

Out-of-Core Dynamic Reasoning and Inference: Scaling Hybrid Graph Traversal to Petabyte-Scale Vector Databases

HyperStreamDB Team
Research & Development
contact@hyperstreamdb.org

September 17, 2026

Abstract

Graph Retrieval-Augmented Generation (GraphRAG) has emerged as a methodology for enabling Large Language Models (LLMs) to reason over complex, multi-hop queries. However, current GraphRAG implementations face significant memory bounds, fundamentally limiting their scalability to large-scale enterprise workloads. Standard graph traversals traditionally require loading entire network topologies into RAM. In contrast, modern vector databases have addressed this memory bottleneck for semantic search by leveraging disk-backed, memory-mapped indices (e.g., HNSW). In this paper, we introduce the unified hybrid architecture of HyperStreamDB, an engine that adapts the out-of-core indexing strategies of modern vector databases for the graph domain. By integrating DataFusion’s global execution engine and a hardware-accelerated memory-mapped HNSW index with a novel property-aware memory-mapped Compressed Sparse Row (CSR) index, this architecture enables zero-copy, out-of-core Dynamic Reasoning and Inference for Federated Texts (DRIFT) search at petabyte scale, mitigating the memory wall constraint.

1 Introduction

1.1 The Rise of GraphRAG

Large Language Models (LLMs) have demonstrated strong capabilities in generating natural language. However, their reliance on static, parameter-bound knowledge makes them susceptible to hallucination and limits their ability to reason over private or rapidly evolving datasets. Retrieval-Augmented Generation (RAG) mitigates this by grounding the model in retrieved context. Moving beyond simple semantic similarity, GraphRAG techniques organize knowledge into interconnected entities, allowing LLMs to perform complex, multi-hop retrieval to answer faceted queries that span discrete pieces of information.

1.2 The Memory Wall and the Vector Database Paradigm

Despite its analytical capabilities, GraphRAG struggles with scalability in production environments due to the "Memory Wall". While modern vector databases have achieved highly efficient out-of-core semantic search (e.g., using disk-backed HNSW graphs) capable of handling petabytes of dense embeddings, graph databases face challenges scaling at the same rate. Graph traversal operations still predominantly require loading network data structures entirely into RAM (e.g., NetworkX, petgraph).

For massive datasets, this translates to terabytes of required memory. Partitioning the graph across a distributed database cluster introduces prohibitive network latencies during multi-hop traversals, limiting real-time inference capabilities. A promising approach involves applying the out-of-core indexing paradigms prevalent in vector databases to graph traversal.

1.3 Our Contribution

We introduce HyperStreamDB’s unified Hybrid Transactional/Analytical Processing (HTAP) engine. This system brings the architectural strengths of a petabyte-scale vector database to bear on GraphRAG. It combines out-of-core global semantic search via an existing hardware-accelerated memory-mapped HNSW with zero-copy local graph traversal via a newly developed Property-Aware CSR memory map. By linking relational storage, memory-mapped vector indices, and memory-mapped graph topologies, we propose an architecture capable of out-of-core DRIFT search that scales to petabytes of data on standard hardware footprints.

2 Background and Related Work

2.1 GraphRAG and DRIFT

Microsoft’s GraphRAG framework introduced a hierarchical approach to querying, summarizing global communities and aggregating local evidence. The DRIFT methodology formalizes this by balancing global semantic clustering with iterative, localized graph exploration, allowing an LLM to navigate a knowledge graph logically.

2.2 Out-of-Core Graph Processing

Significant research has been dedicated to processing graphs that exceed main memory. Systems like Ligra-mmap, Chaos, and Kaleido leverage secondary storage and memory mapping to compute graph analytics. However, these systems are designed for offline analytics (e.g., PageRank, Connected Components) rather than

real-time, low-latency point queries integrated directly with dense vector retrieval.

2.3 HTAP Systems

Hybrid systems like GART seek to bridge the gap between transactional relational data and graph analytics. HyperStreamDB extends this precedent by integrating dense vector embeddings directly into the HTAP paradigm, ensuring that semantic queries can seamlessly transition into graph traversals.

3 System Architecture

HyperStreamDB is built around several core components designed to minimize serialization overhead and maximize disk throughput.

3.1 Unified Storage

At the foundation, Parquet and Arrow tables serve as the single source of truth for both dense vectors and graph edges. This columnar layout ensures highly efficient analytical scans while maintaining structured relationships.

3.2 Out-of-Core Global Search

Global queries and community detection algorithms are executed via the Apache Arrow DataFusion engine. For semantic matching (the "Primer" phase of DRIFT), HyperStreamDB utilizes an existing hardware-accelerated, memory-mapped HNSW index. This ensures that the initial semantic search can quickly locate relevant entry points within the global dataset without requiring the dataset to reside in RAM.

3.3 Property-Aware Memory-Mapped CSR Index

To facilitate local traversal, we introduce a memory-mapped CSR index tailored for properties. Standard CSR arrays represent edges using source offsets and destination arrays.

Novelty: We extend the destination array to store `(dst_id, row_id)` tuples. When navigating from a source node to a neighbor, the traversal engine immediately resolves the underlying `row_id`. This achieves $O(1)$ property retrieval directly from the Parquet/Arrow backend, entirely avoiding secondary table scans or joins during multi-hop exploration.

3.4 The Hybrid Execution Orchestrator

The orchestrator manages the lifecycle of a query. It can seamlessly pivot from an HNSW vector search result directly into a CSR local traversal within microseconds. Because both indices are memory-mapped, the operating system’s page cache governs data locality, ensuring that hot graph regions remain in memory while cold regions are evicted without application-level memory management overhead.

4 Empowering Application-Layer Single-Line Prompts

In a standard GraphRAG setup, local search typically requires a user or application to explicitly specify a seed entity to anchor the graph traversal. To compress this local seed requirement into a frictionless, single-line search prompt, modern GraphRAG applications can implement a **Hybrid Seed Discovery Layer**. While this orchestration resides strictly in the application layer (e.g., LangChain or LlamaIndex), HyperStreamDB provides the requisite HTAP primitives—fusing dense vector embeddings and discrete entity IDs—to enable it without secondary system calls.

4.1 Application-Layer Dual-Index Seed Locator

To avoid forcing the end-user to explicitly specify graph nodes, the application can leverage HyperStreamDB to index graph entities and text chunks into a shared coordinate space:

- **Entity Embedding Index:** Every entity node in the graph stores a dense vector computed from its canonical name, description, and entity type.
- **Lexical/Inverted Index:** An exact-token or BM25 sparse index over canonical entity names and aliases.

When a single-line prompt arrives, the application queries HyperStreamDB across both indices simultaneously. The highest-scoring nodes, determined by a weighted combination of dense vector similarity and BM25 lexical scores, become the anchor seeds for the local graph traversal.

4.2 Application-Side Speculative Pre-Flight Extraction

For queries containing explicit entity mentions that must override fuzzy vector matching, the application layer employs speculative pre-flight extraction. This is achieved by running an in-memory Aho-Corasick automaton across the query against a vocabulary of all canonical graph entities (exported from the DB), or via a fast, small-footprint micro-NER model. Exact matches immediately seed the traversal with zero LLM overhead.

4.3 Seed Set Fusion & Pruning

Because a single query can return noisy seed candidates, a fusion and pruning step occurs before graph expansion. We apply degree-weighted centrality filtering to penalize generic "hub" entities unless explicitly named in the prompt. We then perform score normalization using reciprocal rank fusion (RRF) between lexical and semantic hits, capping the expansion to $K \leq 5$ seed nodes to avoid context window blowup and redundant multi-hop fanout.

5 Implementing Out-of-Core DRIFT

The DRIFT algorithm is mapped directly onto the HyperStreamDB architecture through a

three-phase lifecycle:

1. **Semantic Primer:** A user query is embedded and searched against the hardware-accelerated MMap HNSW and BM25 indices to derive the initial seed nodes.
2. **Subgraph Follow-up:** The execution engine transitions to the MMap CSR index. Starting from the seed nodes, it traverses the graph, utilizing the `row_id` lookup to eagerly fetch node and edge properties required to build the local context window.
3. **LLM Evaluation & Iteration:** The gathered subgraph is presented to the LLM. If the LLM determines more information is needed, it generates follow-up queries or specifies traversal paths, re-triggering the subgraph follow-up phase.

6 Evaluation & Benchmarks

[Note: Empirical data to be populated pending large-scale cluster execution]

6.1 Experimental Setup

We evaluate the system against a multi-terabyte dataset (e.g., Wikipedia + citations). The baseline for comparison is a standard Python orchestrator (LlamaIndex) querying a disaggregated stack: Milvus for vector search and Neo4j for graph traversal over a network boundary.

6.2 Metrics

- **Memory Footprint:** Peak RAM usage during traversal. HyperStreamDB is expected to maintain a near-constant footprint irrespective of graph size, bounded only by the OS page cache.
- **Latency:** End-to-end latency per DRIFT round, measuring the elimination of network hops and serialization overhead.
- **Scalability:** Performance degradation as the graph size is artificially scaled beyond the available physical RAM.

7 Conclusion and Future Work

HyperStreamDB demonstrates that the Memory Wall in GraphRAG can be overcome by deeply integrating memory-mapped vector and graph indices directly with columnar storage. This unified engine unlocks unprecedented reasoning capabilities for LLMs over massive datasets. Future integrations will explore distributed cluster-level graph partitioning to scale beyond single-node NVMe bounds, as well as native support for multi-modal edge properties.