Kanpur Restaurant Information Assistant

Technical Documentation

Adarsh Pal

April 20, 2025

Contents

1 System Architecture

The system is designed using a Retrieval-Augmented Generation (RAG) pipeline to assist users in retrieving relevant restaurant information based on natural language queries. The architecture consists of four main modules:

- **Data Loader**: This module loads and preprocesses restaurant data from a CSV file. It extracts important information such as restaurant names, ratings, price ranges, cuisines, and locations, which are then used for semantic search.
- **Vector Database**: The vector database is built using the Sentence-Transformers model to generate embeddings for the restaurant data. These embeddings are used for efficient semantic search.
- **RAG Pipeline**: This is the core of the system, where the semantic search results are retrieved from the vector database and used as context for generating responses using an LLM.
- **Frontend Interface (Streamlit)**: A user-friendly chat interface is built with Streamlit that allows users to interact with the system and get restaurant recommendations or details based on their queries.

2 Implementation Details

The system is implemented in Python, using the following technologies:

- **Sentence-Transformers**: Used for generating embeddings for text-based data, allowing semantic search based on natural language queries.
- **Pandas**: Used for loading and preprocessing the restaurant data.
- **Streamlit**: Provides an interactive web-based chat interface.
- **Groq**: Used for simulating a large language model (LLM) API call to generate responses based on the context retrieved from the vector database.

The flow of the system can be broken down as follows:

1. **Data Loading**: The data is read from a CSV file, and preprocessing is done to extract location and create a search text field.
2. **Embedding Generation**: Restaurant names, cuisines, locations, ratings, and price details are encoded using the Sentence-Transformers model to create embeddings.
3. **Search and Retrieval**: A query entered by the user is encoded, and semantic search is performed by computing the cosine similarity between the query and restaurant embeddings.
4. **Response Generation**: Relevant results from the vector database are formatted and passed to an LLM for response generation.
5. **Frontend**: The user interacts with the system via a chat interface, asking questions about restaurants, and the system responds with relevant data.

3 Design Decisions

The key design decisions include:

- **Embedding Model**: The choice of using the `all-MiniLM-L6-v2` embedding model was made due to its balance of efficiency and effectiveness in generating embeddings for text-based search tasks.

- **Vector Database**: The use of cosine similarity between query embeddings and restaurant embeddings allows for efficient and scalable retrieval of relevant information.
- **RAG Pipeline**: By combining semantic search with LLM-based generation, we ensure that the response is contextually relevant and based on the available data, avoiding hallucination.
- **Frontend**: The choice of Streamlit as the frontend was based on its simplicity, ease of use, and rapid prototyping capabilities, making it ideal for a chatbot interface.

4 Challenges Faced and Solutions Implemented

During the development of the system, several challenges were encountered, and solutions were implemented as follows:

- **Search Efficiency**: Initial testing showed that the system could be slow when searching across large datasets. To address this, we optimized the search process by using vector embeddings and cosine similarity, which significantly reduced query times.
- **Handling Out-of-Scope Queries**: Users often asked for information such as menus or contact details, which the system could not provide. We added specific instructions in the system prompt to ensure that the assistant does not respond to such out-of-scope queries.
- **LLM Integration**: Integrating an LLM API to generate natural language responses posed challenges due to API limitations. We overcame this by mocking the LLM calls initially and using Groq's API for simulation.

5 Future Improvement Opportunities

Several opportunities for improving the system have been identified:

- **Enhanced Query Understanding**: The system could be enhanced with advanced NLP techniques to better understand complex or ambiguous queries, including query disambiguation and multi-turn conversations.
- **Support for Multi-language Queries**: Since the system is focused on Kanpur, India, supporting multiple languages (e.g., Hindi) would improve accessibility for a wider audience.
- **Integration with Live Data Sources**: Currently, the restaurant data is static. Integrating the system with live data sources such as Zomato or Google Maps would allow for real-time updates on restaurant details and availability.
- **Improved LLM Response Generation**: The quality of responses could be further enhanced by using more advanced models, such as GPT-3 or GPT-4, to provide more human-like responses.
- **User Feedback Loop**: Implementing a feedback mechanism where users can rate the relevance of responses would help fine-tune the model and improve the system's accuracy over time.