Wine Chatbot

1. Overall Approach

The objective of this project is to build an intelligent chatbot for a wine business that can answer customer queries based on a specific corpus of information about their products. The chatbot is designed to:

- Provide quick, accurate responses to user questions.
- Use data only from a predefined corpus.
- Handle context by maintaining conversation history.
- Offer a minimalistic and user-friendly interface using Streamlit.

The approach involves the following steps:

Corpus Preparation:

- Extract text from PDF files containing business information.
- Load sample question-answer pairs from JSON for predefined responses.

Data Indexing:

- Preprocess and index the data into Elasticsearch for efficient querying.
- Use dense vector embeddings to represent textual data for semantic search.

Query Processing:

Implement semantic search using sentence embeddings to find relevant answers in Elasticsearch.
Handle out-of-scope questions by directing users to contact the business directly.

User Interface:

Create a minimalistic front-end using Streamlit for user interactions.
Enable question submission via a form with support for the Enter key.

Context Handling:

Maintain conversation history to allow context-aware responses.

2. Frameworks/Libraries/Tools Used

2.1. Elasticsearch

Usage: For indexing and searching through the corpus data.
Details: Used to store indexed documents and perform semantic searches with dense vector embeddings.

2.2. SentenceTransformers

Usage: For converting text into dense vector embeddings.
Details: The all-MiniLM-L6-v2 model is used for creating embeddings that represent the semantic meaning of the text.

2.3. Streamlit
Usage: For creating the web-based user interface.
Details: Provides an interactive and minimalistic frontend where users can submit questions and view responses.
2.4. PyMuPDF (fitz)
Usage: For extracting text from PDF documents.
Details: Used to read and extract text data from the business's PDF corpus.
2.5. Python
Usage: Programming language used for the development of the chatbot application.
Details: The code is written in Python, leveraging various libraries for data processing, model inference, and web development.

3. Problems Faced and Solutions
3.1. Indexing and Mapping Errors
Problem: Errors related to index mappings and updating existing indices.
Solution: Created a new index with updated mappings and reindexed the data from the old index to the new one.
3.2. Inaccurate Responses
Problem: The chatbot provided irrelevant answers, indicating issues with the semantic search.
Solution: Improved the cosine similarity threshold for better relevance and added filtering steps to ensure the accuracy of responses.

4. Future Scope
4.1. Enhanced Search Capabilities
Contextual Search: Implement more sophisticated context-aware search capabilities using advanced NLP techniques to better handle complex queries.
Multi-language Support: Extend the chatbot to support multiple languages, catering to a broader audience.
4.2. Feature Expansion
Voice Interaction: Integrate speech recognition and synthesis to allow voice-based interactions with the chatbot.
Personalization: Add user profiles and preferences to provide personalized recommendations and responses.
Integration with CRM: Connect the chatbot with a CRM system to provide real-time updates on orders, stock, and customer information.
4.3. Analytics and Feedback
User Feedback: Implement mechanisms to collect user feedback on responses to continuously improve the chatbot’s performance.
Analytics Dashboard: Create an analytics dashboard to monitor user interactions, identify common queries, and assess the chatbot's effectiveness.
4.4. Deployment and Scaling
Cloud Deployment: Deploy the chatbot to cloud services to ensure scalability and reliability.
Performance Optimization: Optimize the performance of the chatbot to handle larger volumes of queries with minimal latency.