Domain-Specific AI Code Search with LLMs as a Ranker

Course: COM S 402 – Senior Design

Client: Dr. Phan

Instructor: Dr Mitra

TAs: shaiqur

Team Members:

- Carter Cutsforth
- Jacob Duba
- Keenan Jacobs
- Conor O’Shea
- Diego Perez

Letter from the Client

[To be attached by the TAs]

Chapter 1: Requirements

Functional Requirements

1. Natural Language Input
2. Comment-Code Embedding using UniXCoder
3. Vector Similarity Search
4. LLM-Based Ranking via OpenRouter
5. Searchable Dataset: CodeSearchNet, others
6. Code Snippet Output with confidence
7. Basic UI for user interaction

Non-Functional Requirements

1. Query response under 5 seconds
2. Scalable to large datasets
3. Robust to API failures (including retries or fallback behavior for LLM responses)
4. Secure internal codebases
5. Modular design
6. Intuitive UI

Tech Stack

Python, Flask, UniXcoder, DeepSeek via OpenRouter, GitLab, SQLite, Ollama-compatible LLMs

Chapter 2: Design and API

The architecture consists of a backend for embedding generation and vector search, a Flask-based web frontend, and an LLM-based re-ranking system using OpenRouter. Embeddings are stored in SQLite for fast access. LLM scoring is invoked via prompt engineering.

API Design:
- `POST /`: Accepts a natural language query
- Returns: Top 10 code snippets before and after LLM ranking

Chapter 3: Work Done by Team

Keenan Jacobs:
Led integration of LLM-based ranking into the backend and implemented the scoring prompt. Built the entire frontend including JavaScript, CSS, and HTML. Reorganized Flask routes to support dual-column display of results. Added loader animations, copy-to-clipboard buttons, and UX polish. Also contributed to the final website, report, and presentation.

Diego Perez:
Built data embedding pipeline and storage architecture. Explored storage and API options. Optimized performance by rewriting the pipeline to store and retrieve embeddings using SQLite. Implemented GPU acceleration for batch embedding generation. Worked closely with Jacob on performance tuning and system architecture for Demo 2. Prepared the system to handle new datasets as provided by the client.

Conor O'Shea:
Built and validated the core vector similarity search algorithm. Verified embedding quality using UniXcoder. In Demo 2, developed benchmarking logic critical for Dr. Phan's research and coordinated local testing with Ollama-based LLMs. Planned cross-dataset benchmarking strategy for final evaluation.

Carter Cutsforth:
Set up datasets, handled environment setup, and helped decide frontend UI plans. In Demo 2, worked on integrating LLM benchmarking. Explored running the Flask server on ISU's network and implemented the Ollama changes.

Jacob Duba:
Scaffolded the initial codebase and managed the GitLab board. For Demo 2, broke down the project into smaller implementation tickets including: SQLite storage, GPU batching, numpy-based vector search, and SSR-style snippet return. Collaborated closely with Diego and Conor on performance tuning and planning the live demo. Provided support for final benchmarking and presentation coordination.

Chapter 4: Results Achieved

- Vector similarity search implemented using UniXcoder embeddings and dot product scoring.

- SQLite database integration for efficient, persistent storage and retrieval of code embeddings.

- Flask-based web application with a user-friendly interface for natural language code search.

- Dual-phase retrieval: initial vector search followed by LLM re-ranking implemented locally using DeepSeek models through Ollama

- Fully functional UI supporting side-by-side comparison of search results before and after LLM re-ranking.

- Copy-to-clipboard and loading animation features enhance usability and user experience.

- Benchmarking framework created to evaluate Mean Reciprocal Rank (MRR) using re-ranked results.

- Local LLM inference supported using Ollama for improved speed and reduced dependency on external APIs.

- JSON-based demos deployed for validation, and performance tested across different datasets and embedding configurations, later updated to SQLite

Appendix: Demo Slides

See attached slides: COMS_402_SD18_DEMO1.pptx and COMS_402_SD18_DEMO2.pptx