Gen-AI Powered Interactive QA Bot with Docker Integration

1. INTRODUCTION

In this project, we developed a Gen-AI Powered Interactive QA Bot utilizing Retrieval-Augmented Generation (RAG) to provide more accurate and contextually relevant answers to user queries. This solution integrates Cohere for natural language processing (NLP) and Pinecone for vector-based search and retrieval, ensuring a seamless question-answering system. The system is containerized using Docker to ensure ease of deployment and scalability.

2. PROBLEM STATEMENT

The main goal of the project was to develop an interactive QA bot that:

- Provides accurate and context-aware responses to user queries.
- Can efficiently retrieve relevant documents or data using RAG techniques.
- Leverages advanced NLP models (Cohere) for generating high-quality responses.
- Utilizes Pinecone for optimized vector search.
- Supports a seamless setup using Docker for modular deployment.

3. APPROACH AND TECHNOLOGIES USED

3.1 Approach

Our approach consisted of developing two main components:

- **Backend (RAG Model):**
    - Manages the retrieval-augmented generation process by integrating Cohere for text generation and Pinecone for vector search.
    - Stores generated answers in a data folder for analysis and further processing.
    - Utilizes API endpoints to serve the frontend and other external applications.
- **Frontend (Interactive QA Bot):**
    - Provides an interactive interface for users to submit queries and receive real-time answers from the RAG-powered backend.
- **Core Features:**

- **Retrieval-Augmented Generation:** The system combines retrieval with generative AI to produce answers that are both contextually relevant and factually accurate.
- **Cohere Integration:** Uses Cohere's large language models for generating natural and human-like responses.
- **Pinecone Integration:** Provides efficient vector storage and retrieval for document search.

❖ **Data Storage:**

- Generated answers are stored in a data folder for further analysis.

3.2 Technologies

- **Cohere API:** For text generation and natural language processing.
- **Pinecone API:** For vector search and retrieval, enabling the RAG model to retrieve relevant data.
- **Python:** Used for the backend RAG model and API implementation.
- **Node.js:** Backend for the frontend to handle API communication.
- **React:** Used to build the interactive UI for the QA bot.
- **Docker:** For containerization of both frontend and backend applications.
- **Docker Compose:** To manage multi-container applications, ensuring smooth deployment of the backend and frontend services.

4. IMPLEMENTATION DETAILS

4.1 Backend Implementation

The backend is built using Python and structured into multiple modules:

- **RAG Model:** The core of the backend, handling the combination of document retrieval and text generation. Cohere is used for generating answers, while Pinecone is used for vector searches.
- **API Endpoints:** Built using Flask, allowing the frontend to interact with the backend.
- **Data Storage:** Generated answers are saved in text files in the data folder, with a timestamp for tracking.

❖ **Key files:**

- **rag_model.py:** Implements the RAG model logic.
- **pinecone_db.py:** Manages Pinecone vector storage and retrieval.

- **routes.py:** Defines the API routes.

4.2 Frontend Implementation

The frontend is built using React and provides a user-friendly interface for querying the QA system. The key features include:

- **User Input:** A simple form where users can submit their questions.
- **Real-time Response:** The system displays answers in real-time as generated by the backend.
- **UI/UX:** Clean and responsive design for a seamless user experience.

❖ **Key file:**

- **app.py:** The main Flask application for the frontend.

4.3 Docker Integration

Both the backend and frontend are containerized using Docker to allow for easy deployment and scaling. A docker-compose.yml file is used to define services, making it simple to spin up the entire system with a single command.

❖ **Key files:**

- **Dockerfile (Backend & Frontend):** Instructions for building Docker images.
- **docker-compose.yml:** Defines and orchestrates the multi-container application.

4.4 Data Pipeline

The data pipeline is structured as follows:

- **User Query:** The user submits a question via the frontend.
- **Pinecone Search:** The backend uses Pinecone to retrieve relevant documents or data vectors.
- **Cohere Generation:** The RAG model combines the retrieved data with Cohere's generative capabilities to create a coherent and context-aware response.
- **Answer Storage:** The response is stored in the data folder with a timestamp.
- **Frontend Display:** The answer is returned to the user and displayed in the frontend.

5. CHALLENGES AND SOLUTIONS

5.1 Challenges

❖ **Integrating Cohere and Pinecone:**

- Challenge: Combining document retrieval with text generation for accurate responses.
- Solution: Developed a pipeline where retrieved documents from Pinecone were fed into the Cohere API to ensure relevant and precise answers.

❖ **Dockerization:**

- Challenge: Ensuring both frontend and backend run smoothly in separate containers with proper communication between them.
- Solution: Utilized Docker Compose to orchestrate multi-container deployments and tested cross-container API communication.

❖ **Answer Storage:**

- Challenge: Managing and storing generated answers for analysis and ensuring efficient retrieval.
- Solution: Implemented a simple file storage mechanism where answers are saved as .txt files in a timestamped format.

5.2 Solutions

- Leveraged Pinecone's vector database for efficient document retrieval.
- Used Cohere's NLP models to generate human-like responses.
- Designed a scalable Docker architecture to simplify deployment across different environments.

6. RESULTS

The final implementation successfully:

- Generates context-aware, accurate answers using the RAG model.
- Stores the generated answers for further analysis.
- Is deployable on any environment using Docker.
- Handles user queries efficiently, retrieving and generating responses in real time.

Interactive QA Bot with RAG

Upload a PDF document

Drag and drop file here
Limit 200MB per file • PDF
Browse files

Maneuver_Detection_Report.pdf 5.0KB

Document uploaded successfully

Ask a question about the document:

list methodologies.

Answer: The methodologies used in the given context are as follows:

1. Data Loading: The first step involves loading the data from a CSV file into a pandas DataFrame. This is a common method for importing and handling structured data in Python, and it provides a flexible and efficient way to work with tabular data.
2. Data Cleaning: This step focuses on data preprocessing and cleaning. It includes removing missing values, handling outliers, and ensuring the data is in a suitable format for further analysis. Data cleaning is crucial to ensure the accuracy and reliability of the subsequent steps.
3. Maneuver Detection: The core methodology used for maneuver detection is adaptive thresholding with Simple Moving Average (SMA). This technique involves applying a sliding window to the data and using a threshold to identify significant changes in the spacecraft's orbital parameters, indicating a maneuver. The adaptive thresholding approach allows for dynamic adjustment of the threshold based on the data characteristics.
4. Evaluation: The detected maneuvers are evaluated using various performance metrics, including accuracy, precision, recall, F1 score, and ROC AUC. These metrics are computed by comparing the detected maneuvers with a reference dataset, providing a quantitative assessment of the algorithm's performance.
5. Visualization: Visualization is employed to graphically represent the detected maneuvers. This step helps in understanding the timing and characteristics of the maneuvers, making it easier to interpret the results and communicate findings.
6. Report Generation: The project generates reports in two formats: JSX (JavaScript XML) and text. The JSX report includes a React component that displays the metrics and key information about the maneuver detection process. The text report provides a summary of the project, including the dataset used, graph path, metrics, and the detection algorithm employed.

These methodologies collectively contribute to a comprehensive approach for spacecraft maneuver detection, evaluation, and reporting. The project demonstrates a structured and systematic process for processing and analyzing spacecraft data, with a focus on accuracy, visualization, and communication of results.

7. CONCLUSION

This project successfully demonstrates how combining retrieval-augmented generation (RAG) with powerful technologies like Cohere and Pinecone can create a sophisticated interactive question-answering system. The use of Docker ensures modularity and ease of deployment, making it suitable for scalable production environments.

8. FUTURE WORK

Potential future enhancements:

- **Improving the Model:** Experiment with other NLP models or fine-tune Cohere's models for specific domains.
- **Advanced Caching:** Implement caching mechanisms for frequently asked questions to improve response times.
- **Real-time Deployment:** Deploy the system on a cloud service like AWS or GCP for real-time usage.
- **UI/UX Improvements:** Enhance the frontend with more interactive features like chat interfaces.