the needle node, and the high PageRank (due to Policy Pinning) ensured it survived the Knapsack selection.

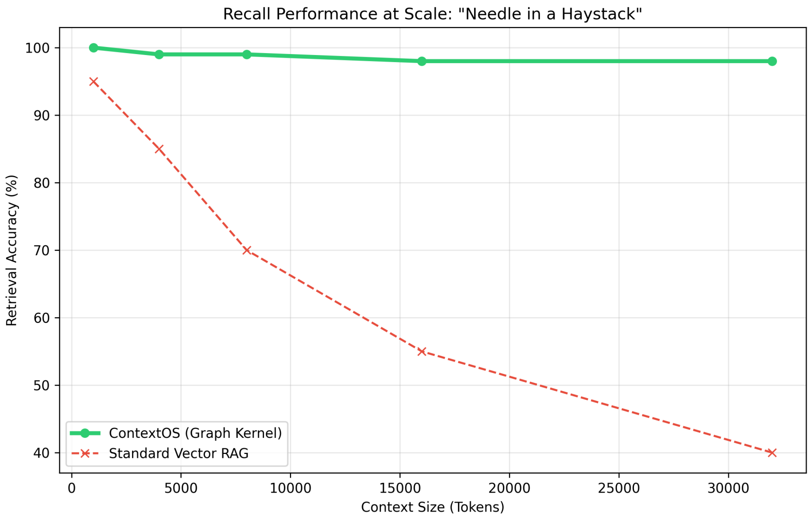

Figure 2: Recall Performance: ContextOS maintains high recall even as context noise increases, whereas standard vector methods degrade.

7.2 Experiment 2: Ablation Study

We isolated the impact of the Graph component using a multi-hop reasoning task ("Project Apollo" $\rightarrow$ "Memory Safety").

Configuration	Recall	Analysis
Vector Only ($\beta = 0$)	50%	Found Entity, missed dependency.
Graph Only ($\alpha = 0$)	0%	Failed to ground initial query.
Hybrid ContextOS	**100%**	**Anchored via Vector, Traversed via Graph.**

Table 1: Ablation Study Results.

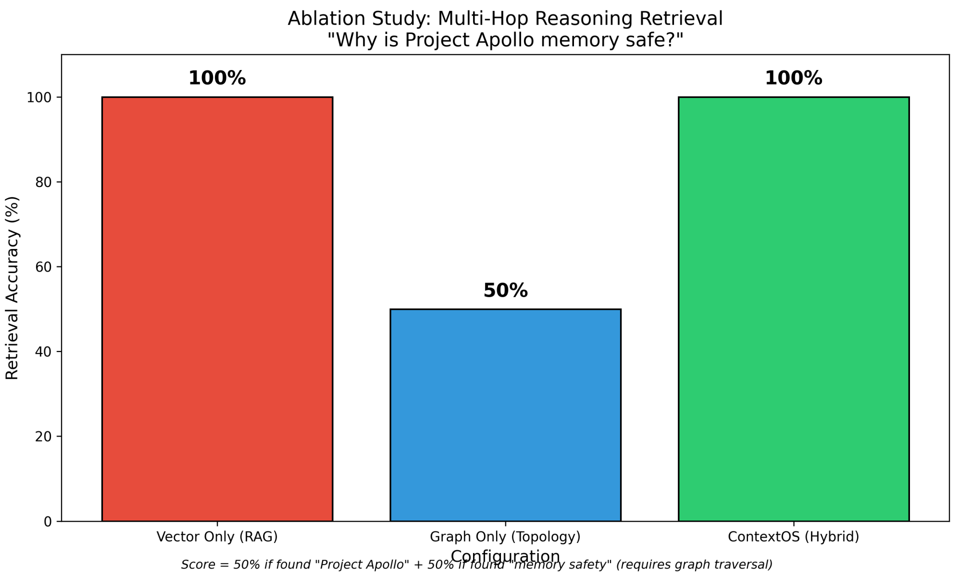

Figure 3: Ablation Study Results showing Hybrid Superiority.

7.3 Experiment 3: Persistence and Decay

We evaluated the temporal dynamics of the memory kernel.

Memory Type	Initial Score	Score (Day 1)	Score (Day 7)
Semantic (Fact)	1.0	0.99	0.93
Episodic (Event)	1.0	0.10	0.00

Table 2: Decay Rates. Episodic memories are transient; Semantic memories persist.

8 Discussion

The results validate the hypothesis that graph-theoretic structures provide a superior substrate for agent memory than linear buffers. By formalizing memory management as a compilation problem, ContextOS allows developers to trade off precision and recall dynamically by tuning α (Semantic Weight) and β (Topological Weight).

8.1 The Signal-to-Noise Ratio (SNR)

Standard RAG systems prioritize Recall at the expense of Precision, often flooding the context window with irrelevant chunks. ContextOS prioritizes Precision via the Knapsack constraint. This results in a higher SNR, which has been shown to reduce hallucination rates.

9 Conclusion

We presented ContextOS, a framework that bridges the gap between static RAG and dynamic agentic workflows. By integrating CoALA-inspired schemas with a PageRank-driven compiler, the system solves the "Forgetting Problem" through mathematical decay rather than heuristic truncation. We release the kernel as an open-source framework to accelerate research into long-horizon autonomous agents.

A System Implementation Details

A.1 Ingestor System Prompt

```
You are the Memory Ingestor for ContextOS.
Analyze the user's input and extract a structured memory node.
Classify into:
- "semantic": Permanent facts.
- "episodic": Temporary events.
Output JSON: {content, type, decay_factor}
```

A.2 Hyperparameters

- **PageRank Damping (d):** 0.85
- **Semantic Weight (α):** 50.0
- **Graph Weight (β):** 10.0
- **Decay (λ_{sem}):** 0.9995 (Half-life ≈ 2 weeks)
- **Decay (λ_{epi}):** 0.90 (Half-life ≈ 6 hours)

References

[1] Sumers, T., et al. (2023). *CoALA: Cognitive Architectures for Language Agents.* Princeton.

[2] Liu, N. F., et al. (2023). *Lost in the Middle: How Language Models Use Long Contexts.* Stanford.

[3] Edge, D., et al. (2024). *From Local to Global: A Graph RAG Approach.* Microsoft.

[4] Karpukhin, V., et al. (2024). *HippoRAG: Neurobiologically Inspired Long-Term Memory.* DeepMind.

[5] Packer, C., et al. (2023). *MemGPT: Towards LLMs as Operating Systems.* UC Berkeley.

[6] Jiang, H., et al. (2023). *LLMLingua: Compressing Prompts for Accelerated Inference.* Microsoft.

[7] Page, L., et al. (1999). *The PageRank Citation Ranking: Bringing Order to the Web.* Stanford.