RAG CAN Application

This project demonstrates a Retrieval-Augmented Generation (RAG) pipeline that integrates text encoding, vector storage, and query processing. It is built on Python 3.12 and uses Qdrant to store and search vector representations of species data. The application supports both keyword-based (sparse) and semantic (dense) searches.

Important Usage Notice

This application is designed to work only with queries related to scientific names or common names of species. If users ask about unrelated topics, the search process will not function correctly, as the vector store is optimized exclusively for species-related information.

Table of Contents

Overview

This application processes queries by:

- Extracting keywords from user input.
- Encoding these keywords into sparse vector representations.
- Performing a semantic search with dense vector representations.
- Displaying results that include scientific names and source URLs.

Requirements

The application is built for Python 3.12. Ensure you have the following packages installed as specified in the `requirements.txt`:

```
psycopg2==2.9.10
sqlalchemy==2.0.40
qdrant-client==1.13.3
keybert==0.9.0
transformers==4.50.3
sentence-transformers==4.0.1
einops==0.8.1
```

```
tiktoken==0.9.0
python-dotenv==1.1.0
```

Installation

1. **Set Up Python Environment:**

   Create a virtual environment using Python 3.12:

   ```
   python3.12 -m venv .venv
   source .venv/bin/activate  # On Windows, use `.venv\Scripts\activate`
   ```

   Once the virtual environment is activated, proceed to install the dependencies.

2. **Install Dependencies:**

   ```
   pip install -r requirements.txt
   ```

Configuration

Before running the application, you must configure the database connections. A .env file should be created based on the provided .env.template file:

```
POSTGRES_USERNAME=<your_postgres_username>
POSTGRES_PASSWORD=<your_postgres_password>
POSTGRES_HOST=<your_postgres_host>
POSTGRES_PORT=<your_postgres_port>
POSTGRES_DB=<your_postgres_database>
MONGO_USERNAME=<your_mongo_username>
MONGO_PASSWORD=<your_mongo_password>
MONGO_HOST=<your_mongo_host>
MONGO_PORT=<your_mongo_port>
MONGO_DB=<your_mongo_database>
```

Ensure these values are set correctly, as the application fetches species data from PostgreSQL and MongoDB.

Usage

Creating the Vector Store

Before you run the main query processor, the vector store must be created with species data. The script `create_vector_store.py` fetches data from PostgreSQL and MongoDB, processes species content, and creates both sparse and dense embeddings.

To create the vector store:

```
python create_vector_store.py
```

If you wish to force the recreation of the vector store (for example, if there have been changes to your data), add the `--force` flag:

```
python create_vector_store.py --force
```

Executing the Main Application

After the vector store is ready, run the main application with your query. The `main.py` script checks for the existence of the collection, processes the query, extracts keywords, and performs both keyword-based and semantic searches.

Example Command:

```
python main.py -q "What are the characteristics and scientific classification of Aphis gossypii in relation to crop plagues?"
```

Query Examples

Here are several example queries that incorporate both plague-related topics and the specific scientific names:

1. **Plant Species in Agricultural Context:**

   ```
   python main.py -q "Provide insights on Sorghum halepense and Azolla pinnata about their influence on plant diseases and plague conditions."
   ```

2. **Aquatic Plant and Insect Vector:**

   ```
   python main.py -q "How does Azolla pinnata interact with local ecosystems during periods of disease spread?"
   ```

3. **Insect Vector Analysis:**

   ```
   python main.py -q "Discuss the biology of Bactericera cockerelli and its potential involvement in spreading plant pathogens."
   ```

Each of these commands will:

- First perform a keyword-based search using sparse vector representations.
- Then execute a semantic search using dense vector representations filtered by valid source URLs.
- Print out results that include the scientific names and associated content.

Project Structure

- **`main.py`**: Main application script that processes user queries.
- **`create_vector_store.py`**: Script to initialize and populate the vector store.
- **`services/`**: Contains modules for `VectorStore`, `QueryProcessor`, and `TextEncoder`.
- **`repositories/`**: Contains data access logic (e.g., `CabiSpeciesRepository`).
- **`models/`**: Contains the `VectorizableDocument` definition.
- **`data/content_chunks/`**: Directory where dense embedding content chunks are stored.

Additional Notes

- **Python Version:** This application is designed for Python 3.12.
- **Data Sources:** The species data is fetched from PostgreSQL and MongoDB. Ensure your database connections are properly configured.
- **Model Configuration:** The application uses the following models:
  - Sparse model: `"naver/splade-v3"`
  - Dense model: `"jinaai/jina-embeddings-v3"`
- **Vector Collection:** The collection name is `"cabi_species_vectors"`, with separate names for sparse and dense embeddings.
- **Error Handling:** If the vector store collection does not exist, the application will instruct you to run `create_vector_store.py`.