Movie Chatbot Development Process

Git Link: https://github.com/maheshsai252/movies-rag

Demo:

1. Domain Selection

Application: Movie Chatbot

Scope and Data Type:

- The application will handle data related to movies, including movie overview, genres, release dates, and ratings.
- The chatbot will respond to user queries about movies, provide recommendations, and offer information about movie details.

2. Data Collection and Preprocessing

Data Source: Kaggle dataset on movies, Prompt Engineering to get mock data

Steps:

1. **Acquire Data:** Download the movie dataset from Kaggle.

2. **Data Cleaning:**
    - Remove any irrelevant columns or information not needed for the chatbot.
    - Handle missing data by filling in defaults or removing incomplete entries.
    - Generate missing plots of movies using LLM.
3. **Text Preparation:**
    - Remove special characters and unnecessary whitespace.
    - Address missing by prompt - engineering.

3. Vector Database Implementation

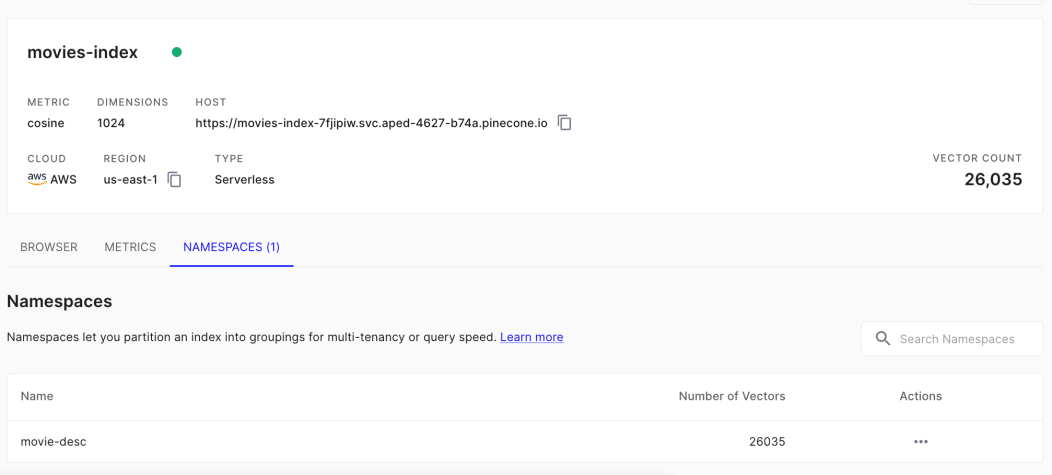
Chosen Database: Pinecone

Steps:

1. **Setup Pinecone:**
    - Create an account and set up an index in Pinecone.
2. **Data Indexing:**
    - Use sentence transformer to Transform movie descriptions and relevant textual data into embedded vectors.
    - Store these vectors in Pinecone, ensuring efficient indexing for quick retrieval.
3. **Configuration:**
    - Configure the index to support semantic similarity searches, which will allow the chatbot to retrieve relevant movie data based on user queries.

4. Upload Data to Vector DB

Upload cleaned data into pinecone vector db. We uploaded around 26k movies



5. Application Development

Frameworks Used: FastAPI for backend, Streamlit for frontend

Steps:

1. **User Interface:**
    - Designed a simple, user-friendly interface in Streamlit where users can input natural language queries.
2. **Backend Logic:**
    - Implemented a FastAPI backend to handle incoming queries
    - Based on chat history, refine latest user question to standalone question based on chat history. Use a GPT-3.5 instruct to understand user intent and extract relevant information.
    - Use the Pinecone vector database to retrieve the most relevant movie data based on the processed query
    - Embed the retrieved context and refined query into prompt in langchain and send the content to user.
3. **Orchestration**: orchestrated frontend and backend using docker-compose.

5. Evaluation and Testing

- Conduct extensive testing with a variety of queries to ensure the chatbot understands and responds accurately.
- Include edge cases and unusual queries to evaluate robustness.

Testing whether bot can chat without losing context:

Queries: Recommend Animated Movies, Should have Sci-FI as well, Add drama

Hallucination test:

Queries: what is plot of movie ‘ksk’, changed release year of titanic in vector db and tested asking year

Complex queries:

Queries: any movies you know involving gang dramas story origin from india

I am going to India for holidays, i want to watch movies on the way that shows the country heritage

I am going to Florida for vacation, show me some movies involving catastrophic situations that can happen there

Confusing Queries:

any bollywood movies where good wins over bad
any bollywood movies where good wins over bad

Refer to Video for results from our chatbot

Steps to Run Application :

- CD to Web-Service
- Place your pinecone_api_key and openai_key.
- “docker-compose build” build requirements
- “docker-compose up” to start the app.

Challenges Faced

Data Quality:

Challenge: The Kaggle dataset contained incomplete or inconsistent entries.

Solution: Implemented robust data cleaning procedures, including filling missing values and removing irrelevant information.

Vector Database Configuration:

Challenge: Ensuring efficient and accurate indexing in Pinecone.

Solution: Experimented with different indexing parameters and configurations to optimize performance for semantic similarity searches.

Query Understanding:

Challenge: Ensuring the LLM accurately understands and processes diverse natural language queries.

Solution: Tuned prompt of LLM using a variety of techniques learnt in class to improve understanding and accuracy.

Performance Optimization:

Challenge: Maintaining fast response times while handling complex queries.

Solution: Tested latency against different retreival mechanisms cosine, dot product and euclidian for latency