Semantic Search in Codebase
KDTrees, Abstract Syntax Trees and MinHeaps

Aditya Deshmukh (612203036)
Saurabh Deulkar (612203038)
Avadhoot Ghewade (612203057)
Pramod Dudhal (612203049)
Batch T4 Div 1

March 28, 2025

Department of Computer Science and Engineering
COEP Technological University
Academic Year 2024-2025

Contents

1 Abstract

Codebases tend to grow significantly over time, making it increasingly difficult for developers to locate relevant functions and methods. This project introduces a command-line tool that enables developers to search their codebase using semantic queries. By leveraging embeddings generated from machine learning models and advanced data structures like KD-Trees and min-heaps, the tool efficiently retrieves relevant functions based on semantic similarity.

2 Problem Statement

Developers often struggle to find specific code snippets within large repositories. Traditional keyword-based searches are limited in their ability to capture the semantic meaning of functions. This project aims to bridge that gap by implementing a semantic search mechanism that enables intuitive and efficient code retrieval.

3 Aim

To develop a CLI-based tool that allows developers to perform semantic searches on their codebase, retrieve relevant functions efficiently, and cluster similar code snippets for better organization and analysis.

4 Project Objectives

1. Implement a CLI-based search tool that allows developers to find functions and methods using semantic queries.
2. Generate embeddings for code snippets using advanced machine learning models.
3. Extract relevant nodes from the codebase using an Abstract Syntax Tree (AST) via Tree-sitter.
4. Utilize efficient data structures such as KD-Trees and min-heaps for fast retrieval and clustering.
5. Provide clustering functionality to identify and group similar code snippets.
6. Enable seamless integration with code editors for quick access to retrieved functions.

5 Flowchart

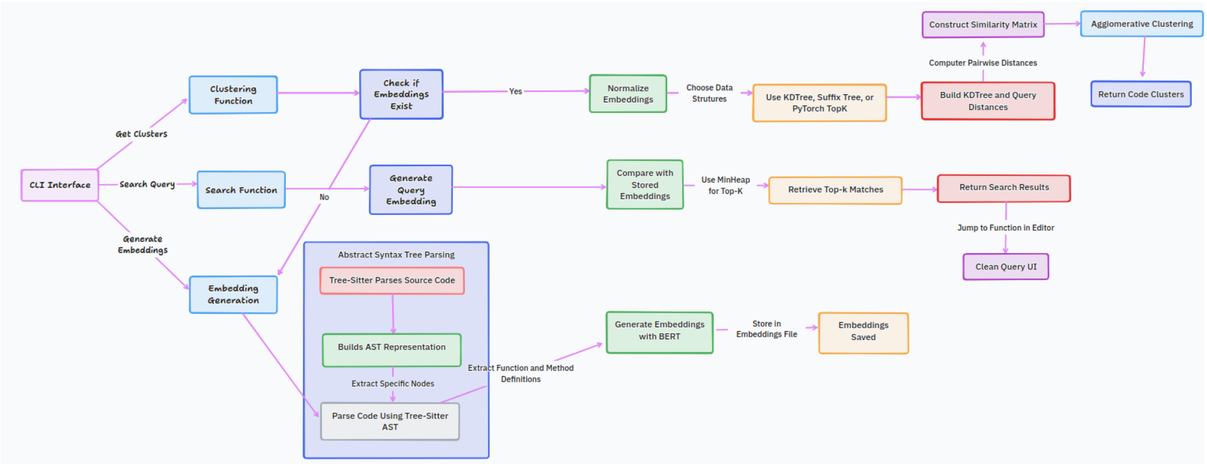

Figure 1: Project Workflow Flowchart

6 Data Structures

6.1 Abstract Syntax Tree (AST) – Tree-sitter

- **Purpose:** Extract meaningful components of the codebase (e.g., function definitions, class declarations).
- **Implementation:**
  - The CLI takes the input source code.
  - It is parsed using Tree-sitter, which generates an AST representation.
  - The AST allows extraction of only relevant nodes (functions and methods).
- **Benefit:** Helps in structuring and preprocessing the code before generating embeddings.

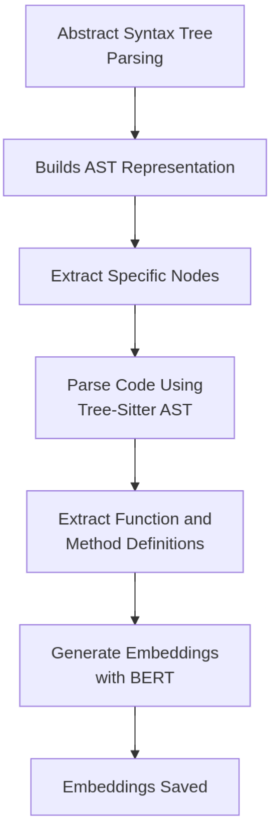

Figure 2: Abstract Syntax Tree Representation

6.2 Embeddings – KD-Tree

- **Purpose:** Efficiently store and search for semantically similar embeddings.
- **Implementation:**
  - Each extracted function is converted into an embedding using BERT-based models.
  - Embeddings are stored in a KD-Tree.
  - Query embeddings are compared against the stored KD-Tree to find nearest neighbors.
- **Time Complexity:**
  - Embedding Normalization: $O(n \times d)$
  - K-D Tree Construction: $O(n \log(n) \times d)$
  - K-Nearest Neighbor Query: $O(n \log(n) \times d)$

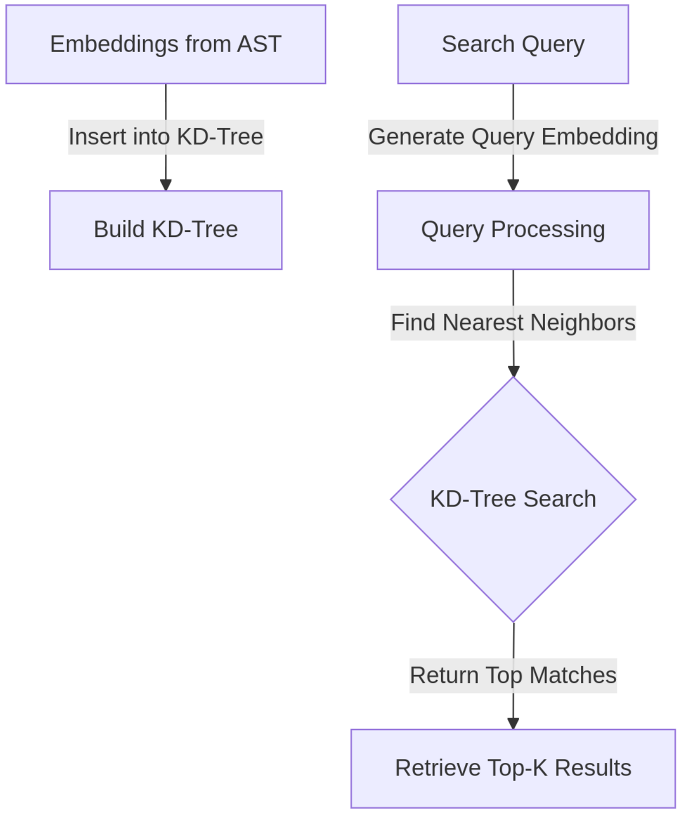

Figure 3: KD-Tree Embedding Structure

6.3 Min-Heap – Top-K Retrieval

- **Purpose:** Retrieve the top K most relevant results from the KD-Tree search.
- **Implementation:**
  - Use min-heap to store Top-K results after KD-Tree search.
  - Similarity scores (cosine similarity) determine function relevance.
- **Advantage:** Efficiently maintains top-K elements in $O(\log K)$ time.

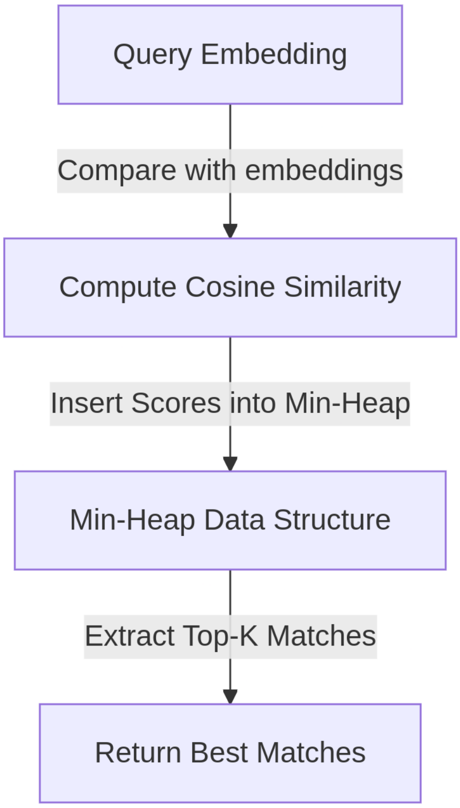

Figure 4: Min-Heap Top-K Retrieval

6.4 Clustering – Advanced Approach

- **Purpose:** Group semantically similar functions together.
- **Implementation:**
  - Check existing embeddings in storage.
  - Normalize embeddings for distance-based clustering.
  - Utilize KD-Tree for efficient nearest neighbor searches.
  - Apply Agglomerative Clustering to form meaningful groups.
- **Performance Improvement:** Reduced clustering complexity from $O(n^3)$ to $O(n^2)$.

6.5 Suffix Tree – (Inefficient Alternative)

- **Purpose:** Identify similar function names and patterns.
- **Drawbacks:**
  - Super slow for large datasets (¿ 10,000 points).
  - Extremely memory-intensive.
  - Total Time Complexity: $O(m^2 \times l^2)$
- **Recommendation:** Prefer KD-Trees for efficient searching.

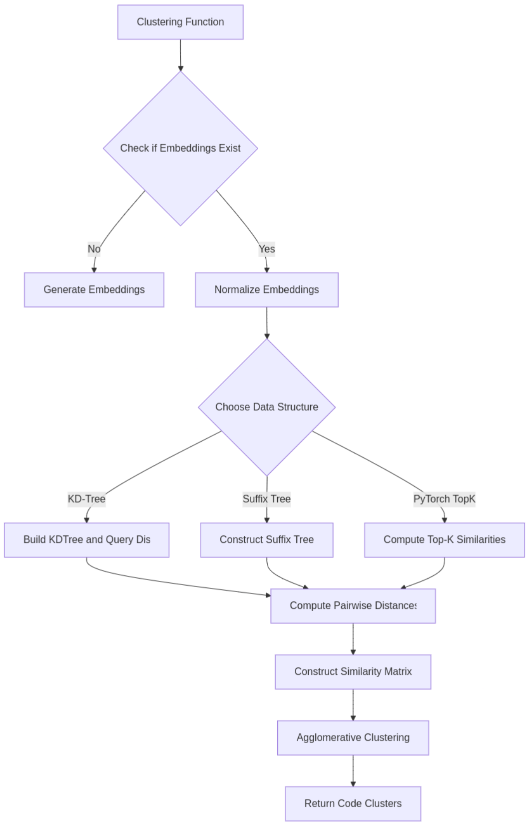

Figure 5: Clustering Process

7 References

1. Tree-sitter: `https://tree-sitter.github.io/`
2. Semantic Search using Sentence-BERT: `https://www.sbert.net/docs/quickstart.html`
3. Scipy KDTree Documentation: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.KDTree.html`

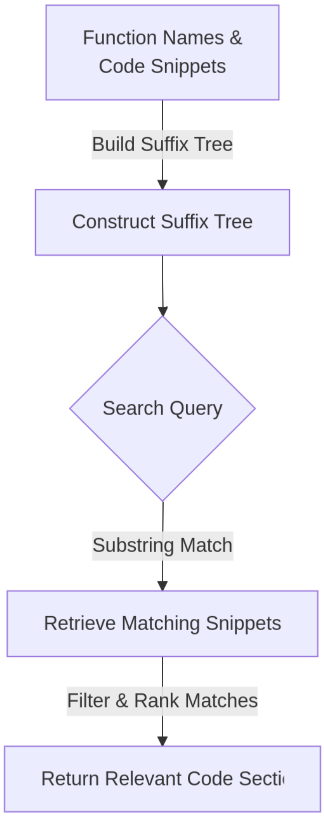

Figure 6: Suffix Tree Representation