MINOR PROJECT - 2

SYNOPSIS REPORT

Galava: A Scalable Search Engine for Organizational Database Pools using Large Language Models

Submitted By:

Name	Roll No	Branch
SAI ANKITA KASIBATLA	R2142210910	BTECH CSE AIML
SAAHIL SINGH RATHORE	R2142211304	BTECH CSE CCVT

School of Computer Science
UNIVERSITY OF PETROLEUM & ENERGY STUDIES
Dehradun-248007, Uttarakhand

Dr. Hanumat Sastry G
Professor
Data Science Cluster
Project Guide

School of Computer Science
University of Petroleum & Energy Studies

Project Title

Galava: A Scalable Search Engine for organizational database pools using Large Language Models.

Abstract

The advent of big data and the proliferation of diverse database technologies have significantly enhanced organizational capabilities to collect, store, and manage vast volumes of information across various domains. This expansion, however, has introduced complex challenges in data retrieval, notably the fragmentation of data across multiple databases, the steep learning curve associated with specialized query languages, and critical issues surrounding scalability and the time-consuming nature of obtaining results from multiple databases.

Firstly, the fragmentation of organizational data across disparate databases creates silos, obstructing a unified and comprehensive view of the data landscape. This fragmentation complicates data retrieval efforts, as users must navigate multiple systems with varying interfaces and query languages.

Secondly, the reliance on specialized query languages for different database systems restricts data access to users with specific technical expertise, creating a bottleneck in data-driven decision-making processes.

Lastly, and perhaps most critically, the scalability of traditional data retrieval methods is increasingly challenged by the growing size and number of databases. As organizations expand their data infrastructure, the time required to query and compile results from multiple databases escalates, impeding the agility and responsiveness of data analytics efforts.

These challenges highlight the urgent need for an innovative approach to data retrieval that not only integrates diverse data sources but also offers a scalable, efficient, and user-friendly querying mechanism. Such a solution would significantly enhance organizational agility by enabling swift, comprehensive, and intuitive access to data across multiple databases, thereby empowering a wider range of stakeholders to leverage data in decision-making and insight generation, regardless of their technical proficiency.

In response to these challenges, we introduce Galava, a search engine specifically designed for navigating through a web of databases, providing consistent, low latency and scalable interface. Galava is engineered to seamlessly integrate with a diverse family of database architectures, including SQL and NoSQL systems thereby enabling a unified search experience across the organizational data spectrum.

Introduction

In the era of information, data has become the lifeblood of organizations, driving decision-making, innovation, and strategic planning across various sectors. The ability to efficiently store, manage, and retrieve data is paramount to leveraging its full potential. Over the years, advancements in database technologies have led to the development of diverse systems tailored to specific types of data and use cases, ranging from traditional relational databases to more flexible NoSQL databases designed to handle unstructured data. While these technological advancements have significantly enhanced data storage and management capabilities, they have also introduced a new set of challenges in data retrieval, particularly in organizational environments characterized by a multitude of disparate database systems.

The proliferation of diverse databases within organizations has led to data silos, where information is stored in separate, often incompatible systems. This fragmentation hinders the ability to perform comprehensive searches across the entire data landscape, limiting the visibility and accessibility of valuable information. Moreover, the complexity of query languages specific to each database system necessitates a high degree of technical expertise, restricting data access to a select group of individuals within the organization. This barrier not only slows down the decision-making process but also diminishes their ability to fully capitalize on its data assets.

Furthermore, as organizations continue to grow and their data ecosystems expand, scalability becomes a critical concern. Traditional data retrieval methods struggle to keep pace with the increasing volume of data and the demand for real-time access, leading to delays and inefficiencies that can hamper operational agility and responsiveness.

Recognizing these challenges, the need for a unified, efficient, and user-friendly solution for navigating the complex web of organizational databases has become more apparent than ever. Such a solution would not only break down data silos and provide a holistic view of the data assets but also simplify the data retrieval process, making it accessible to a broader range of users regardless of their technical expertise.

In response to this need, we introduce Galava, a novel search engine designed to bridge the gap between diverse database systems and the users seeking to access their data. Galava stands out by offering a low-latency, scalable, and intuitive interface that allows for seamless navigation across a plethora of databases, effectively democratizing data access within organizations. By leveraging advanced natural language processing (NLP) techniques and knowledge graphs, Galava aims to transform the data retrieval landscape, making it more efficient, inclusive, and agile.

This project outlines the development of Galava, detailing its innovative approach to overcoming the challenges of data fragmentation, complexity, and scalability in organizational database environments. Through a comprehensive examination of its architecture, functionality, and potential impact, we aim to contribute to the ongoing discourse on data management and retrieval solutions, offering insights and perspectives that could shape the future of organizational data strategies.

Literature Review

Reference	Findings	Applications
[1](Bazaga et al., 2021)	Demonstrated the capability to predict ranked answers to natural language questions over SQL database tables by embedding both the question and the table into distributed representations.	Develop more intuitive and efficient natural language interfaces for SQL databases, particularly in domains where accuracy and the ability to interpret complex queries are critical, such as in business intelligence tools, customer support databases, and academic research databases.

[2](Dar et al., 2019)	Revealed a strong inclination towards English language support in these frameworks, and categorization of SQL-based frameworks into statistical, symbolic, and connectionist approaches, providing a comprehensive overview of the landscape of natural language database querying frameworks and their methodological diversities.	guide the development of next-generation database querying systems that are more inclusive of various database types and languages. It can also inform the expansion of natural language processing capabilities to accommodate linguistic variations, making database querying more accessible to non-technical users across different domains.
[3](Papenmeier et al., 2021)	The study found that chatbot-like interfaces could elicit significantly longer and more detailed natural language queries from users, with a greater number of key facts mentioned. This indicates the potential of conversational interfaces to enhance the richness of user queries, which could lead to more accurate and tailored search results.	designing user interfaces for search engines and databases that encourage more natural and detailed user queries. This could be particularly beneficial in e-commerce search engines, online libraries, and other digital platforms where users seek specific information but may not know the exact terms to use in their search queries.

Problem Statement

Organizations today face significant challenges in data retrieval due to the fragmentation of information across various database systems, the complexity of query languages that require specialized technical skills, and the scalability issues associated with handling large volumes of data across multiple databases. This situation creates barriers to efficient and timely access to data, limiting the ability of organizations to make informed decisions and leverage their full data potential. There is a critical need for a solution that simplifies and unifies data access, enabling users to retrieve information quickly and efficiently from diverse databases without requiring extensive technical expertise, and that can scale effectively to meet growing data demands.

Thus, the core problem this project aims to address is the development and introduction of a scalable search interface facilitated by language modeling to overcome the limitations in current approaches to data retrieval practices in distributed pools of databases. Leveraging natural language processing and intelligent query optimization, it promises to transform data retrieval practices, ensuring comprehensive, efficient, and user-friendly access to data assets, thereby enhancing decision-making processes and operational efficiency across multiple sectors.

Objectives

- To design a scalable search engine that seamlessly interfaces with distributed pools of databases, ensuring consistent performance and adaptability across varying data volumes and user demands
- Leverage fast and efficient HNSW (Hierarchical Navigable Small World) indexing algorithms to maintain an optimal balance between accuracy and speed, enabling rapid and precise data retrieval in large-scale database environments
- Employ a write-ahead caching mechanism to enhance retrieval speeds, allowing for the immediate availability of frequently accessed data and thereby improving the overall efficiency of the search process
- Integrate advanced Natural Language Processing (NLP) techniques to interpret and process user queries, facilitating an intuitive search experience that allows users to engage with the system using everyday language, thereby lowering the technical barrier to data access

Methodology

1. Vault Design:

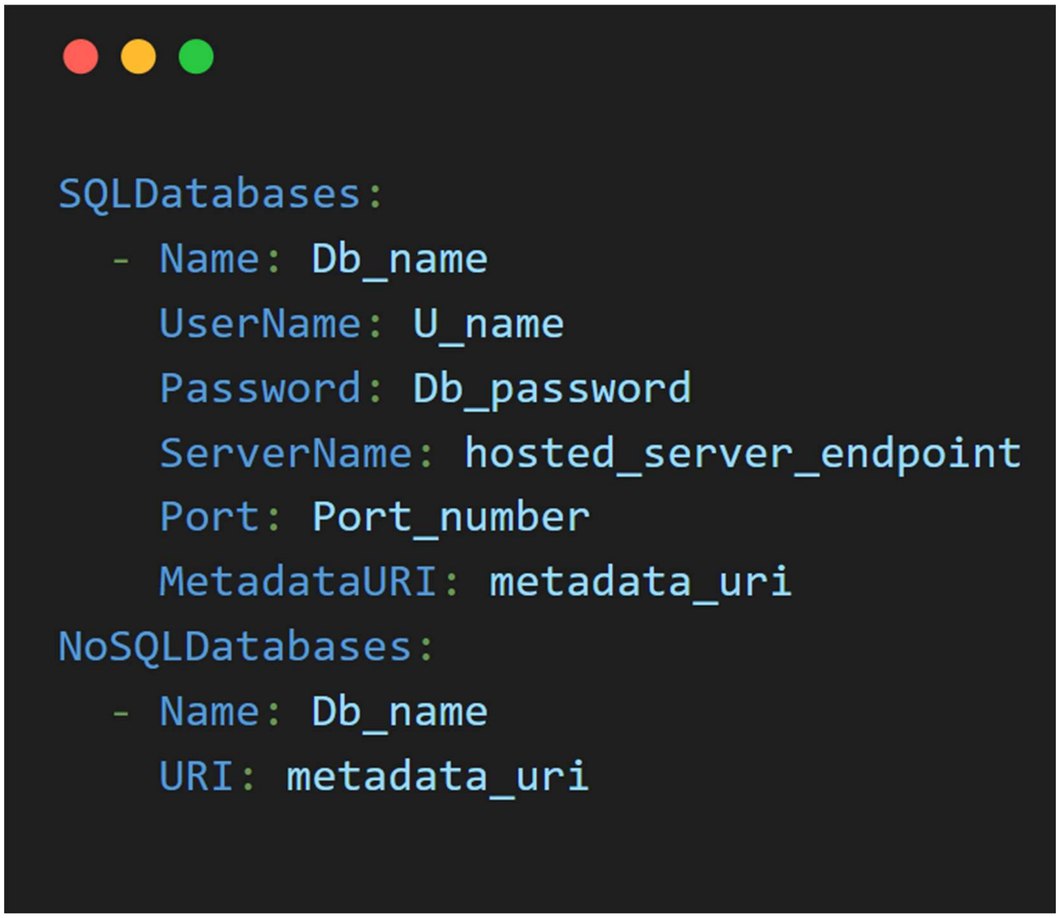
Properties Definition:

Secrets:

- The vault securely stores a secret key containing the URI to an object storage location. This object storage holds a crucial YAML configuration file that Galava utilizes to establish connections. Galava is granted read-only access to this object storage, ensuring it can securely retrieve the necessary configuration without compromising the integrity or security of the underlying databases.

Configuration YAML File:

- Galava relies on a configuration file, retrieved from the designated object storage bucket as specified by the vault's URI. This YAML file (Fig.1) is meticulously structured to include all necessary credentials, connection parameters, and configurations required for the application to interface with the various databases within the organizational ecosystem.



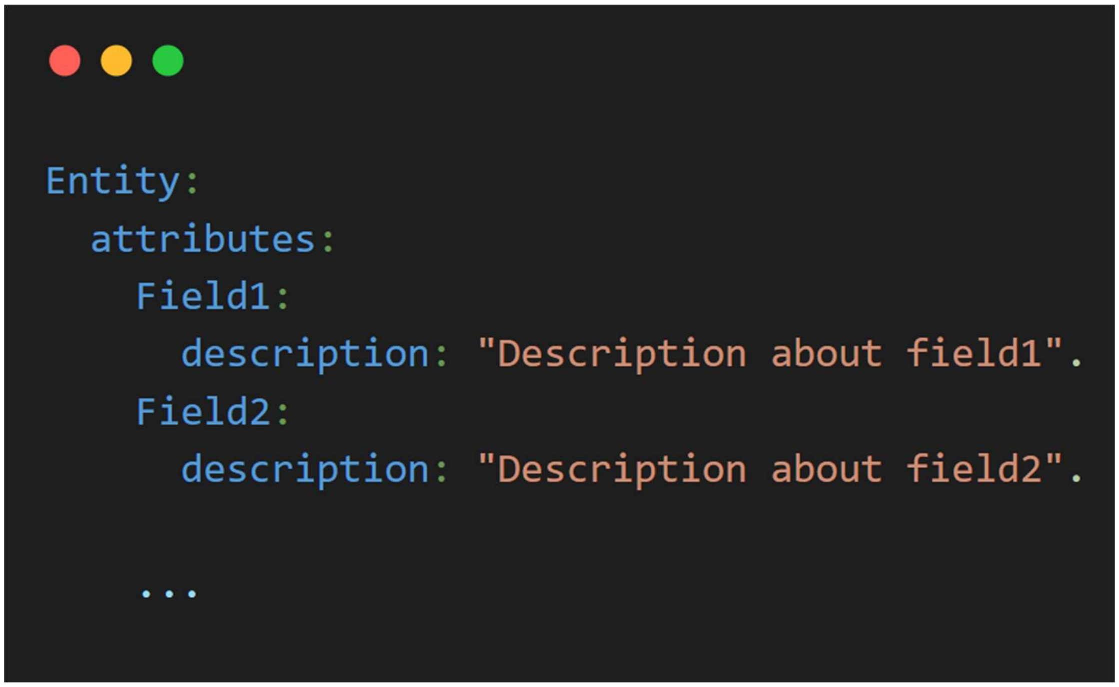
```yaml
SQLDatabases:
  - Name: Db_name
    UserName: U_name
    Password: Db_password
    ServerName: hosted_server_endpoint
    Port: Port_number
    MetadataURI: metadata_uri
NoSQLDatabases:
  - Name: Db_name
    URI: metadata_uri
```

Figure 1:Configuration file design

Behavior Definition

1. Metadata:

    - Each entity, whether a table in an SQL database or a document in a NoSQL database, is associated with metadata that encapsulates its characteristics, semantics, and additional relevant information(shown in fig.2). This metadata, crucial for understanding the structure and context of the data, is stored in an object storage location specified by the "Metadata URI" in the configuration file.



```
Entity:
  attributes:
    Field1:
      description: "Description about field1".
    Field2:
      description: "Description about field2".

    ...
```

Figure 2:Entity Metadata design

2. Knowledge Graph**:**

Structure:

- Each entity node in the knowledge graph will have an entity name.
- Entity nodes are grouped by a defined relationship with corresponding database nodes.
- These nodes will contain an attribute which stores its corresponding vector.

Fig.3 shows the high-level representation of knowledge graph.

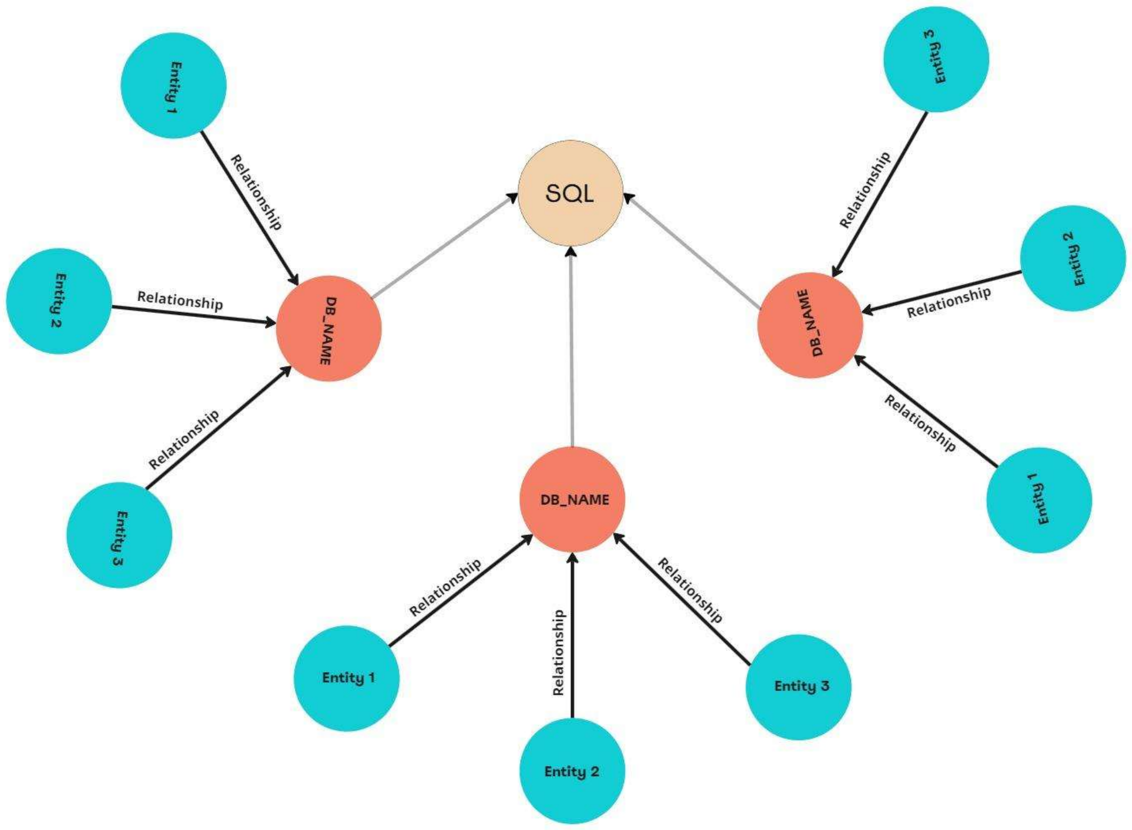


Figure 3:High-level representation of Knowledge Graph

3.Vectorization:

Embeddings:

- Each entity is pre-computed and stored as a vector embedding, which represents its semantic structure having a fixed dimension.

4. Approximate Nearest Neighbour Search:

Indexing Algorithm:

- The Hierarchical Navigable Small Worlds (HNSW) (fig.4) provides an efficient ANN search algorithm to identify relevant nodes for the querying vector.

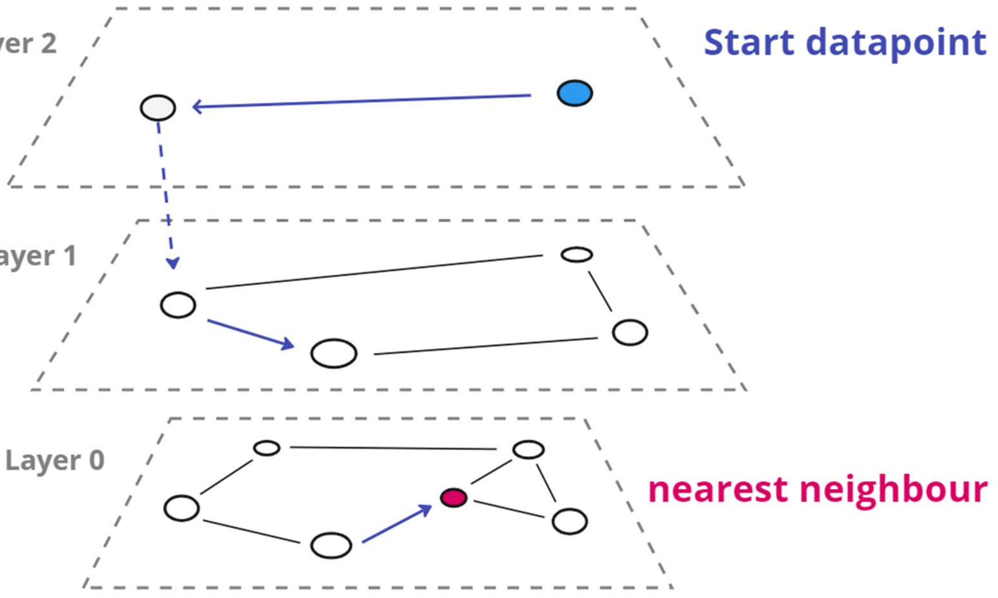


Figure 4:Hierarchical Navigable Small World (HNSW)

ALGORITHM:

Input: User query **q**, Vault URI **v_uri**, Configuration file **f1**, Metadata URI **m_uri**
Output: Query results for **q**

```
1. Retrieve the Vault URI v_uri to access the secure storage location
2. Load the configuration file f1 from the object storage specified by v_uri
3. for each database configuration in f1:
        a. Establish a connection to the database using its specific connection parameters
        b. Fetch the metadata from m_uri associated with the database
        c. Vectorize the fetched metadata to create a numerical representation for indexing
        d. Add a node for each vectorized metadata entity to the Knowledge Graph
4. Vectorize the user query q to transform it into a numerical vector q_vec
5. Utilize the HNSW (Hierarchical Navigable Small World) algorithm to identify the nearest nodes
in the Knowledge Graph to the vectorized query q_vec
6. Add the identified nearest nodes to a processing queue Q
7. for each node in Q:
        a. Extract the table details and relevant query information
        b. Formulate an equivalent database query using an LLM
        c. Execute the formulated query against the respective database
        d. Collect the query results
8. Aggregate the collected query results from all processed nodes
```

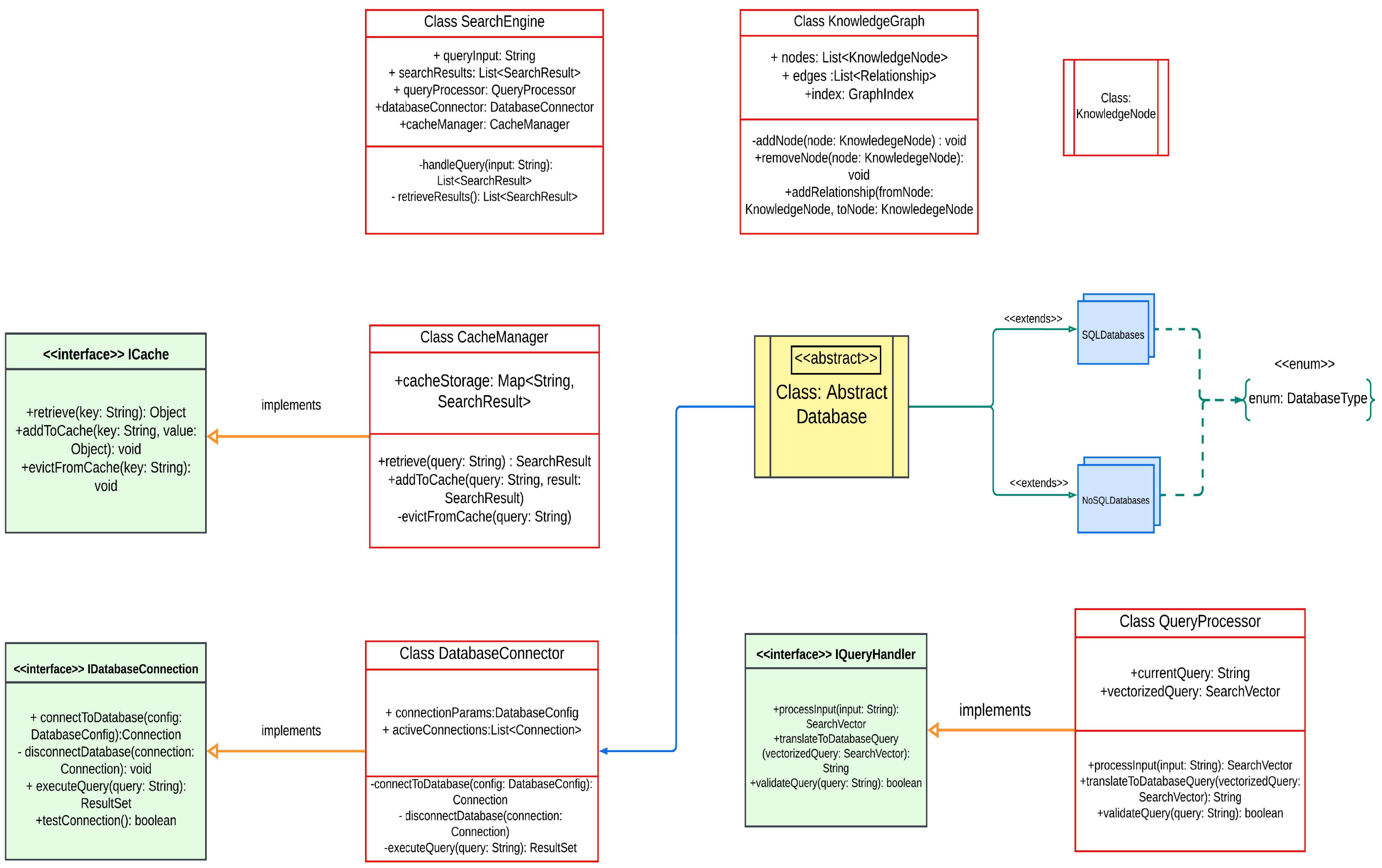

Class SearchEngine
+ queryInput: String
+ searchResults: List<SearchResult>
+ queryProcessor: QueryProcessor
+databaseConnector: DatabaseConnector
+cacheManager: CacheManager
-handleQuery(input: String): List<SearchResult>
- retrieveResults(): List<SearchResult>
Class KnowledgeGraph
+ nodes: List<KnowledgeNode>
+ edges :List<Relationship>
+index: GraphIndex
-addNode(node: KnowledegeNode) : void
+removeNode(node: KnowledegeNode): void
+addRelationship(fromNode: KnowledgeNode, toNode: KnowledegeNode
Class: KnowledgeNode
<<interface>> ICache
+retrieve(key: String): Object
+addToCache(key: String, value: Object): void
+evictFromCache(key: String): void
implements
Class CacheManager
+cacheStorage: Map<String, SearchResult>
+retrieve(query: String) : SearchResult
+addToCache(query: String, result: SearchResult)
-evictFromCache(query: String)
<<abstract>>
Class: Abstract Database
<<extends>>
SQLDatabases
<<extends>>
NoSQLDatabases
<<enum>>
enum: DatabaseType
<<interface>> IDatabaseConnection
+ connectToDatabase(config: DatabaseConfig):Connection
- disconnectDatabase(connection: Connection): void
+ executeQuery(query: String): ResultSet
+testConnection(): boolean
implements
Class DatabaseConnector
+ connectionParams:DatabaseConfig
+ activeConnections:List<Connection>
-connectToDatabase(config: DatabaseConfig): Connection
- disconnectDatabase(connection: Connection)
-executeQuery(query: String): ResultSet
<<interface>> IQueryHandler
+processInput(input: String): SearchVector
+translateToDatabaseQuery (vectorizedQuery: SearchVector): String
+validateQuery(query: String): boolean
implements
Class QueryProcessor
+currentQuery: String
+vectorizedQuery: SearchVector
+processInput(input: String): SearchVector
+translateToDatabaseQuery(vectorizedQuery: SearchVector): String
+validateQuery(query: String): boolean

Figure 5: UML Class Diagram

Diagram key

🟠 Concrete Class

🟡 Abstract Class

🟢 Interface

{ } enum

PERT Chart

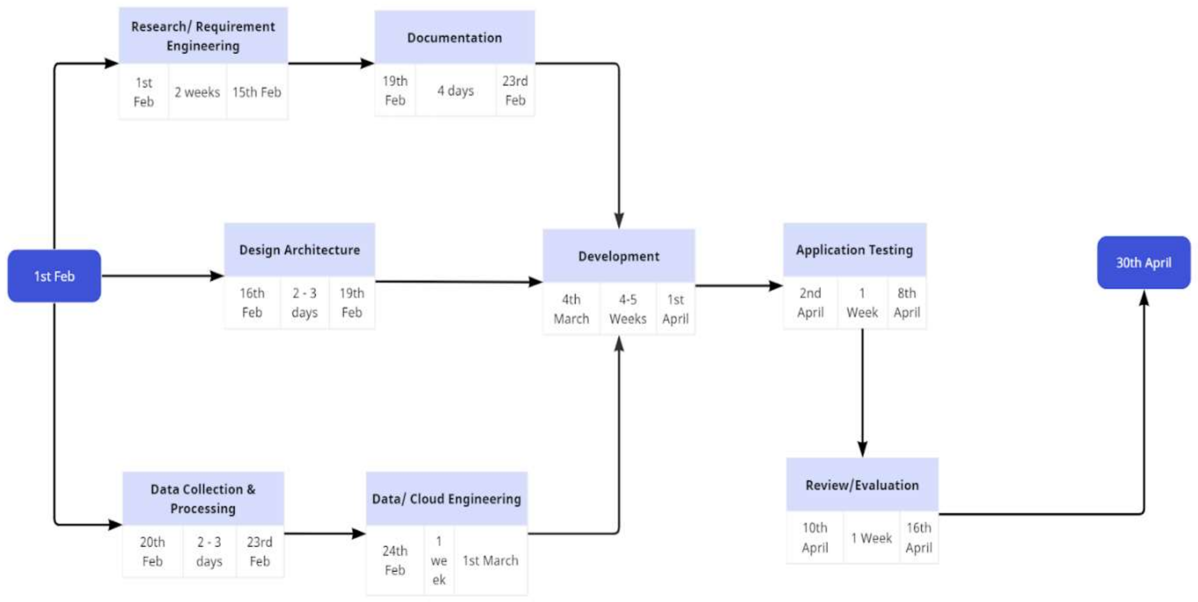


Figure 6:PERT chart

Tech Stack

- Version Control: GitHub
- Development: Flask, Neo4j, Llama2/GPT4, Scapy, AWS RDS, AWS Secret Manager, AWS S3, Streamlit, Langchain, Pinecone SDK

References

1. Bazaga, A., Gunwant, N., & Micklem, G. (2021). Translating synthetic natural language to database queries with a polyglot deep learning framework. *Scientific Reports*, (Bazaga et al., 2021)

2. Dar, H. S., Lali, M. I., Din, M. U., Malik, K. M., & Bukhari, S. A. (2019). Frameworks for Querying Databases Using Natural Language: A Literature Review. (Dar et al., 2019)

3. Papenmeier, A., Kern, D., Hienert, D., Sliwa, A., Aker, A., & Fuhr, N. (2023). Starting Conversations with Search Engines -- Interfaces that Elicit Natural Language Queries. (Papenmeier et al., 2021)