Web Scraping, Database Storage, and Full-Text Search for Manufacturing Articles

Introduction

This project aims to scrape manufacturing-related articles from The Economic Times, store them in a SQLite database, and implement full-text search (FTS) and embedding-based search functionalities. The main objectives are:

1. **Web Scraping**: Extract article titles, links, and content from the webpage.
2. **Database Management**: Store the articles in a structured SQLite database.
3. **Full-Text Search**: Enable efficient retrieval of articles using SQLite's FTS5 functionality.
4. **Embedding-Based Search**: Use Sentence Transformers to generate embeddings for articles and retrieve similar articles based on query embeddings.
5. **Language Model Response Generation**: Use OpenAI's GPT-4 for summarizing and responding to user queries in a chatbot interface built using Streamlit.

Steps Followed

1.Web Scraping Using BeautifulSoup

1.1 **Getting the Webpage**: It uses the `requests` library to download the webpage's content.

1.2 **Reading the HTM**L: The `BeautifulSoup` library processes the downloaded webpage, allowing easy access to different parts of the HTML.

1.3 **Finding Article Titles and Links:** The code searches the webpage for article titles and their links, then saves them in a list for later use.

2.Database Management with SQLite

2.1 **Creating Tables**: Two tables are set up—one called "articles" to store the basic information and another called "articles_fts" for full-text searching.

2.2 **Inserting Data**: The scraped article details are then added into the database.

3. Fetching and Storing Article Content

`get_article_content` function retrieves the full text of each article by visiting its URL. It then stores the article's content in the SQLite database.

4.Full-Text Search with FTS5

4.1 **Creating a Virtual FTS5 Table**: A special table, called `articles_fts`, is created to make full-text searches faster and more efficient.

4.2 Searching Articles: Users can search for articles using keywords, and the code will return matching article titles along with brief excerpts from their content.

5 Embedding-Based Search Using Sentence-BERT

The project uses an embeddings-based search to find articles that are related to the user's query in a meaningful way. It uses Sentence-BERT to create embeddings (numeric representations) for both the articles and the user's query, and then compares them using cosine similarity to find the best matches.

1. **Creating Embeddings**: Sentence-BERT generates vector representations of the article content.
2. **Storing Embeddings**: These embeddings are saved in the SQLite database.
3. **Similarity Search**: The system compares the query's embeddings with the stored ones and retrieves the most relevant articles based on similarity scores.

6 Integration with GPT and Streamlit Interface

The project features a chatbot interface created with Streamlit, allowing users to ask questions about manufacturing and supply chains. The chatbot searches for relevant articles using either full-text search (FTS5) or embeddings, then uses GPT-4 to provide a detailed answer.

1. **RAG (Retrieval-Augmented Generation) Pipeline**: The chatbot first finds relevant articles in the database, then combines that information into a prompt for GPT-4, which generates a well-informed response.

Key Features

- **Web Scraping**: Extracts titles, links, and content from The Economic Times.
- **SQLite Storage**: Efficiently stores and retrieves articles using both traditional and FTS5 tables.
- **Full-Text Search**: Enables keyword-based retrieval of articles.
- **Embedding-Based Search**: Uses semantic search to find articles similar to a user's query.
- **Chatbot Interface**: Powered by GPT and integrated with Streamlit for interactive queries.

Libraries used

`requests`: Fetches web pages from the internet.

`BeautifulSoup`: Parses and extracts data from HTML content.

`sqlite3`: Manages the SQLite database for storing and retrieving article data.

`SentenceTransformer`: Converts article content and user queries into vector embeddings for semantic search.

`numpy`: Handles numerical operations, especially with embeddings.

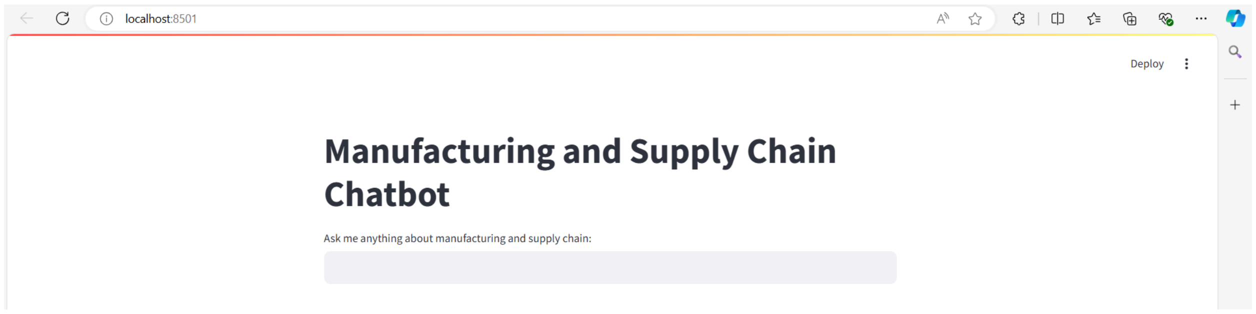
`pickle`: Serializes and deserializes data (like embeddings) for storage in the database.

`scipy.spatial.distance`: Calculates cosine similarity to compare embeddings.

`openai`: Accesses GPT to generate detailed responses based on article data.

`streamlit`: Creates a web interface for the chatbot, allowing user interaction.

Result



localhost:8501

Deploy

Manufacturing and Supply Chain Chatbot

Ask me anything about manufacturing and supply chain: