Part II CST Project Proposal

A Hybrid Graph-Vector Database in OCaml

Student: Oliver Fogelin, Homerton College

Day-to-day Supervisor: Ryan Gibb

Marking Supervisor: Professor Jon Crowcroft

Director of Studies: Dr John Fawcett

1 Introduction

Retrieval-Augmented Generation (RAG) grounds LLM responses in external data. The naive approach to implementing RAG is to embed relevant documents with a pre-trained embedding model, store them in a vector database, and then return the top k closest documents to an LLM's embedded query during a generation step.

This approach is often insufficient for high-quality LLM responses, as dense retrieval (via embedding closeness) has been shown to underperform more traditional keyword-based sparse retrieval (e.g. BM25[1]) on several BEIR datasets.[2] In practice, merging sparse and dense retrieval results improves recall and downstream RAG accuracy over using either in isolation.[3]

Recent hybrid sparse/dense retrieval systems such as GraphRAG[4] have further demonstrated the value of graph structure in the sparse retrieval component. By carefully setting up a knowledge graph to expose semantically meaningful edges between entities, models can be augmented with much more powerful search and retrieval capabilities over embedded external data. However, most such systems are text-only, built as offline batch processes, and slow to update. Their indices are typically reconstructed in large jobs rather than incrementally maintained, and they rarely target lightweight, embeddable deployments. Likewise, general-purpose graph databases with vector add-ons tend to have runtime overhead and operational complexity that is undesirable for a small, embedded engine.

Over the course of the project, I will produce a single-machine graph-vector database in OCaml. The graph store will behave as a conventional graph database, allowing CRUD[5] operations on nodes, edges, and their associated types and metadata. The vector addition will allow the user to link a number of labelled vectors to a node or edge, and perform semantic search queries over the nodes and edges using the vectors. The graph store will

[1] Best Matching 25 (BM25). `https://en.wikipedia.org/wiki/Okapi_BM25`

[2] Thakur et al. 2021; Izacard et al. 2022.

[3] Sawarkar, Mangal, and Solanki 2024.

[4] Microsoft Research 2024.

[5] Create, read, update, delete.

be built on top of LMDB[6] as an underlying B+-tree storage engine. I will implement a memory-mapped HNSW[7][8] index with incremental updates to serve as the vector store, and surface a unified interface for the two stores that keeps vector state consistent with ongoing graph transactions. This interface will be served via Cap'n Proto[9] to allow database clients in a variety of languages to execute hybrid graph CRUD/semantic search transactions.

OCaml's strong type system, performance characteristics, and lightweight concurrency primitives make it fit well for this project, both in ease of development and for resulting performance/safety guarantees. Mature OCaml libraries exist for both Cap'n Proto and LMDB, along with Bigarray[10] for efficient memory-mapped arrays and parsing tooling (Menhir[11], Angstrom[12]) for query language extensions. OxCaml[13] will also be useful for performance improvements as an extension, through its allocation control and safe concurrency type system.

2 Starting Point

I have previously explored a similar project in C++, `https://github.com/olifog/stardust`. This project gave me a grounding in the basics of Cap'n Proto and LMDB, and some experience in graph database internals. I have limited experience with OCaml, exclusive to knowledge from the CST courses. An existing set of LMDB bindings in OCaml, `https://github.com/Drup/ocaml-lmdb`, will be used as a starting point for the graph store.

3 Project Structure

The core of this project is a unified interface on top of two memory-mapped stores. A KV store[14] will back node metadata, edge metadata, and adjacency lists. An HNSW index will store vector data to support efficient semantic search. The interface will keep transactions consistent between the two stores, incrementally adding/removing vectors to the HNSW index using snapshot epochs based on node/edge reads and writes to the graph store. This interface will be consumed by a Cap'n Proto server that exposes a cross-language protocol to communicate with the database.

Design goals are correctness first (with the KV store as the source of truth), predictable resource usage suitable for embedded deployments, incremental indexing to avoid long rebuilds, and as high performance as possible given these constraints. The concrete file

[6] Lightning Memory-Mapped Database. `https://www.symas.com/lmdb/`
[7] Hierarchical Navigable Small Worlds (HNSW)
[8] Malkov and Yashunin 2020.
[9] `https://capnproto.org/`
[10] `https://ocaml.org/api/Bigarray.html`
[11] `https://gitlab.inria.fr/fpottier/menhir/`
[12] `https://github.com/inhabitedtype/angstrom`
[13] `https://oxcaml.org/`
[14] Key-value (KV) database. `https://en.wikipedia.org/wiki/Key-value_database`

format and epoch/transaction concurrency mechanics of the memory-mapped HNSW index will be finalised during implementation.

The project can be further broken down into a number of components, in dependency order, with each component generally building on the previous:

- LMDB environment and database setup, with a wrapper exposing graph CRUD operations. The LMDB database key structure will need to be carefully designed to be optimal for the most common expected operations.
- A memory-mapped HNSW index implementation, with a thin wrapper exposing vector search operations. Significant engineering work will be needed here to allow delete operations, using tombstones and epoch swaps or periodic rebuilds.
- A unified interface over the two stores.
- A careful synchronization mechanism between the two stores, or manager on top of the HNSW index that creates multiple epochs of memory-mapped HNSW slabs.
- A Cap'n Proto schema and server that exposes the unified interface.
- A test suite and benchmark suite for the database.

4 Possible Extensions

- Exploit Cap'n Proto promise pipelining to reduce round-trips for batched transactions and operations.
- Crash safety, on restart the database should rebuild affected HNSW epochs.
- Add a Cypher parser and query executor in addition to the raw graph/vector operations.
- Push performance to be competitive (or beat) existing graph-only and vector-only peers using OxCaml extensions.

5 Success Criteria

- An LMDB-backed graph store with CRUD and graph traversal operations.
- A memory-mapped HNSW implementation with add/delete/approximate k-NN[15].
- A unified transactional API with incremental HNSW updates on graph transactions.
- A Cap'n Proto server with a minimal client to run tests and benchmarks.
- Performance (both latency and throughput) is close to existing graph-only and vector-only peers for their respective operations.

[15] k-nearest neighbours (k-NN). `https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm`

6 Evaluation Plan

The primary dataset that will be used to evaluate the project is a subgraph of Wikidata combined with a subset of Wikipedia.

1. **Correctness**: Graph/vector CRUD operations will be verified through a suite of unit tests and an integrity checker.
2. **HNSW Quality**: HNSW retrieval will be evaluated against ground truth brute-force nearest-neighbour search to compute recall@k ($k \in \{10, 50\}$) across a range of sample vector queries. This recall, along with recorded average/99th percentile latency/throughput for these queries, will be compared against a number of other existing vector databases running the same queries on the same data.
3. **Graph Performance**: Basic graph workloads such as traversal, insertion, and querying will be benchmarked against contemporary dedicated graph databases to verify that the system does not overly sacrifice graph latency and throughput for the sake of vector operations. Throughput-latency curves will be generated for each workload and compared across a range of contemporary dedicated graph databases.
4. **(Optional/extension) Hybrid Retrieval**: To demonstrate the broader utility of a combined graph-vector design, an optional Graph-RAG prototype may be built on top of the database. This experiment will test whether hybrid and graph-augmented retrieval improves recall and ranking quality over sparse or dense retrieval alone.

7 Plan of Work

Regular bi-weekly supervisor meetings are scheduled throughout the project to review progress, discuss technical decisions, and address emerging challenges.

7.1 Michaelmas Term

- **Week 2/3 (16/10/25 - 29/10/25)**
  - Invest time into learning production-grade OCaml and OxCaml.
  - Review the LMDB API and design initial key structure for the graph store.
  - **Supervisor meeting:** Present and discuss proposed LMDB key structure and graph data model. Review overall system architecture and component boundaries.
- **Week 4/5 (30/10/25 - 12/11/25)**
  - Refine key structure based on supervisor feedback.
  - Implement the LMDB wrapper and graph store.
  - Establish a scalable build system, test suite, and benchmark suite.

- **Supervisor meeting:** Code review of LMDB wrapper implementation. Discuss testing strategy and benchmark design.
- **Milestone:** LMDB-backed graph store with CRUD and graph traversal operations.

- **Week 6/7 (13/11/25 - 26/11/25)**
  - Write comprehensive tests for the graph store.
  - Write brute force nearest-neighbour search for testing purposes.
  - Begin HNSW index design: research memory-mapped format and epoch management strategies.
  - **Supervisor meeting:** Review test coverage and graph store performance. Present HNSW design proposal including file format, delete strategy, and epoch synchronization approach.
  - **Milestone:** Comprehensive test suite for the graph store.

- **Week 8 (27/11/25 - 03/12/25)**
  - Create a benchmarking/testing suite for vector retrieval.
  - Refine HNSW design based on supervisor feedback.
  - **Supervisor meeting:** Finalize HNSW design decisions. Discuss potential risks and mitigation strategies for Christmas vacation implementation.
  - **Milestone:** Testing suite for vector retrieval comparing to initial brute force search as a baseline.

7.2 Christmas Holidays

- **Week 1/2/3 (04/12/25 - 24/12/25)**
  - Implement the HNSW index and wrapper following agreed design.
  - Fully test and benchmark the HNSW index.
  - Design unified interface and transaction manager: specify consistency guarantees and transaction semantics.
  - **Supervisor meeting (mid-vacation):** Review HNSW implementation and performance results. Present unified interface design and discuss transaction synchronization approach.

- **Week 4/5 (25/12/25 - 07/01/26)**
  - Implement the unified interface and transaction manager based on supervisor feedback.
  - Address any bugs or performance issues identified in HNSW testing.
  - **Milestone:** Single interface for a joint graph-vector database, with functional HNSW index.

- **Week 6/7 (08/01/26 - 21/01/26)**

  - Design and implement the Cap'n Proto schema and server.
  - Expand the test suite and benchmark suite to cover all evaluation criteria.
  - **Supervisor meeting:** Review transaction manager implementation and Cap'n Proto API design. Discuss whether success criteria are met and identify any gaps.
  - **Milestone:** Cap'n Proto server with a minimal client to run tests and benchmarks.

7.3 Lent Term

- **Week 1/2 (22/01/26 - 04/02/26)**
  - Refactor and clean up the codebase, document core code.
  - Address any remaining issues from supervisor feedback.
  - Write the progress report.
  - *Extension:* Performance improvements with OxCaml extensions.
  - **Supervisor meeting:** Review final implementation against success criteria. Discuss progress report draft and presentation approach.
  - **Milestone:** Success criteria met. Progress report complete. (Due 06/02/26)
- **Week 3/4 (05/02/26 - 18/02/26)**
  - Prepare slides for and deliver progress presentation.
  - *Extension:* Cypher parser and query executor.
  - **Supervisor meeting:** Discuss extension priorities and dissertation structure. Plan evaluation benchmarks.
- **Week 5/6 (19/02/26 - 04/03/26)**
  - Run comprehensive evaluation benchmarks and collect results.
  - Dissertation - Write Chapter 1: Introduction and Chapter 2: Preparation.
  - *Extension:* Crash safety for HNSW index.
  - **Supervisor meeting:** Review evaluation results and discuss dissertation Introduction/Preparation chapters.
- **Week 7/8 (05/03/26 - 18/03/26)**
  - Dissertation - Write Chapter 3: Implementation.
  - *Extension:* Cap'n Proto Promise Pipelining.
  - **Supervisor meeting:** Review Implementation chapter draft. Discuss technical depth and clarity.

7.4 Easter Holidays

- **Week 1/2 (19/03/26 - 01/04/26)**
  - Dissertation - Write Chapter 4: Evaluation.
  - **Supervisor meeting:** Review Evaluation chapter draft and discuss results presentation.
  - *Extension:* Finish up/extend in-progress extension work.
- **Week 3/4 (02/04/26 - 15/04/26)**
  - Dissertation - Write Chapter 5: Conclusions.
  - Share full draft with supervisor and DoS for comprehensive review.
  - **Supervisor meeting:** Discuss overall dissertation structure, coherence, and areas for improvement.
  - **Milestone:** Full dissertation draft complete and shared for review.
- **Week 5/6 (16/04/26 - 29/04/26)**
  - Address feedback from supervisor and DoS.
  - **Supervisor meeting:** Final review of revised dissertation. Discuss any remaining issues.
  - **Milestone:** Reviewed dissertation draft complete.

7.5 Easter Term

- **Week 1/2 (30/04/26 - 13/05/26)**
  - Add any last extension work to dissertation.
  - Finalise dissertation, last round of revisions.
  - Prepare source code for submission.
  - **Milestone:** Dissertation submitted on or before 15/05/26.

8 Resource Declaration

I will be working on my personal computer, a 16-inch 2021 Macbook Pro with 16GB of RAM. All code will be written on this local device and backed up to a private GitHub repository. I am fully responsible for this computer and regularly back it up to a secondary external drive, with a separate Linux laptop usable as a backup. Any LLM API calls needed for RAG evaluation will be provided through AWS Bedrock, in which I have a personal account with a sufficient number of existing credits. I will be writing my dissertation on my local device in LaTeX.

References

Izacard, Gautier et al. Contriever: Unsupervised Dense Information Retrieval with Contrastive Learning. URL: https://arxiv.org/abs/2112.09118.

Malkov, Yu. A. and D. A. Yashunin. *"Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs"*. URL: https://doi.org/10.1109/TPAMI.2018.2889473.

Microsoft Research. GraphRAG. URL: https://www.microsoft.com/en-us/research/project/graphrag/.

Sawarkar, Kavish, Abhishek Mangal, and S. R. Solanki. Blended RAG: Improving RAG Accuracy with Semantic Search and Hybrid Query-Based Retrievers. URL: https://arxiv.org/abs/2404.07220.

Thakur, Nandan et al. *"BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models"*. URL: https://arxiv.org/abs/2104.08663.