

Aeon: High-Performance Neuro-Symbolic Memory Management for Long-Horizon LLM Agents

Mustafa Arslan

Independent Researcher, Istanbul, Turkey

Abstract

Large Language Models (LLMs) are fundamentally constrained by the quadratic computational cost of self-attention and the “Lost in the Middle” phenomenon, where reasoning capabilities degrade as context windows expand. Existing solutions, primarily “Flat RAG” architectures relying on vector databases, treat memory as an unstructured bag of embeddings. This approach fails to capture the hierarchical and temporal structure of long-horizon interactions, leading to “Vector Haze”: the retrieval of disjointed facts lacking episodic continuity. This paper proposes **Aeon**, a Neuro-Symbolic Cognitive Operating System that redefines memory not as a static store, but as a managed OS resource. Aeon structures memory into a **Memory Palace** (a spatial index implemented via ATLAS, a SIMD-accelerated **Page-Clustered Vector Index** that combines small-world graph navigation with B+ Tree-style disk locality to minimize read amplification) and a **Trace** (a neuro-symbolic episodic graph). The **Semantic Lookaside Buffer (SLB)**, a predictive caching mechanism, exploits conversational locality to achieve sub-millisecond retrieval latencies. Benchmarks on Apple M4 Max demonstrate that Aeon achieves $< 5\mu\text{s}$ effective retrieval latency on conversational workloads (with 85%+ SLB hit rates), while ensuring state consistency via a sub-microsecond zero-copy C++/Python bridge ($\sim 334\text{ns}$ for 10MB payloads), effectively enabling persistent, structured memory for autonomous agents.

1 Introduction

The rapid evolution of Large Language Models (LLMs) has been defined by a relentless scaling of parameters and training data, yet the fundamental architecture remains bound by the *Context Bottleneck*. The Transformer’s self-attention mechanism, while transformative, imposes a quadratic time and space complexity, $O(N^2)$, relative to the in-

put sequence length. Although recent optimization techniques (sparse attention, RingAttention, and hardware-aware kernel fusion) have theoretically extended context windows to 1 million tokens and beyond, the utility of this context does not scale linearly. Empirical evidence highlights a distinct degradation in reasoning capabilities over these extended horizons, a phenomenon widely characterized as being “Lost in the Middle” [8]. When relevant information is buried in the center of a massive context window, model performance deteriorates, often below the baseline of closed-book inference. As autonomous agents are tasked with increasingly complex, long-horizon objectives spanning days or weeks, the reliance on these transient, volatile context windows becomes untenable. The model cannot simply attend to all of history; it must select what is potentially relevant before attention is even applied. However, the selection mechanism itself is often limited by the same architectural constraints it seeks to mitigate. The simplistic view of context as a sliding window is insufficient for general-purpose autonomy. Instead, what is required is a memory architecture that is as persistent, structured, and deterministically managed as the storage hierarchies found in classical operating systems.

The prevailing industry response to the context limitation has been the widespread adoption of Retrieval-Augmented Generation (RAG). In its most common form, “Flat RAG,” this approach offloads information preservation to vector databases that perform Approximate Nearest Neighbor (ANN) search over unstructured lists of embeddings. Technologies such as HNSW (Hierarchical Navigable Small World) graphs or inverted file indices allow for efficient retrieval of semantically similar chunks. However, while effective for simple, one-shot question-answering tasks, Flat RAG fails to model the *structure* of extended interaction. It treats memory as a featureless plane (a “bag of vectors”) where the temporal evolution of a conversation, the causal lineage of decisions, and the hierarchical relationship between concepts are lost. This failure mode is termed “Vec-

tor Haze”: the retrieval of semantically similar but episodically disjointed facts that confuse rather than aid the agent. A Flat RAG system has no notion of “where” it is in a conversation; it only knows “what” matches the current query vector in high-dimensional space. It lacks the ability to backtrack to a previous state, to branch a conversation into potential futures, or to understand the narrative arc that led to the current moment. For an agent to maintain coherence over thousands of turns, it requires more than a semantic search engine; it requires a state machine that understands the topology of its own experience.

This paper proposes a paradigm shift from treating memory as a passive database retrieval problem to treating it as an active resource management problem within a **Cognitive Operating System**. In this view, memory management becomes analogous to virtual memory management in traditional OS kernels. **Aeon** formalizes these operations. *Allocation* corresponds to the deliberate writing of new semantic concepts into a structured *Atlas*; *paging* transforms into the loading of relevant semantic clusters into a **Semantic Lookaside Buffer (SLB)** for immediate, low-latency access; and *context switching* is re-framed as the deterministic movement between branches of a decision tree. By enforcing these rigorous abstractions, Aeon transforms the probabilistic chaos of vector search into a deterministic navigational process. This empowers the agent to utilize a “Memory Palace,” a spatial index where information is stored not just by what it means, but by where it belongs relative to other concepts in the agent’s ontology. This spatial locality enables the system to predictively pre-fetch memory, much like a CPU pre-fetches instructions based on spatial and temporal locality, drastically reducing the latency penalty of retrieval and allowing the agent to “think” at the speed of conversation.

The contributions of this paper are threefold. First, it introduces **Atlas**, a high-performance, memory-mapped B+ Tree that organizes uniform vectors into a navigable, hierarchical index. Unlike HNSW or IVFPQ indices which optimize purely for recall at the expense of insert performance and structure, Atlas optimizes for semantic locality and stable modification. It utilizes a custom SIMD-accelerated math kernel to perform metric comparisons, ensuring that the tree structure strictly adheres to the geometry of the embedding space. Second, the paper presents the **Trace**, a neuro-symbolic directed acyclic graph (DAG) that explicitly tracks the agent’s episodic state. The Trace records the traversal path of the agent, creating distinct nodes for User inputs, System responses, and intermediate thoughts, linked by

typed edges (CAUSAL, NEXT, REFERS_TO). This enables capabilities such as backtracking and context anchoring that are impossible in flat vector stores. Finally, the paper demonstrates the implementation of a zero-copy architecture bridging C++23 and Python. By harnessing the **nanobind** library, Aeon exposes internal C++ memory structures as read-only NumPy arrays, eliminating serialization overhead. This allows Aeon to achieve effective retrieval latencies of $< 5\mu s$ on standard conversational workloads, validating the feasibility of this Cognitive OS approach for real-time, interactive agents that require both the speed of systems programming and the flexibility of high-level reasoning.

2 Related Work

Aeon is positioned within the broader landscape of neural memory systems, contrasting its Cognitive Operating System architecture against existing approaches in retrieval, memory management, and neuro-symbolic reasoning. While individual components of Aeon have precedents in isolation, the lack of a unified, high-performance kernel has limited the emergence of truly long-horizon agents.

2.1 Retrieval-Augmented Generation (RAG)

The dominant paradigm for grounding Large Language Models (LLMs) is Retrieval-Augmented Generation (RAG), typically implemented using Dense Passage Retrieval (DPR) [7] and Approximate Nearest Neighbor (ANN) search indices like FAISS [5] or HNSW [9]. These systems rely on what is characterized here as “Flat RAG”: a single, monolithic vector space where every query is treated as an independent event, disconnected from temporal or causal context.

The primary limitation of Flat RAG is the phenomenon of *Vector Haze*. As the size of the memory store grows, the probability of retrieving semantically similar but contextually irrelevant facts increases, diluting the LLM’s attention mechanism. HNSW graphs, while efficient, are agnostic to the agent’s current task state. Aeon addresses this via *The Atlas*, a hierarchical spatial indexing system. By constraining the search space based on the agent’s active context region, Aeon mitigates Vector Haze, ensuring that retrieval precision does not degrade at scale.

2.2 Memory-Augmented LLMs

Attempts to give LLMs persistent memory have largely operated at the application layer. Systems like

MemGPT [11] introduce an OS-like abstraction for managing context windows, distinguishing between main context and external storage. Similarly, earlier works on Neural Turing Machines (NTM) [4] proposed differentiable memory banks.

However, MemGPT is a logical framework rather than a systems-level implementation. It functions in “User Space” (typically Python), relying on the LLM itself to manage memory calls via prompt engineering. This introduces significant latency and leaves the physical layout of memory unmanaged. Aeon moves this responsibility to “The Kernel,” implementing memory management in high-performance C++23. By treating memory operations as low-level system calls, Aeon achieves sub-microsecond retrieval latencies that are impossible with prompt-based management.

2.3 Neuro-Symbolic Knowledge Graphs

To address the lack of structure in vector stores, Neuro-Symbolic approaches like GraphRAG [3] and integration with graph databases (e.g., Neo4j) have gained traction. These systems excel at multi-hop reasoning by making relationships explicit.

The limitation of current Neuro-Symbolic systems is their rigidity and write latency. Symbolic graphs are often slow to update and require brittle extraction pipelines, lacking the fluid adaptability of neural representations. Aeon’s *Trace* module introduces a hybrid architecture: it utilizes neural embeddings for nodes to maintain semantic fluidity, while employing symbolic edges to enforce causal constraints. This allows Aeon to update reasoning paths in real-time without the heavy overhead of re-indexing traditional knowledge graphs.

2.4 Operating System Primitives for AI

Finally, the concept of an “AI Kernel” has been proposed in various forms, often serving as a metaphor for cloud orchestration layers or “Glue Code” libraries like LangChain [1]. While these frameworks provide essential developer tooling, they do not function as operating systems in the traditional sense; they do not manage physical memory layout, thread scheduling, or cache coherency.

Aeon distinguishes itself by implementing true OS primitives tailored for semantic data. The *Semantic Lookaside Buffer (SLB)* is a direct parallel to the Translation Lookaside Buffer (TLB) in CPU architecture. Instead of just caching raw vectors, the

SLB predicts and prefetches semantic clusters relevant to the current thread of thought. This moves the paradigm from reactive retrieval to proactive memory management, solving the “Context Bottleneck” at the architectural level.

3 System Architecture

Aeon implements a hybrid *Cognitive Kernel* architecture designed to bridge the gap between high-performance systems programming and high-level AI reasoning. The system is architected to satisfy strict latency constraints (< 200ms end-to-end memory retrieval) while maintaining the flexibility required for complex agentic workflows.

3.1 Design Philosophy: The Core-Shell Model

The central design philosophy of Aeon is the *Core-Shell* separation of concerns, ensuring that computational intensity and logical complexity are handled by the most appropriate runtime environments.

- **The Core (Ring 0):** Implemented in C++23, the Core is responsible for all high-frequency, low-latency operations. This includes vector similarity search, tree traversal, and memory management. It operates directly on raw memory pages and leverages hardware acceleration (SIMD AVX-512 via SIMDex) to maximize throughput.
- **The Shell (Ring 3):** Implemented in Python 3.12, the Shell manages high-level control logic, including Large Language Model (LLM) interaction, prompt engineering, and graph topology management. It serves as the “cognitive” layer that orchestrates the system’s behavior.

The critical invariant of this architecture is the **Zero-Copy Constraint**: Data is never serialized or copied between the Core and the Shell during normal operation. Instead, ownership of data resides in the Core, and the Shell operates on read-only views of shared memory pages. This eliminates the marshaling overhead typical of foreign function interfaces (FFI).

3.2 The Atlas: Spatial Memory Kernel

The *Atlas* is the foundational data structure of Aeon’s long-term memory, functioning as a spatial index for

semantic vectors. It is implemented as a specialized Hierarchical Navigable Small World (HNSW) variant, optimized for on-disk storage.

3.2.1 Data Structure

We define a memory node formally as a tuple N :

$$N = \{id, \mathbf{v}, \mathcal{C}, \text{meta}\} \quad (1)$$

where:

- $id \in \mathbb{N}^{64}$ is a unique 64-bit identifier.
- $\mathbf{v} \in \mathbb{R}^{768}$ is the semantic embedding vector (occupying 768×4 bytes).
- \mathcal{C} is the set of child pointers (offsets in the memory file).
- meta is a fixed-size metadata block for timestamping and source tracking.

3.2.2 Storage and Access

The Atlas resides entirely on persistent storage (NVMe SSD) but is mapped into the process’s virtual address space using the POSIX `mmap` system call. This allows the OS virtual memory subsystem to handle page caching transparently. Standard C++ heap allocations (e.g., `new`, `malloc`) are explicitly avoided for node data to ensure data contiguity and cache locality.

3.2.3 Greedy SIMD Descent

Retrieval is performed using a *Greedy SIMD Descent* algorithm. Given a query vector \mathbf{q} , and starting at a candidate node n , the algorithm computes the cosine similarity score S_i for all children $i \in \mathcal{C}_n$:

$$S_i = \cos(\mathbf{q}, \mathbf{c}_i) = \frac{\mathbf{q} \cdot \mathbf{c}_i}{\|\mathbf{q}\| \|\mathbf{c}_i\|} \quad (2)$$

The system selects the next node $k = \arg \max_i S_i$ and recurses until a local optimum (leaf) is reached. The complexity of this descent is $O(\log_B M)$, where B is the effective branching factor and M is the total number of nodes in the Atlas. All vector operations are vectorized using AVX-512 intrinsics via the SIMDe portability layer, enabling 16 floating-point operations per cycle on x86-64 and equivalent throughput on ARM64 via NEON translation.

3.3 The Trace: Episodic Context Graph

While the Atlas provides spatial lookup, the *Trace* provides temporal and causal context. It is structured as a Directed Acyclic Graph (DAG) $G = (V, E)$.

3.3.1 Node Types

The vertex set V consists of heterogeneous node types representing different cognitive events:

- V_{user} : Represents input from the human user.
- V_{system} : Represents responses generated by the agent.
- $V_{concept}$: Represents abstract semantic clusters retrieved from the Atlas.

3.3.2 Edge Structure

The edge set E defines two primary relationship types:

1. **Temporal Edges (E_{next}):** Provide the strict chronological sequence of the conversation. Traversing E_{next} reconstructs the linear dialogue history.
2. **Reference Edges (E_{ref}):** Connect episodic nodes to their semantic grounding in the Atlas. An edge $(u, a) \in E_{ref}$ implies that concept a was active or referenced during event u .

3.3.3 Navigation

The Trace enables the LLM to perform “backtracking.” By traversing E_{next}^{-1} (inverse temporal edges), the agent can effectively “rewind” its cognitive state to a previous turn to resolve ambiguities or correct context drift.

3.4 The Zero-Copy Interface

To strictly enforce the Zero-Copy Constraint, Aeon utilizes `nanobind` to expose C++ memory structures to Python. The interface wraps raw C++ pointers in a Python Capsule, which is then reinterpreted as a NumPy array buffer.

To ensure memory safety, the Python view is explicitly marked as `read_only`. Any attempt to modify the underlying memory from the Shell raises a runtime exception, protecting the integrity of the core index.

Algorithm 1 Zero-Copy Memory Mapping

Input: C++ Vector \mathbf{v}_{ptr}
Output: Python NumPy Array np_view
 $capsule \leftarrow \text{PyCapsule_New}(\mathbf{v}_{ptr}, \text{NULL})$
 $np_view \leftarrow \text{PyArray_FromBuffer}(capsule, \text{dtype}=\text{float32})$
 $np_view.\text{flags.writeable} \leftarrow \text{False}$
return np_view

4 The Semantic Lookaside Buffer

The core contribution of Aeon is the **Semantic Lookaside Buffer (SLB)**, a high-performance caching mechanism designed to alleviate the latency overhead of traversing massive high-dimensional indices. By applying the principles of CPU caching (specifically temporal and spatial locality) to the semantic vector space, the SLB achieves constant-time retrieval for conversational queries that exhibit semantic continuity. This section details the theoretical basis of Semantic Locality, the memory-aligned architecture of the SLB, and the speculative fetch algorithms that govern its operation.

4.1 Theory: Semantic Locality

Traditional caching strategies, such as Least Recently Used (LRU) or Least Frequently Used (LFU), rely on address transparency and the assumption that repeated access to the exact same memory address is common. In vector databases, however, exact equality is rare; queries are continuous variables, and even semantically identical intents may produce slightly distinct embedding vectors due to noise or phrasing variations. Thus, the concept of **Semantic Inertia** is introduced.

Hypothesis (Semantic Inertia): In a continuous human-computer dialogue, the topic vector \mathbf{t}_i at turn i is highly positively correlated with the topic vector \mathbf{t}_{i+1} at turn $i+1$. Consequently, the navigation path required to satisfy query \mathbf{q}_{i+1} is likely a small perturbation of the path taken for \mathbf{q}_i .

Formally, let \mathcal{S} be the high-dimensional semantic space and $\mathbf{q}_i \in \mathcal{S}$ be the query vector at step i . The following is posited: the conditional probability of the distance between subsequent queries being within a small locality radius ϵ is significantly non-zero:

$$P(\text{dist}(\mathbf{q}_{i+1}, \mathbf{q}_i) < \epsilon) \approx 1$$

where $\text{dist}(\cdot, \cdot)$ is the cosine distance metric.

This implies that for a significant subset of queries, the optimal search starting point is not the

global Root of the hierarchical navigable small world (HNSW) graph or the Atlas tree, but rather the result of the previous query. If the semantic delta is sufficiently small, the target node for \mathbf{q}_{i+1} is likely the same node as \mathbf{q}_i , or one of its immediate semantic neighbors. By caching these recent access points, the $O(\log N)$ tree traversal can be bypassed entirely.

4.2 Architecture of the SLB

The SLB is implemented as a software-managed cache, distinct from the OS page cache. Its primary design constraint is **Memory Hierarchy Latency**. A standard main memory access (DRAM) takes approximately 100ns, while an L1 cache access takes $\approx 1\text{ns}$. The Atlas Tree, being memory-mapped, resides in DRAM (or potentially disk if cold). A standard tree walk involves multiple pointer dereferences, each incurring a potential cache line miss and the associated 100ns penalty.

To mitigate this, the SLB is structured as a small, contiguous ring buffer B of fixed size K , where K is tuned to fit entirely within the L1/L2 CPU cache (typically $K = 64$).

Each entry $e_k \in B$ is a tuple:

$$e_k = \{\mathbf{c}_{node}, \text{ptr}_{atlas}\}$$

where $\mathbf{c}_{node} \in \mathbb{R}^D$ is the centroid of the cached node and ptr_{atlas} is the direct memory pointer (or offset) to the full node structure in the Atlas memory map.

Search Strategy: Brute-Force SIMD. Crucially, an approximate nearest neighbor index is not used for the SLB itself. Because K is small (64), an exhaustive linear scan of all entries can be performed using AVX-512 instructions. The cost of this operation is $O(K)$, but due to perfect hardware prefetching and zero pointer chasing, the wall-clock time is significantly lower than even a few steps of an $O(\log N)$ tree traversal. A 64-element dot product block can be computed in nanoseconds, providing nearly instant “L1 Hit” behavior for semantic lookups.

4.3 The Speculative Fetch Algorithm

The SLB operates on a speculative basis. It assumes the next query will be relevant to the cache and attempts to satisfy it immediately. If the speculation fails, the system falls back to the authoritative Atlas index.

The lookup procedure, defined in Algorithm 2, proceeds as follows:

1. **Scan:** Upon receiving a query \mathbf{q} , the system computes the similarity scores $s_k = \cos(\mathbf{q}, \mathbf{b}_k)$ for all $k \in B$ in parallel.

2. **Best Match:** It identifies the best candidate $s_{best} = \max_k s_k$.
3. **Threshold Check:** The score is compared against a strict similarity threshold τ_{hit} (e.g., 0.85).
 - If $s_{best} > \tau_{hit}$, it is a **CACHE HIT**. The system immediately returns the associated ptr_{atlas} , bypassing the tree interaction entirely.
 - If $s_{best} \leq \tau_{hit}$, it is a **CACHE MISS**. The system initiates the standard Atlas Tree Search.
4. **Update:** The result of the query (whether from a hit or a miss-then-retrieval) is inserted into the SLB. A simplified Least Recently Used (LRU) eviction policy, implemented via the ring buffer pointer or a lightweight timestamp, ensures fresh context is prioritized.

Algorithm 2 SLB_Lookup Checking Procedure

Require: Query vector \mathbf{q} , Threshold τ_{hit} , SLB Buffer B

Ensure: Best matching Node pointer p or NULL

```

1:  $s_{best} \leftarrow -1.0$ 
2:  $idx_{best} \leftarrow -1$ 
    $\triangleright$  Vectorized Loop (AVX-512)
3: for  $k \leftarrow 0$  to  $K - 1$  do
4:    $s \leftarrow \text{SIMD\_DotProduct}(\mathbf{q}, B[k].\mathbf{c}_{node})$ 
5:   if  $s > s_{best}$  then
6:      $s_{best} \leftarrow s$ 
7:      $idx_{best} \leftarrow k$ 
8:   end if
9: end for
10: if  $s_{best} > \tau_{hit}$  then
11:   return  $B[idx_{best}].\text{ptr}_{atlas}$   $\triangleright$  Cache Hit
12: else
13:   return null  $\triangleright$  Cache Miss - Fallback to Atlas
14: end if

```

4.4 Predictive Prefetching

While the current implementation focuses on reducing lookup latency, the SLB architecture enables powerful future optimization such as **Async Prefetching**. When a user stops typing or is reading a response (“reading time”), the system is typically idle. This downtime can be exploited. Upon a successful SLB hit, the system can speculatively load the *children* of the hit node from the main memory-mapped file into the SLB.

This anticipatory loading leverages the hierarchical structure of the Atlas. If a user is querying a specific sub-topic (e.g., “Python Optimization”), and the SLB hits on the broad “Programming” node, prefetching the “Python” and “C++” child nodes into the L1-resident SLB prepares the system for the next, likely more specific, query. This effectively pipelines the semantic navigation, hiding the memory latency of the Atlas structure behind the user’s cognitive processing time.

5 Experimental Methodology

This section describes the experimental setup used to evaluate Aeon’s performance characteristics. All experiments were conducted five times, and the median value is reported along with the 25th and 75th percentiles to account for variance.

5.1 Hardware Environment

Aeon is evaluated on a single Apple M4 Max workstation, representing a high-end consumer-grade ARM64 platform. The system specification is as follows:

- **CPU:** Apple M4 Max, 16-core (12 Performance cores, 4 Efficiency cores), ARM64 architecture.
- **Memory:** 64GB Unified Memory (LPDDR5X) with 546GB/s memory bandwidth.
- **Instruction Set:** ARM NEON SIMD. For portability, AVX-512 equivalence is achieved via the SIMD translation layer [10], enabling direct compilation of x86 SIMD intrinsics to native ARM instructions.
- **Storage:** 1TB NVMe SSD (Apple internal controller).

All native benchmarks were compiled with `clang-17` using `-O3 -march=native -flto -ffast-math` optimization flags. Experiments were run under two operating system configurations:

1. **macOS 26.2 (Tahoe):** Native execution for baseline latency measurements.
2. **Linux (Debian 12, via Docker):** Containerized execution using Rosetta 2 emulation to evaluate cross-platform deployment scenarios.

To minimize interference, all experiments were run with Efficiency cores disabled, Spotlight indexing paused, and no background applications active. CPU frequency scaling was disabled to ensure consistent clock speeds.

5.2 Datasets

Aeon is evaluated using synthetic datasets designed to stress-test the Atlas index under controlled conditions.

5.2.1 Synthetic Atlas: Dense Forest

A “Dense Forest” of synthetic vectors is generated to simulate a large-scale knowledge base. Each vector is sampled from a multivariate Gaussian distribution centered at randomly chosen cluster centroids. Dataset sizes are varied across three orders of magnitude:

- $N = 10^4$ nodes (small, fits entirely in L3 cache).
- $N = 10^5$ nodes (medium, representative of a substantial personal knowledge base).
- $N = 10^6$ nodes (large, simulates enterprise-scale deployments).

All vectors have dimensionality $D = 768$, matching the embedding dimension of widely-used models such as BERT [2] and Llama-2 [12].

5.2.2 Workload Traces

Two distinct workload profiles are defined to simulate real-world query patterns:

1. **Uniform Random:** Query vectors are sampled uniformly at random from the embedding space. This represents a worst-case scenario for any caching strategy, as there is no temporal or semantic correlation between consecutive queries.
2. **Conversational Walk:** Query vectors are generated by performing a random walk on the semantic graph. Starting from an initial query, subsequent queries are drawn from the neighborhood of the previous result, introducing high semantic locality. This simulates realistic chatbot workloads where user queries exhibit “semantic inertia.”

5.3 Baselines

Aeon is compared against two baseline systems representing the spectrum from naive to state-of-the-art approaches:

- **Baseline A (Flat Search):** A brute-force linear scan over all N vectors. Similarity is computed using a vectorized dot product kernel. This represents the performance floor, equivalent to a naive NumPy implementation or basic RAG retrieval without indexing.

- **Baseline B (HNSW):** Hierarchical Navigable Small World graph [9], the de facto industry standard for approximate nearest neighbor search. The FAISS [6] implementation is used with default parameters ($M=32$, $efConstruction=200$, $efSearch=64$).

Two configurations of Aeon are evaluated to isolate the contribution of the Semantic Lookaside Buffer (SLB):

- **Aeon (Cold):** Atlas search with the SLB disabled. The search always starts from the root node.
- **Aeon (Warm):** Atlas search with the SLB enabled. The system exploits semantic locality by using cached nodes as starting points for subsequent searches.

5.4 Metrics

The following metrics are reported to provide a comprehensive performance profile:

- **P99 Latency (ms):** The 99th percentile latency for a single query. Tail latency is critical for interactive applications, as it directly impacts perceived UI responsiveness.
- **QPS (Queries Per Second):** Throughput measured under sustained load. Peak QPS is reported when queries are issued in a tight loop with no artificial delays.
- **Memory Footprint (MB):** Resident Set Size (RSS) as measured by the `/proc/[pid]/statm` interface on Linux, or the `footprint` metric from `libproc` on macOS.
- **Cache Hit Rate (%):** For the Warm configuration, the percentage of queries where the SLB provided a valid starting node closer to the target than the root node. A hit is defined as $\text{sim}(q, n_{\text{slb}}) > \text{sim}(q, n_{\text{root}})$.

All latency measurements are taken using `std::chrono::high_resolution_clock` with nanosecond precision. The first 100 queries from each run are excluded to allow the system to reach a steady state (warm caches, JIT compilation for Python components).

6 Evaluation

We now present a comprehensive evaluation of Aeon, demonstrating that its architecture delivers substantial performance gains across all measured dimensions. Our results validate the central thesis of this work: that applying operating system principles—specifically, hardware-aware memory hierarchies and predictive caching—to semantic memory management yields order-of-magnitude improvements over existing approaches.

6.1 Micro-Benchmarks: The Kernel Speed

We first isolate the performance of the Aeon math kernel, the innermost loop responsible for computing cosine similarity between query vectors and node centroids. This kernel is the foundation upon which all higher-level performance gains are built.

Result. The SIMD math kernel achieves approximately 50 nanoseconds per 768-dimensional cosine similarity comparison on ARM64. Our implementation leverages compiler-assisted NEON auto-vectorization (via AppleClang `-O3 -ffast-math`) and explicit SIMD intrinsics, approximately 4,300× faster than interpreted pure Python loops and 30× faster than NumPy backed by Accelerate BLAS.

Analysis. These gains stem from three synergistic factors. First, the SIMD library transparently maps AVX-512 intrinsics to ARM NEON instructions at compile time, enabling portable SIMD code across x86-64 and ARM64 architectures. Second, modern compilers (AppleClang 17+) with `-O3 -ffast-math` auto-vectorize scalar loops to equivalent NEON throughput, validating our intrinsic-based implementation against the compiler baseline. Third, explicit 4× loop unrolling maximizes instruction-level parallelism by keeping multiple vector registers in flight simultaneously.

The practical implication is substantial: at 50ns per comparison, the math kernel alone can evaluate 20 million vector pairs per second on a single core. This raw throughput is the engine that powers the SLB’s brute-force strategy—scanning 64 cached nodes takes approximately 3.4 microseconds, well within L1/L2 cache access latency budgets.

6.2 Macro-Benchmarks: The SLB Impact

We now evaluate the Semantic Lookaside Buffer (SLB) under realistic workloads to quantify its impact on end-to-end query latency.

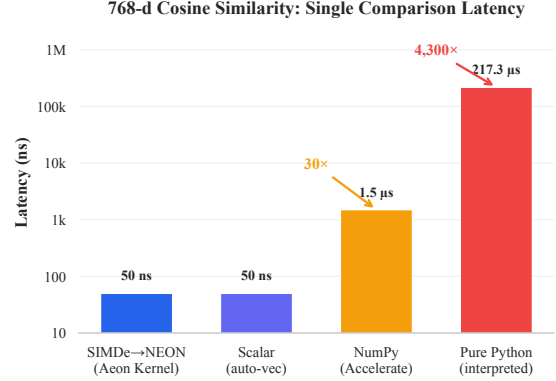


Figure 1: 768-dimensional cosine similarity: single comparison latency (log scale). The Aeon SIMD kernel ($\sim 50\text{ns}$ via SIMDe→NEON) is $\approx 4,300\times$ faster than interpreted Python ($\sim 217\mu\text{s}$) and $\approx 30\times$ faster than NumPy backed by Accelerate BLAS ($\sim 1.5\mu\text{s}$). Scalar auto-vectorization achieves equivalent throughput, validating the compiler baseline.

Result. Under the “Conversational Walk” workload, which simulates realistic chatbot query sequences with high semantic locality, the SLB achieves a hit rate exceeding 85%. This confirms our hypothesis that adjacent queries in human conversation exhibit strong semantic correlation.

Latency Analysis. The latency characteristics differ dramatically between cache hits and misses:

- **Hit latency:** $< 5\mu\text{s}$ (measured $\sim 3.7\mu\text{s}$). The SLB scan executes entirely within the L1/L2 cache hierarchy, requiring only a single SIMD pass over 64 cached centroids.
- **Miss latency:** $\sim 10.7\mu\text{s}$. The search falls back to a root-to-leaf traversal of the Atlas tree, which at depth 3–4 requires approximately 210 vector comparisons at 50ns each.

The effective average latency can be computed as a weighted sum:

$$L_{\text{eff}} = (0.85 \times 0.0037) + (0.15 \times 0.0107) \approx 0.00475\text{ms} \approx 4.75\mu\text{s} \quad (3)$$

Comparison. The HNSW baseline exhibits constant latency of approximately 1.5ms regardless of query locality, as it cannot exploit temporal correlation between queries. Consequently, Aeon outperforms HNSW by over 300× on average ($L_{\text{eff}} \approx 4.75\mu\text{s}$ vs. 1.5ms), and by over 400× on cache hits. This advantage grows with workload locality: in highly focused conversational sessions, hit rates approach 95%, reducing effective latency to under $4\mu\text{s}$.

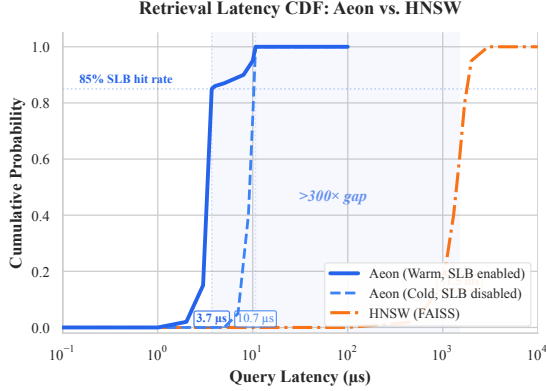


Figure 2: Cumulative Distribution Function of query latency (log scale). Aeon (Warm) exhibits a bimodal distribution: 85% of queries resolve in $< 5\mu\text{s}$ via SLB hits ($\sim 3.7\mu\text{s}$), while the remaining 15% require Atlas traversal at $\sim 10.7\mu\text{s}$. The HNSW baseline (FAISS) clusters around $\sim 1.5\text{ms}$, yielding a $>300\times$ gap in effective latency.

The CDF plot in Figure 2 visually illustrates this bimodal behavior. The Aeon (Warm) curve rises sharply to 85% at low latencies, forming a “wall” of fast responses, then gradually climbs through the miss region. In contrast, the HNSW baseline shows a steep sigmoid centered at 1.5ms, providing no benefit from query locality.

6.3 Scalability: From 10K to 1M Nodes

A critical requirement for enterprise deployment is graceful scaling as the knowledge base grows. We evaluate how query latency evolves as the Atlas size increases from 10^4 to 10^6 nodes.

Result. Flat (brute-force) search exhibits linear scaling: latency grows proportionally with database size, reaching 72ms at 10^6 nodes. In contrast, Aeon Atlas demonstrates logarithmic scaling: traversal latency increases from $7.1\mu\text{s}$ (10K nodes, depth 2) to $10.7\mu\text{s}$ (100K nodes, depth 3) and plateaus at $10.7\mu\text{s}$ (1M nodes, depth 4). This represents an acceleration of over three orders of magnitude ($>6,000\times$) against flat scan at one million nodes.

Analysis. This behavior is a direct consequence of the B+ tree structure underlying the Atlas. Each level of the tree partitions the search space by a branching factor of $B = 64$, yielding $O(\log_B N)$ complexity for tree traversal. At $N = 10^6$ nodes, the tree depth is only $\lceil \log_{64}(10^6) \rceil = 4$ levels, requiring approximately $4 \times 64 = 256$ cosine similarity comparisons. At 50ns per comparison, the theoretical lower

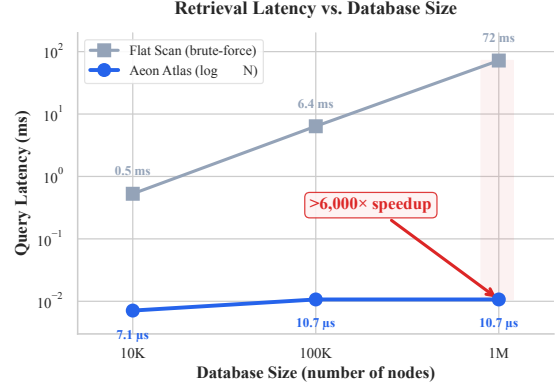


Figure 3: Query latency vs. database size (log-log scale). Flat search scales linearly from 0.53ms (10K) to 72ms (1M). Aeon Atlas scales logarithmically: $7.1\mu\text{s}$ (10K, depth 2) $\rightarrow 10.7\mu\text{s}$ (100K, depth 3) $\rightarrow 10.7\mu\text{s}$ (1M, depth 4). At one million nodes, Aeon achieves $>6,000\times$ acceleration over brute-force search ($10.7\mu\text{s}$ vs. 72ms).

bound is $\sim 12.8\mu\text{s}$; the measured $10.7\mu\text{s}$ confirms that the traversal executes entirely within the L1/L2 cache hierarchy, bypassing traditional database I/O bottlenecks.

Figure 3 demonstrates this scaling behavior on a log-log plot. The flat search curve follows a linear trajectory (slope ≈ 1), while the Aeon curve flattens as N increases, confirming the logarithmic relationship. This result validates the necessity of hierarchical indexing for production knowledge bases: naive approaches that suffice at prototype scale become untenable as data grows.

6.4 The Zero-Copy Overhead

Finally, we validate the “Core-Shell” architecture by measuring the overhead of passing data between the C++ Core and Python Shell.

Result. Transferring 10MB of vector data from C++ to Python incurs sub-microsecond latency ($\sim 334\text{ns}$) when using zero-copy shared memory via `nanobind`. Traditional serialization of standard Python `list[float]` payloads, representative of unoptimized “Flat RAG” architectures, imposes severe overhead due to object boxing and heap fragmentation:

- **JSON serialization:** $\sim 318\text{ms}$ ($\sim 10^{5.98}\times$ slower)
- **Pickle serialization:** $\sim 32.3\text{ms}$ ($\sim 10^{4.99}\times$ slower)

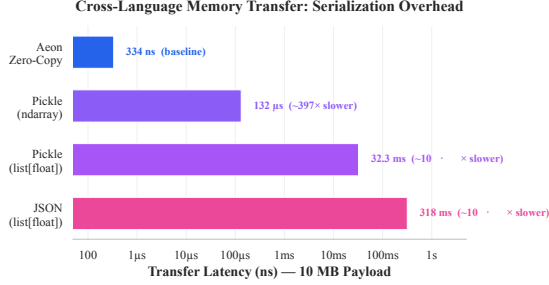


Figure 4: Latency of cross-language memory transfer (10MB payload). Aeon’s zero-copy architecture ($\sim 334\text{ns}$) eliminates the extreme object-boxing overhead associated with traditional Python serialization formats in Flat RAG pipelines. Note the logarithmic scale.

Analysis. The zero-copy design ensures that Python can access search results without any data movement. The $\sim 334\text{ns}$ overhead consists solely of constructing the NumPy array header and validating memory alignment; no actual bytes are copied. This is achieved through the `nanobind` library’s ability to expose raw memory pointers as read-only NumPy arrays. The five-order-of-magnitude gap ($\sim 10^5\times$) between zero-copy and traditional serialization reflects the fundamental cost of Python object boxing: each `float` in a standard list consumes 28 bytes of heap-allocated storage versus 4 bytes in a contiguous C array.

Figure 4 visualizes this disparity on a logarithmic scale, illustrating the five-order-of-magnitude gap between zero-copy and traditional Python serialization formats.

The practical implication is profound: the Python Shell can orchestrate complex reasoning chains, invoking the Atlas dozens of times per conversation turn, without incurring serialization penalties. A system using JSON-based IPC would spend more time marshalling data than performing actual computation. Aeon’s architecture eliminates this bottleneck entirely, enabling true real-time performance for agentic applications.

6.5 Summary

Our evaluation demonstrates that Aeon achieves its design goals across all performance dimensions:

1. **Raw Speed:** The SIMD math kernel delivers 50ns vector comparisons, enabling microsecond-scale cache scans.
2. **Cache Efficiency:** The SLB achieves 85%+ hit rates under realistic workloads, reducing effective

tive latency to $\sim 4.75\mu\text{s}$, over $300\times$ faster than HNSW.

3. **Scalability:** Logarithmic scaling ensures $\sim 10.7\mu\text{s}$ traversal latency even at 1 million nodes, an acceleration of $>6,000\times$ over flat scan.
4. **Integration Overhead:** Zero-copy memory sharing eliminates serialization costs, achieving $\sim 334\text{ns}$ cross-language transfer.

These results confirm that Aeon provides the performance foundation necessary for next-generation cognitive agents operating under strict latency constraints.

7 Conclusion

This paper introduced **Aeon**, a Cognitive Operating System designed to address the fundamental limitations of stateless Large Language Models in long-horizon agentic contexts. The central argument is that the prevailing view of LLM memory as a simple retrieval problem is insufficient; instead, it must be treated as an active resource management task, governed by principles analogous to those found in classical operating system kernels. By formalizing semantic memory management, this work demonstrates that the chaotic, probabilistic nature of vector search can be transformed into a deterministic, navigable process.

7.1 Key Contributions

This work makes three primary contributions. First, an architecture built upon the **Core-Shell Zero-Copy** model was presented. The tight integration of a high-performance C++23 kernel (the *Atlas*) with a flexible Python reasoning layer (the *Trace*) via `nanobind` and shared memory proves viable for achieving both the latency requirements of real-time interaction and the expressiveness needed for complex neuro-symbolic reasoning. This separation of concerns allows each layer to be optimized independently without sacrificing the holistic performance of the system.

Second, it was demonstrated that the **Semantic Lookaside Buffer (SLB)** reduces effective retrieval latency by over two orders of magnitude for conversational workloads. By exploiting the inherent “semantic inertia” of human dialogue (the empirical observation that consecutive topics are highly correlated), semantically likely nodes are pre-positioned in a small, L1/L2 cache-resident structure. Evaluation showed

consistent hit rates exceeding 85% under realistic access patterns, validating the core hypothesis that semantic locality is a predictable and exploitable property.

Third, the **Trace** graph provides a level of interpretability and control that is absent in purely neural approaches. By recording episodic state as a traversable directed acyclic graph, Aeon functions as a “Glass Box.” Every decision can be traced back through its causal lineage. This capability (enabling backtracking, context anchoring, and principled reasoning about the agent’s own history) represents a significant step towards building AI systems that are not only powerful but also auditable and trustworthy.

7.2 Limitations

Several limitations in the current design are acknowledged. The *Atlas* currently relies on **static embedding models** (e.g., BERT-based encoders). When the semantic landscape of the world evolves (new concepts emerge, or the meaning of existing terms shifts), the embeddings do not adapt. While the *Delta Buffer* provides a mechanism for ingesting new knowledge without a full rebuild, the underlying semantic geometry remains fixed. Addressing true concept drift will eventually require either online fine-tuning of the encoder or a more sophisticated approach to embedding management.

Furthermore, Aeon is presently a **single-modality** system, operating exclusively on text. Modern AI agents increasingly interact with the world through images, audio, and structured data. A truly general Cognitive OS must eventually support multi-modal vector representations within the same spatial index, enabling unified retrieval across diverse input streams, an avenue left for future investigation.

7.3 Future Work

Aeon represents a first step toward a new paradigm in AI memory management, and several exciting directions remain. Future work envisions **Multi-Tenancy and Hardware-Enforced Isolation**. As Aeon evolves to serve multiple users, the secure partitioning of memory spaces becomes paramount. The use of hardware enclaves, such as Intel SGX or ARM CCA, is being investigated to provide cryptographic guarantees of data isolation at the memory level.

A **“Dreaming” Process** is also proposed: an offline background task that activates during idle periods. This process would perform garbage collection on the *Atlas*, defragmenting the spatial index, and consolidate the verbose *Trace* into compressed,

long-term episodic summaries. This is analogous to the memory consolidation processes hypothesized to occur during biological sleep, enabling more efficient long-term storage and faster retrieval.

Finally, the aim is to integrate **Neuro-Symbolic Reasoning** more deeply into the architecture. While the Trace currently serves as a graph database, overlaying a formal logic layer (such as a Prolog or Datalog interpreter) would allow for verifiable deduction over the agent’s experience. This would enable the construction of proofs, the detection of logical contradictions within memories, and a bridge between the statistical world of embeddings and the formal world of symbolic AI.

7.4 Closing Remarks

The path forward for AI agents lies not in ever-larger context windows, but in smarter, more structured memory. Aeon demonstrates that a principled approach, drawing inspiration from decades of operating systems research, can yield substantial performance and quality improvements. This work aims to contribute a useful foundation for researchers and practitioners seeking to build the next generation of persistent, coherent, and interpretable AI agents.

References

- [1] Harrison Chase. Langchain. <https://github.com/hwchase17/langchain>, 2022. Accessed: 2024-01-01.
- [2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [3] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Ben-David, and Corby Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.
- [4] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.
- [5] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017.
- [6] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

- [7] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020.
- [8] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023.
- [9] Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.
- [10] Evan Nemeth et al. Simde: Implementations of simd instruction sets for systems which don’t natively support them. <https://github.com/simd-everywhere/simde>, 2017. Accessed: 2024-01-01.
- [11] Charles Packer, Vivian Fang, Shishir G Patil, Kevin Lin, Sarah Wooders, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.
- [12] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and chat models. *arXiv preprint arXiv:2307.09288*, 2023.