RAG Based Chatbot to Boost Efficiency

Introduction:

The project involved developing a financial chatbot based on the Retrieval-Augmented Generation (RAG) system. The primary aim of the chatbot is to serve as a centralized knowledge base for employees, enabling them to quickly and effectively find answers to document-related questions. These documents include company policies, frequently asked questions, rules, handbooks, guides, technical walkthroughs, and more. By leveraging the RAG system, the chatbot integrates document retrieval with AI-based answer generation, ensuring accurate and contextually relevant responses.

This solution was designed to improve employee productivity by streamlining access to information, thus enabling them to cater to customers more efficiently. The chatbot ensures that the knowledge base is always up to date by syncing the latest document changes, thereby guaranteeing accurate responses to queries. The AI model used in the system detects relevant documents based on the query and analyzes their content to extract precise details required to generate answers. This functionality is of immense value to organizations seeking to improve knowledge accessibility, enhance employee performance, and deliver better customer service.

The core functions of the chatbot are to connect to the database, fetch relevant documents based on queries, analyze the content using AI models, and deliver concise and accurate responses. Additionally, the AI model provides insights into frequently searched topics and highlights knowledge gaps, thereby enabling continuous knowledge base improvement.

Client details:

Name: Confidential | **Industry:** Finance | **Location:** US

Technologies:

Python, FastAPI, PostgreSQL, SQLAlchemy, AWS EC2, , NumPy, Pandas, Matplotlib, Seaborn, FAISS, Nginx, Gunicorn, OpenAI

RAG Based Chatbot to Boost Efficiency

Project Description:

The project was mainly categorized into two different phases:

1. Application Overview

The front end was built using Adobe ColdFusion to provide an intuitive and user-friendly interface for employees. For authentication and authorization, employees are required to log in through the dashboard, post which, they can interact with the chatbot with their queries. The application supports features such as role-based access, user dashboards, document feedback submission, and admin functionalities to upload and update documents.

The backend was developed using Python with the FastAPI framework in order to ensure high performance and scalability. PostgreSQL was used as the database to store and manage document metadata and query logs. The backend APIs used SQLAlchemy as the ORM framework for database interactions.

The RAG system uses a combination of document retrieval and AI-based answer generation. The document retrieval component leverages embeddings generated using Sentence Transformers to identify relevant documents for a given query. The answer generation component, built using the OpenAI library, processes the retrieved documents to generate precise answers. Data preprocessing and embeddings are handled using libraries such as Pandas, NumPy and OpenAI, while visualization and analysis of chatbot performance are performed using Matplotlib and Seaborn.

2. Data Processing and Deployment Workflow

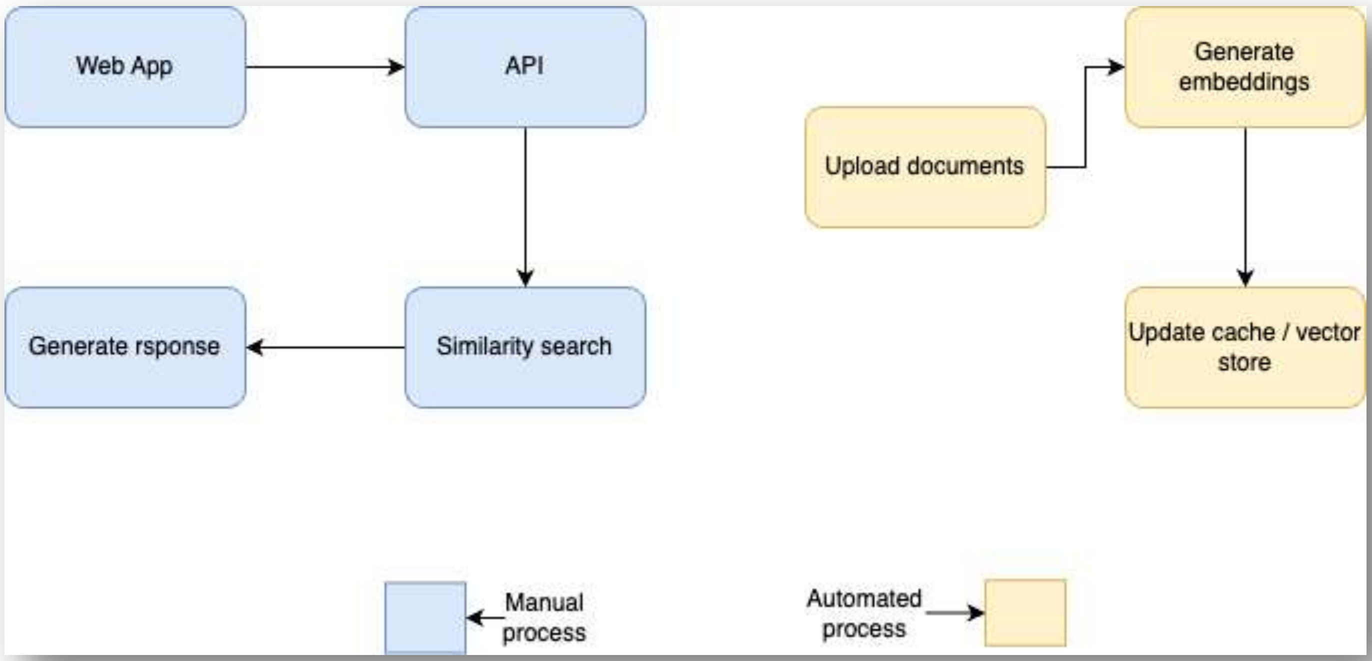
This is the backbone of the chatbot, ensuring efficient data handling and continuous updates. The workflow is multithreaded and supports batch processing of documents to handle large volumes of data. The core job of the workflow includes:

- Parsing and indexing newly uploaded documents to generate embeddings.
- Synchronizing the database to reflect any updates or deletions in real-time.
- Querying the indexed documents to retrieve relevant ones for AI analysis.
- Generating embeddings and storing them in a vector database for fast retrieval.

The chatbot is deployed on AWS EC2 instances to ensure scalability and reliability.

RAG Based Chatbot to Boost Efficiency

Model Architecture:



Variables Impacting the Performance of the Chatbot:

- The size and complexity of the document knowledge base.
- The quality of document embeddings and their indexing.
- The size of each document batch during preprocessing.
- Network bandwidth for handling high-frequency queries.
- The latency of AI model inference during answer generation.
- The concurrency of queries during peak usage.
- Parallel processing of multiple queries.

RAG Based Chatbot to Boost Efficiency

Key Measures to Overcome Optimization Challenges:

- **Optimized Data Retrieval:** The document retrieval pipeline uses approximate nearest neighbor (ANN) algorithms to accelerate the search process for relevant documents.
- **Batch Processing:** Large document batches are split into smaller chunks for efficient embedding generation and indexing.
- **Caching Frequently Accessed Data:** Popular documents and embeddings are cached to reduce retrieval times.
- **Real-Time Updates:** Document metadata is normalized and continuously synchronized to ensure accurate and up-to-date information.
- **Parallel Query Handling:** The system uses Python's multiprocessing and async capabilities to handle multiple queries simultaneously.
- **Memory Optimization:** Preprocessing pipelines are optimized to minimize the memory footprint of large datasets.

Screenshot: