DocBot ⛵

Enhancing Document Retrieval and Generation with RAG

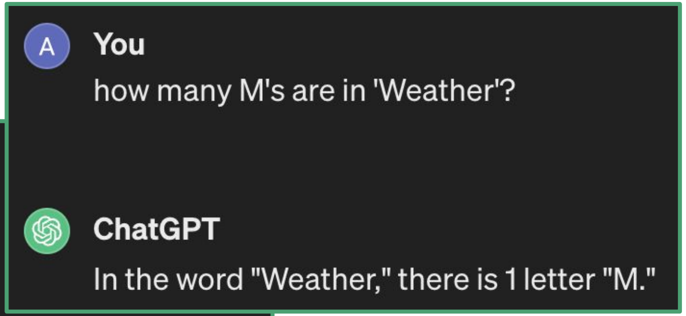
By: Charmi Chokshi

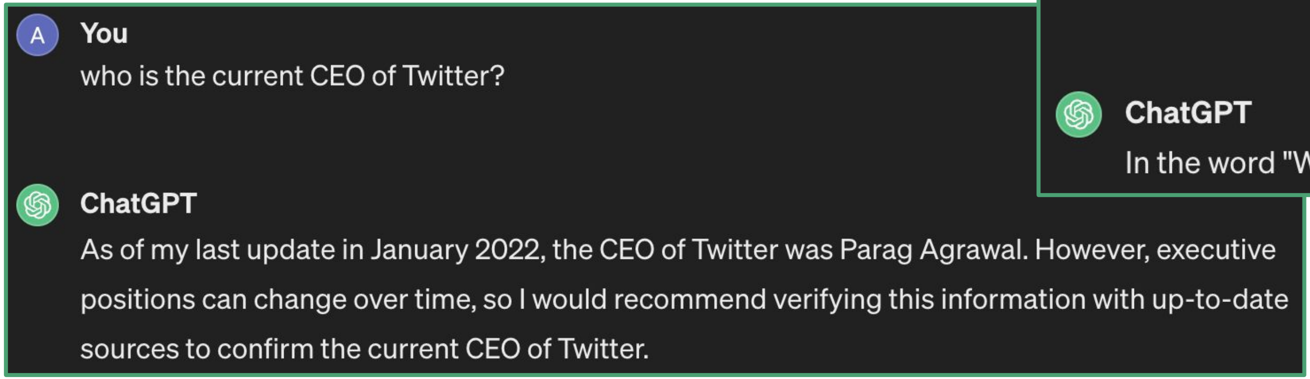
Agenda

- Problem Statement
- Solution Overview
- System Overview
  - RAG Framework
  - Technologies Used
  - DocBot Flow
- Features of DocBot ⛵
- Methodology
- Demo
- Future Work
- Conclusion
- Q&A

Problem Statement

- Large Language Models (LLMs) suffer from issues such as **Knowledge cut-offs and hallucinations** and they are not exposed to private data.
- Traditional search engines and FAQ systems fail to provide contextually relevant and natural language responses.





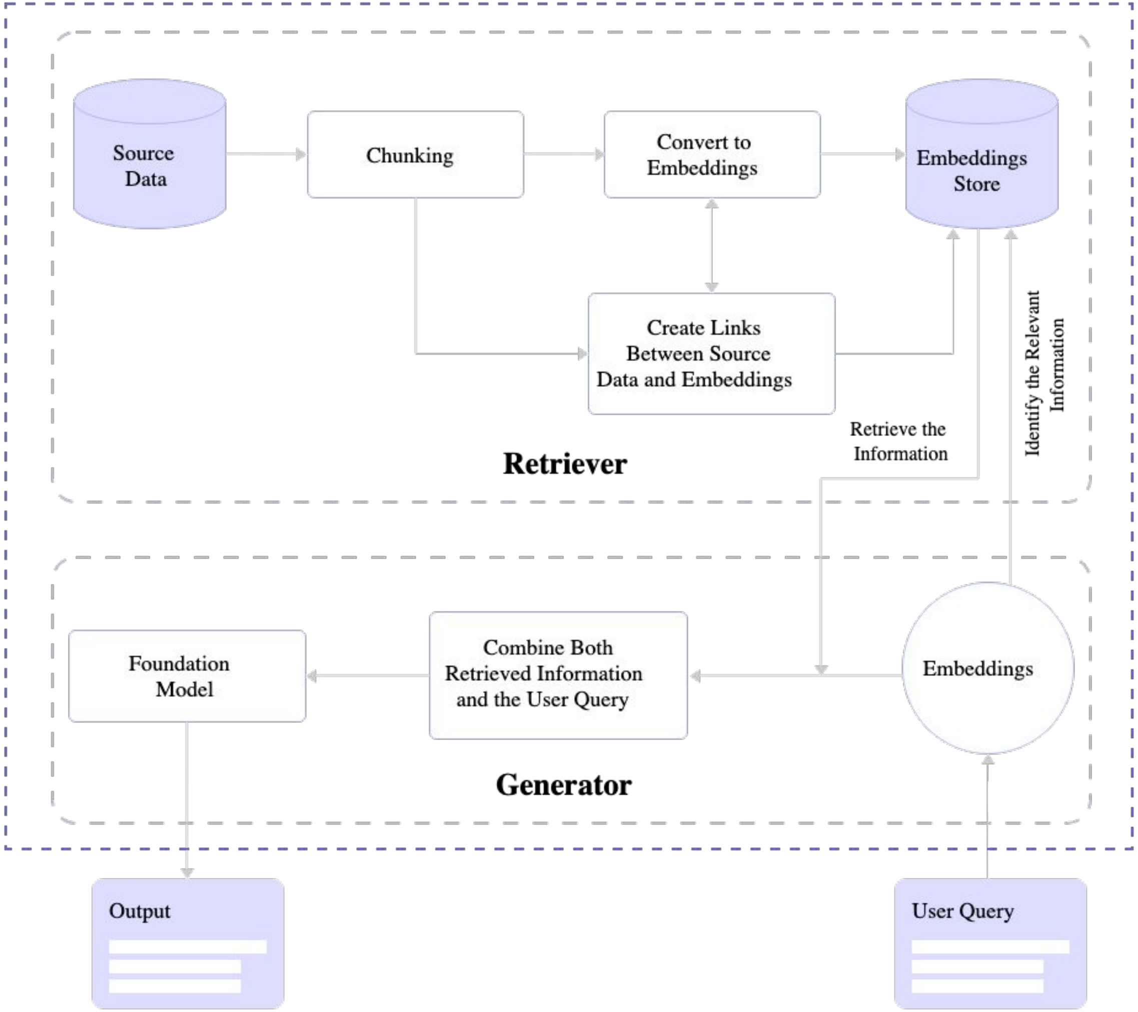
Solution Overview

- There is a need for an intelligent system that can provide **accurate and relevant answers based on specific document context while leveraging the language understanding capabilities of LLMs**.
- DocBot ⛵ is a Retrieval-Augmented Generation (RAG) chatbot that combines the power of LLMs with a searchable vector store created from user-provided documents.
- This approach ensures high-quality, contextually relevant responses without hallucinations, bridging the gap between traditional search engines and language models.

System Overview

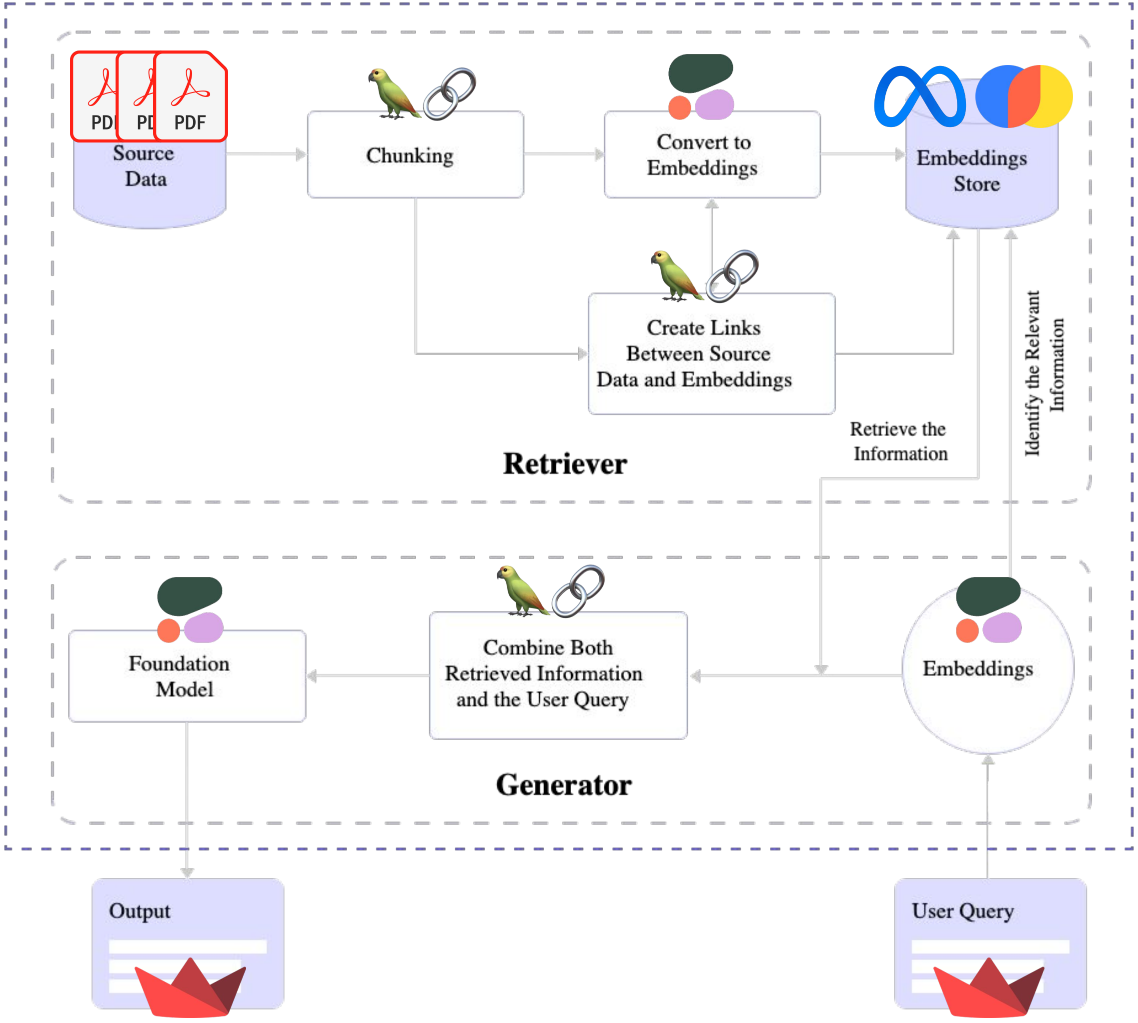
RAG Framework

- **Retriever**: Responsible for retrieving relevant information from a large knowledge base.
- **Generator**: Responsible for taking the retrieved information and generating coherent and contextually relevant responses or text.



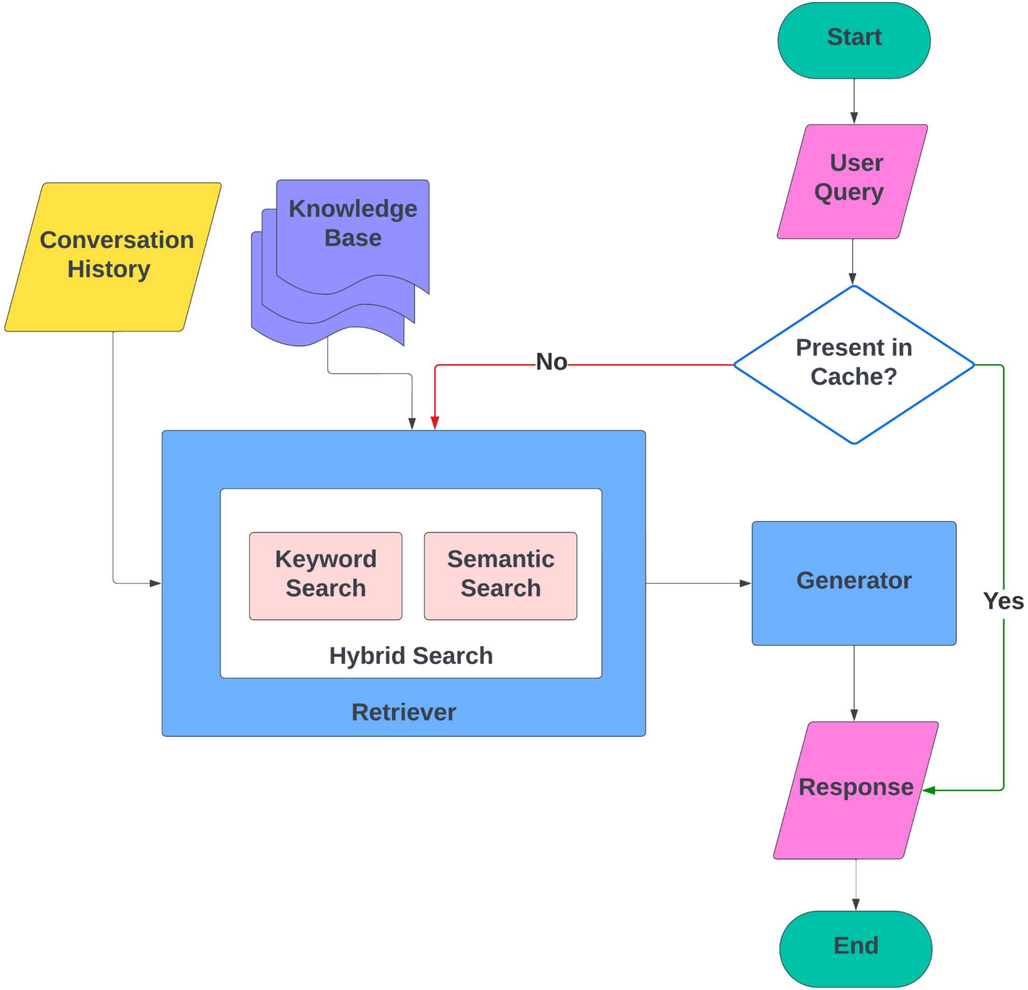
Technologies Used

- **LLM**: ChatCohere
- **Embeddings**: Cohere embed-english-v3.0
- **Doc Processing**: Langchain
- **Vector Store**: FAISS, Chroma
- **Web App**: Streamlit
- **PDF Processing**: PyPDF2



DocBot ⛵ Flow

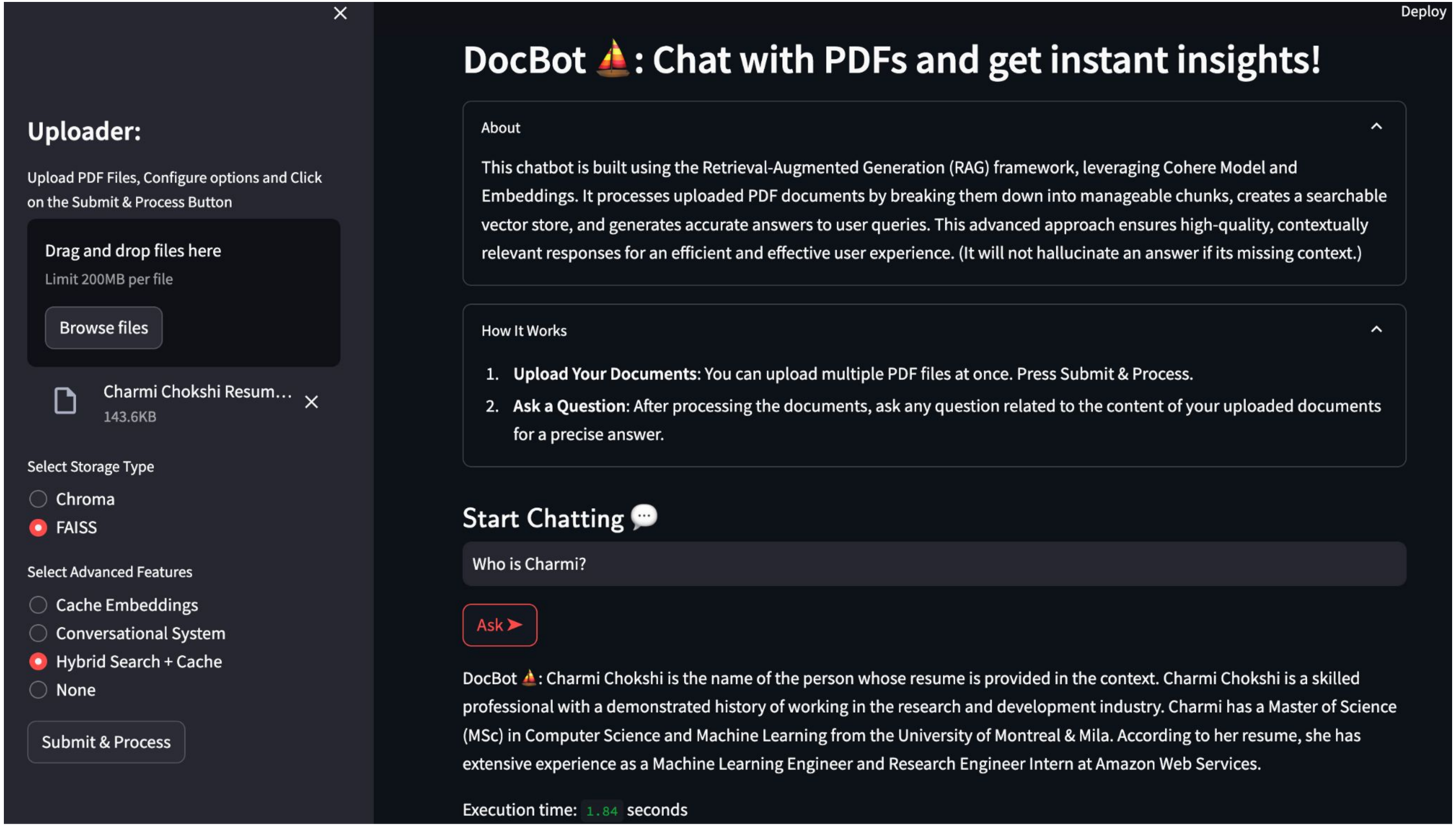
- **Caching Embeddings**: Embeddings are cached to avoid recomputing.
- **Hybrid Vector Search**: Combines keyword search (BM25 Algorithm) and semantic search (FAISS) for efficient retrieval.
- **In-Memory Caching**: Caches responses for previously asked queries to improve performance.
- **Conversational System**: Remembers and considers the context of previous questions and answers.



Features of DocBot ⛵

- Accurate and contextually relevant answers based on user-provided documents.
- No hallucinations or factual inconsistencies due to the use of RAG.
- Efficient and scalable document processing and retrieval.
- Conversational capabilities with context tracking.
- Flexible and extensible architecture.

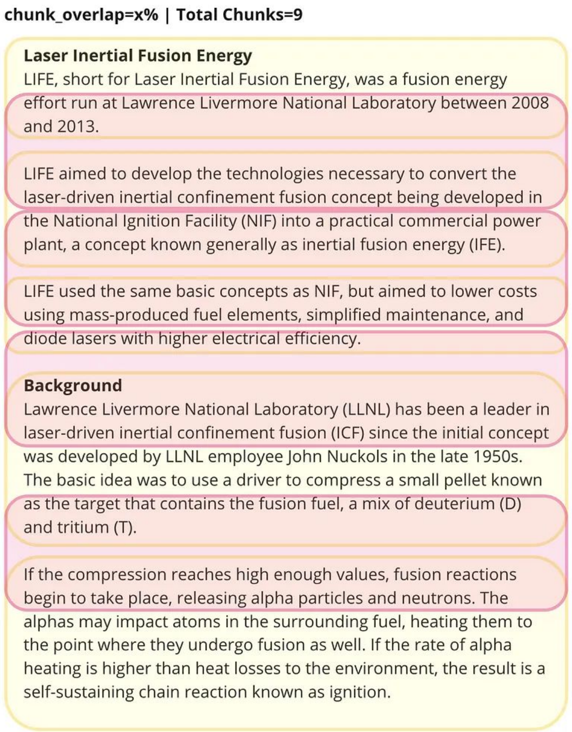
Features of DocBot ⛵



Methodology

Methodology: Document Processing

- Extract the text content and metadata from the PDF.
- Split them into **manageable chunks** for efficient processing and retrieval.
- The "RecursiveCharacterTextSplitter" class from LangChain is used.
  - "**chunk_size**" determines the maximum length (in characters) of each text chunk.
  - "**chunk_overlap**" specifies the number of overlapping characters between consecutive chunks.
- The resulting chunks are stored as Document objects in LangChain, ready for the next step.



Methodology: Vector Store

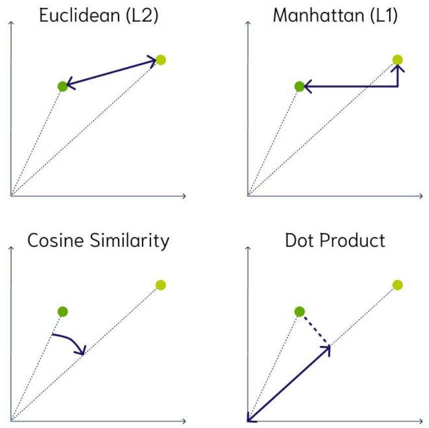
- Flexible Storage: Supports both FAISS and Chroma vector stores.
- Cohere's "embed-english-v3.0" model used for generating embeddings
- **Embeddings** computed for text chunks and user queries
- **FAISS index** or **Chroma vector store** created
- Text chunk embeddings stored for efficient similarity search

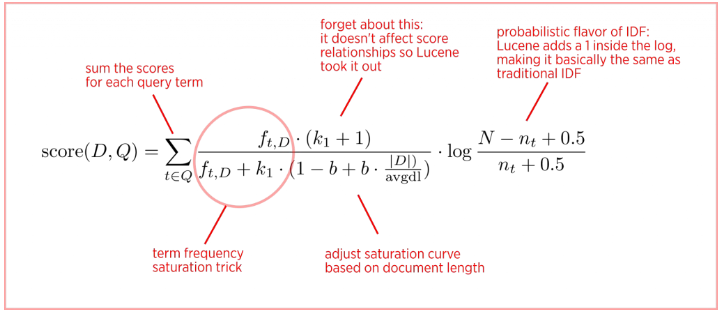
Methodology: Cache

- "CacheBackedEmbeddings" used for **caching embeddings**
- Avoids recomputing embeddings for previously processed text
- Improves performance and efficiency
- **In-Memory Caching**
  - Utilized for caching embeddings in memory
  - Provides fast access to cached embeddings
  - Suitable for shorter-term caching during runtime
  - Useful when the **cache size is relatively small**
- **Persistent Caching**
  - LocalFileStore explored for persistent caching on disk
  - Embeddings stored in './cache/' directory
  - Allows caching across multiple runs and sessions
  - Usefulness in scenarios where the **cache size is large**

Methodology: Searching Techniques

- **Similarity Search**
  - FAISS: "similarity_search" finds nearest neighbors in the vector space based on L2 (Euclidean) distances or dot products
  - Chroma: similarity_search retrieves documents with embeddings closest to the query embedding
- **Hybrid Retrieval Approach**
  - "EnsembleRetriever" used for combining vector and keyword search
  - FAISS for semantic vector search
  - BM25 Algorithm for keyword-based search
    - Ranking function used for keyword-based search, which ranks documents based on TF-IDF

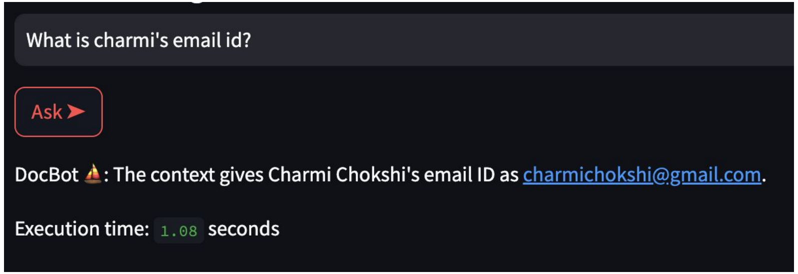


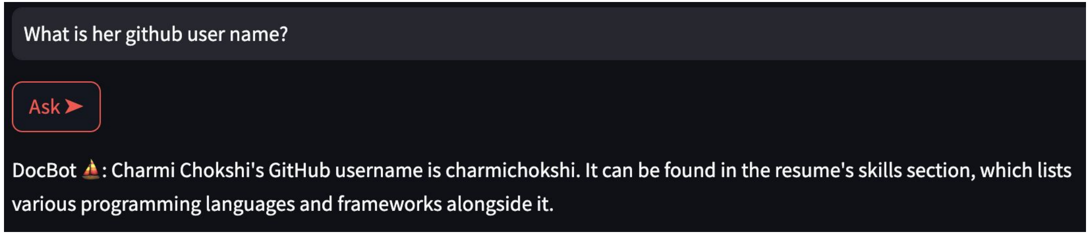


$$\text{score}(D,Q) = \sum_{t \in Q} \frac{f_{t,D} \cdot (k_1 + 1)}{f_{t,D} + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})} \cdot \log \frac{N - n_t + 0.5}{n_t + 0.5}$$

Methodology: Conversational History

- Enables the system to have a coherent, contextual conversation
- Conversation Buffer Memory
  - "ConversationBufferMemory" from LangChain used to store conversation history
  - Remembers previous questions and responses for context
- Conversational Retrieval Chain
  - "ConversationalRetrievalChain" combines retrieval and generation
  - Utilizes "load_qa_with_sources_chain" for answer generation
  - Leverages conversation history from "ConversationBufferMemory"





Demo

Future Work

- Fine-tuning Google Gemma-2B on domain-specific documents using PEFT techniques like QLoRA for improved performance in specific domains (e.g., finance).
- Data curation and continuous learning to enhance the knowledge base.
- Exploration of other state-of-the-art LLMs and vector databases.
- PoC has been done: https://github.com/charmichokshi/Experiments-with-Gemma

Conclusion

- Developed and Deployed a Chatbot using RAG framework.
- Explored various Caching and Vector Matching techniques for optimised retrieval of data from PDFs.
- These advanced approaches ensures high-quality, contextually relevant responses for an efficient and effective user experience.

References

- Code: https://github.com/charmichokshi/DocBot/
- App: https://docbot-by-charmi.streamlit.app/

Thank You!

Questions?