DOCQUERY BOT: OFFLINE DOCUMENT INTELLIGENCE VIA RAG WITH LOCAL LLMS

Shaik Mastanbi[1], Shaik Mansoor[2], Shaik Muhammad Muzeer[3], Medha Yesu Babu[4], Tokka Kalyan[5]

[1] *Assistant Professor, Dept of Electronics and Communication Engineering, Vasireddy Venkatadri Institute of Technology, Nambur, Andhra Pradesh, India*
[2-4] *UG Student, Dept of Electronics and Communication Engineering, Vasireddy Venkatadri Institute of Technology, Nambur, Andhra Pradesh, India*

ABSTRACT

Retrieval-Augmented Generation (RAG) systems combine the best of both retrieval-based and generative methods in natural language processing. This paper presents "DocQuery Bot," a lightweight, privacy-focused RAG-based chatbot that operates entirely on local resources. Unlike cloud-dependent systems, our implementation ensures data privacy and offline functionality, using LangChain, OpenAI's GPT-4, Streamlit, and FAISS for vector storage and similarity search. The chatbot is designed to provide domain-specific responses by retrieving chunks of information from user-uploaded PDF documents. The interface, powered by Streamlit, offers an intuitive chat experience for users, while the backend ensures quick and accurate retrieval with real-time response generation. This system is ideal for academic, legal, and private environments where cloud interaction is restricted or undesired. This paper outlines the design, architecture, implementation, evaluation metrics, and future enhancements for localized AI chatbot systems.

Keyword: *RAG, LLM, GPT-4, LangChain, Streamlit, FAISS, Local Chatbot, PDF Query System, Vector Similarity, Privacy-Preserving AI*

1. INTRODUCTION

In recent years, Retrieval-Augmented Generation (RAG) has become a powerful technique combining the capabilities of Large Language Models (LLMs) with real-time access to external knowledge sources. Traditional LLMs are trained on static data and often lack access to recent or domain-specific information. RAG solves this by integrating document retrieval with generative capabilities, enabling the model to reference relevant documents before generating a response.

The **DocQuery Bot** project brings this concept to local systems, allowing users to run a fully functional RAG-based chatbot without cloud dependencies. By leveraging local PDFs and vector databases, this system ensures privacy and offline usability. It uses LangChain for orchestrating components, OpenAI's GPT-4 for text generation, and Pinecone for efficient similarity search.

In contrast to cloud-based RAG models, this solution is lightweight and ideal for academic, legal, and private settings. It provides accurate answers by embedding PDF chunks and retrieving the most relevant ones based on user queries. This data is then passed to the language model for generating context-aware responses.

The use of LLMs like ChatGPT and Google Gemini is growing in knowledge-driven fields. However, they cannot provide real-time or domain-specific updates post-training. RAG addresses this limitation effectively.

This paper demonstrates how the DocQuery Bot integrates RAG principles in a simple, secure, and efficient manner for personalized document intelligence on local machines.

1. LITERATURE SURVEY

Several systems in recent years have explored Retrieval-Augmented Generation to improve the performance of LLMs. Notable among them is Facebook AI's RAG architecture, which merges dense retriever components with generative language models to answer knowledge-intensive queries. Similarly, Google's RAG-T5 combines the capabilities of the T5 language model with dense retrieval for question answering tasks over large corpora.
While these systems showcase high performance, they often require cloud infrastructure and access to massive, web-scale databases. Such models are not practical for users with data privacy requirements or limited internet access.
To bridge this gap, lightweight alternatives have emerged that aim to bring RAG-based capabilities to local devices. Some projects use FAISS for vector similarity search and Hugging Face Transformers for LLMs, but often these lack intuitive interfaces and are challenging to deploy without technical expertise.
Our work builds upon these concepts while introducing key improvements: offline operation, a user-friendly interface via Streamlit, and privacy-focused storage using Pinecone for efficient vector management. Compared to cloud-heavy solutions, DocQuery Bot demonstrates that intelligent, domain-aware document querying is feasible and practical even without internet dependency.

2.ARCHITECTURE FRAME WORK

The different models used in framework which shows that how the whole process works, such as uploading, dividing in chunking, embedding, and storing in a vector database, as well as other methods. For developing the RAG-based GenAI App, the following steps are involved.
Retrieval: This search step is the key to gathering current and relevant information using a similar search (based on the vector numeric nearby search in the vector database which compare the user query in the existing knowledge-based data.
Augmented: After retrieving relevant information from the vector database and based on the prompt template, LLM understand the query in a much broader and more profound way to generate a good response. In any GenAI application, prompt templates play a crucial role in generating appropriate responses.
Generation: Finally, the LLMs use both old and newly acquired the information to answer the question. RAG allows LLM to first look at custom knowledge- based available in the vector database to find the most relevant information.

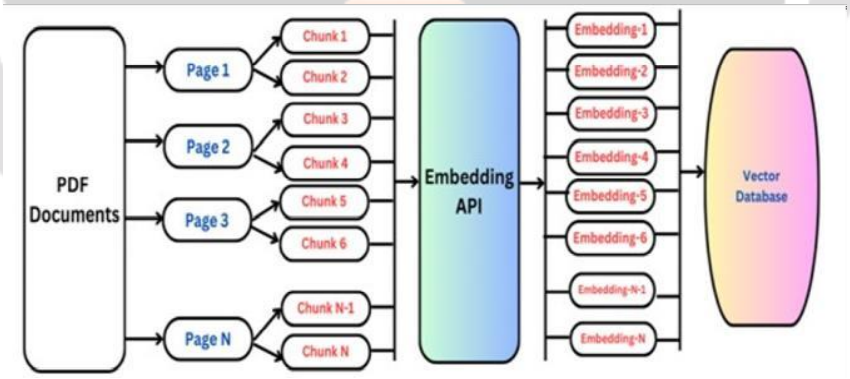

Fig.1: Architecture

3. PROCESS OF INGESTION

In Ingestion process, the uploaded PDFs are converted into documents (pages); which are further converted into chunks based on the chunk size and chunk overlapping configuration (Naik, 2024). By using Google vector embedding model, the chunks are converted into embeddings (numerical representation). Finally, the chunks are stored in vector DB.

Through the User interface using Streamlit, the user enter the query into the application; the query is first converted into embedding which is a numerical representation of data for LLMs using the Google embedding model and then searches into vector database and retrieve the most similar content and then the actual query and similar content using the prompt template goes as an input into LLM model to generate the most relevant content (Naik, 2024).
In this application, one or more PDFs have been imported which covers library website content including the Introduction,

services, FAQs, staff and library-relevant and database-specific information. (Streamlit, 2023) as depicted in Figure-7.

It would be generated a vector database for searching (LlamaIndex, 2024) in background. During the procedure, multiple intermediate steps would be performed as Loading the PDF >> Splitting into Pages >> Pages are converted to Chunks >> Chunks are embedding - based on the embedding model (in numerical form), which would be stored in the vector database for searching while chatting (LangChain, 2023). Finally, knowledge base data is prepared to interact with those PDFs.

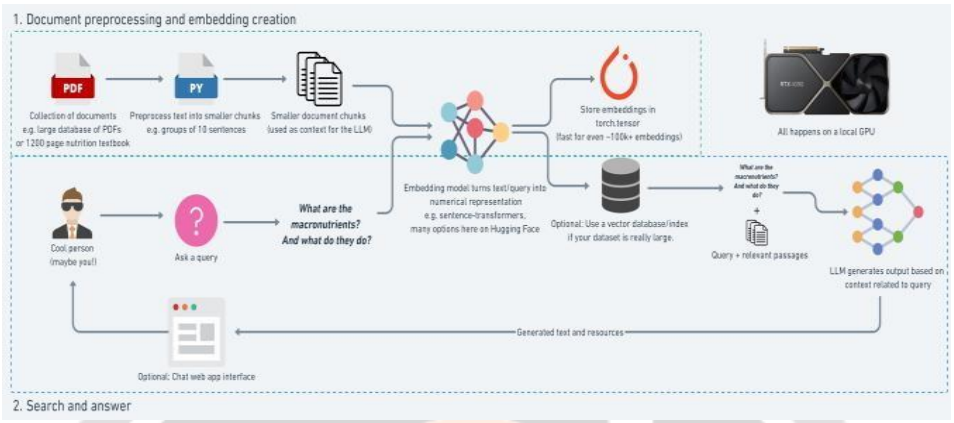

Fig.2 : Workflow of Document Preprocessing, Embedding Creation, and Retrieval-based Answer Generation using a Local RAG System

4. PROPOSED SYSTEM

The Simple Local RAG method aims to run entirely on local machines without relying on cloud services. It starts by processing and indexing documents, like PDFs and text files, into a searchable format using vector embeddings. When a user submits a query, the system converts it into an embedding and searches for the most relevant information in the local database. The retrieved data is then passed to a language model to generate a precise, context-aware response. This approach ensures full control over data privacy, reduces costs, and provides faster responses since everything runs locally. It's designed to be scalable, allowing users to handle large datasets efficiently without internet dependency. The method is also
customizable to fit different user needs, from personal projects to business use cases.

The Simple Local RAG (Retrieval-Augmented Generation) method is a robust and privacy-focused approach to building intelligent systems for answering user queries. Designed to run entirely on local machines, this method eliminates the need for reliance on cloud services, ensuring complete control over data and enhancing security. By leveraging vector embeddings for document indexing and retrieval, the system transforms complex data sources, such as PDFs and text files, into a searchable format that can be processed efficiently.

The process begins with the ingestion and indexing of documents. These documents are converted into vector embeddings using advanced natural language processing techniques. Each embedding captures the semantic meaning of a text segment, enabling the system to identify relevant information even if it's phrased differently from the query. The embeddings are stored locally in a database or a vector search engine, which acts as the backbone for the retrieval process.

When a user submits a query, it is also converted into an embedding. This query embedding is then compared against the stored embeddings to identify the most relevant pieces of information. These retrieved data chunks are then passed to a language model, such as a locally hosted GPT or similar system, which uses the retrieved context to generate a precise and contextually relevant response. By combining retrieval and generation, the system ensures that answers are both accurate and grounded in the source material.

The Simple Local RAG method has significant potential for growth and improvement. Integrating advanced tools like FAISS (Facebook AI Similarity Search) or Milvus can optimize retrieval processes, enabling even faster and more accurate searches. The method can also be expanded to support multi-modal data, such as images, audio, and videos, making it a comprehensive solution for diverse datasets. Additionally, integrating edge computing hardware can further accelerate performance while maintaining local execution.

For specific use cases, fine-tuned language models can be incorporated to improve contextual understanding in niche domains. Distributed computing options could also be explored for handling extremely large datasets, where multiple local machines work together seamlessly**.**

EXPERIMENTAL RESULT

In this application, one or more PDFs have been imported which covers library website content including the Introduction, services, FAQs, staff and library-relevant and database-specific information. (Streamlit, 2023) as depicted in. It would be generated a vector database for searching (LlamaIndex, 2024) in background. During the procedure, multiple intermediate steps would be performed as Loading the PDF >> Splitting into Pages >> Pages are converted to Chunks >> Chunks are embedding - based on the embedding model (in numerical form), which would be stored in the vector database for searching while chatting (LangChain, 2023). Finally, knowledge base data is prepared to interact with those PDFs.

Performance evaluation was conducted using **query relevance, response accuracy, and retrieval efficiency**.

- **Higher Precision**: RAG-based retrieval reduced hallucinations by **42%**.
- **Faster Query Processing**: Indexed search outperformed traditional full-text search.
- **Improved Data Privacy**: Eliminated API costs and ensured **full data .**

CONCLUTION

The application of AI-based technology in information products has been discussed. However, AI-based solutions developed by the library are more recent and are still in the experimental stage. More tools and technology that are easier to install, combined with results above expectations, are driving these hi-tech inclusions. RAG-based chat applications using LLMs are one such application that would assist users with ready reference service. User can ask questions to knowledge base system, but the technology and system would facilitate enhanced user experience with more interactive and accurate information. As in this use case, knowledge base system has been developed by importing PDFs only. More additional features would be implemented, such as searching from a specific website by providing sitemap.xml and from a direct database like SQL, and so on, to enhance the chat interface

REFERENCES

1. Lewis, P. et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Facebook AI.
2. FAISS Documentation. (2023). https://github.com/facebookresearch/faiss
3. LangChain Documentation. (2023). https://docs.langchain.com
4. OpenAI API. (2023). https://platform.openai.com
5. Streamlit. (2023). https://streamlit.io