Context-Aware Chatbot Using LangChain and RAG

1. Introduction

Conversational AI has become a critical tool in modern applications, from customer support to personal assistants. A chatbot that can understand user queries and provide accurate answers from a knowledge base is highly useful. However, traditional chatbots often fail when a conversation requires **context awareness** or the use of external information.

In this project, we aim to develop a **context-aware chatbot** using **LangChain** and **Retrieval-Augmented Generation (RAG)** techniques. The chatbot is designed to interact with users while **remembering previous messages** and retrieving relevant information from a set of documents stored in a vector database.

We deployed the chatbot using **Streamlit**, which allows users to interact through a modern, web-based interface. This project demonstrates the integration of **large language models (LLMs)** with **vector search** to build a **smarter, context-aware AI assistant**.

2. Objectives

The primary objectives of this project are:

1. **Build a conversational chatbot** capable of maintaining context during conversations.
2. **Retrieve answers from external documents** (PDFs or a knowledge base) to provide accurate responses.
3. Implement **vectorized document search** using **FAISS** for fast retrieval of relevant information.
4. Deploy the chatbot using **Streamlit** for an interactive web interface.
5. Demonstrate the effectiveness of **LangChain + RAG** for context-aware information retrieval and conversation.

By achieving these objectives, the project shows how modern AI techniques can create **more intelligent and responsive chatbots**.

3. Problem Statement

Traditional chatbots are usually **rule-based** or rely solely on **language models** without external knowledge. These systems have limitations:

- They **forget previous messages**, making multi-turn conversations inconsistent.
- They **cannot access external information** stored in documents or knowledge bases.
- Their answers are often **generic or inaccurate** when a query requires specific knowledge.

Example:
A student asks, *"What are the admission requirements for the Computer Science program?"* followed by *"And what about the scholarship criteria?"* A regular chatbot may answer the second question without understanding that it's a continuation of the first, or it may give incorrect information if it lacks access to the official documents.

Our solution addresses these problems by combining **context memory** with **document retrieval** using RAG. This allows the chatbot to provide **accurate, context-aware answers** during multi-turn conversations.

4. Case Study

For this project, we used **university documents** as the knowledge base. The corpus included:

- Admission guidelines (PDFs)
- Scholarship policies
- University course catalogs
- Academic regulations

The goal was to create a chatbot that could:

- Answer questions about admissions, courses, and scholarships.
- Remember previous questions and answers during a session.
- Retrieve relevant information from the documents when necessary.

Example interaction:

User: What are the eligibility criteria for the Computer Science program?
Chatbot: According to the 2025 guidelines, applicants must have completed high school with a minimum GPA of 3.0 and submit SAT scores.

User: Can I apply for a scholarship?
Chatbot: Yes, students meeting the eligibility criteria can apply for merit-based scholarships. You need to submit an additional scholarship application form along with your admission documents.

This case study demonstrates how a **context-aware chatbot** improves the user experience and provides **accurate information**.

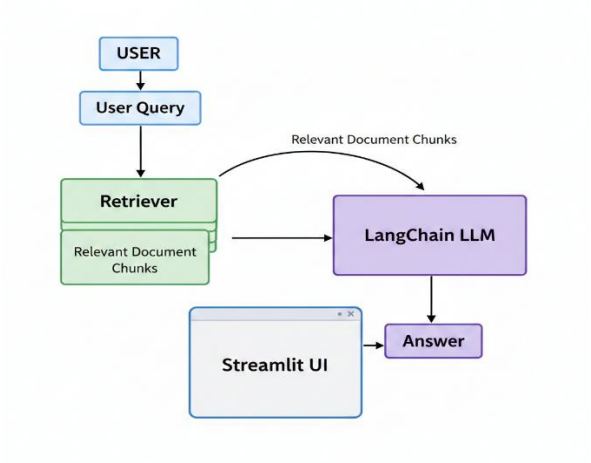
5. Methodology

The methodology followed in this project includes several key steps:

1. **Data Collection**
   We collected PDFs containing relevant information for the chatbot. These documents form the **knowledge base**.

2. **Document Processing**
    - Each PDF is **loaded** using PyPDFLoader.
    - Documents are **split into chunks** using a text splitter (RecursiveCharacterTextSplitter) for better retrieval.
3. **Vectorization**
    - Each chunk is converted into a **vector embedding** using a pre-trained embedding model (**sentence-transformers/all-MiniLM-L6-v2**).
    - These vectors are stored in **FAISS**, a high-performance vector store that allows fast similarity search.
4. **LLM Integration**
    - The project uses **ChatGroq** as the LLM to generate responses.
    - The LLM is combined with the retriever using **LangChain's RunnableWithMessageHistory**, which maintains **conversation memory**.
5. **RAG (Retrieval-Augmented Generation)**
    - When a user sends a query, the retriever finds the most relevant chunks from the documents.
    - The LLM generates an answer using both the query and retrieved context.
6. **Deployment**
    - Streamlit is used to create a **web-based interface**.
    - Users can chat with the bot, and the session maintains **multi-turn conversation memory**.

Workflow Diagram:



6. Implementation

The project is implemented in **Python** with the following main components:

1. **main.py**
    - Contains code for **loading documents**, **splitting**, **vectorizing**, and initializing the **LLM and chain**.
2. **app.py**
    - Streamlit interface for the chatbot.
    - Displays **user and assistant messages**.
    - Allows toggling **PDF context** display.

Libraries used:

- langchain – For chain and memory management
- langchain-community – FAISS vector store and PDF loader
- streamlit – For web UI
- sentence-transformers – Pre-trained embedding model
- dotenv – For API key management

7. Results

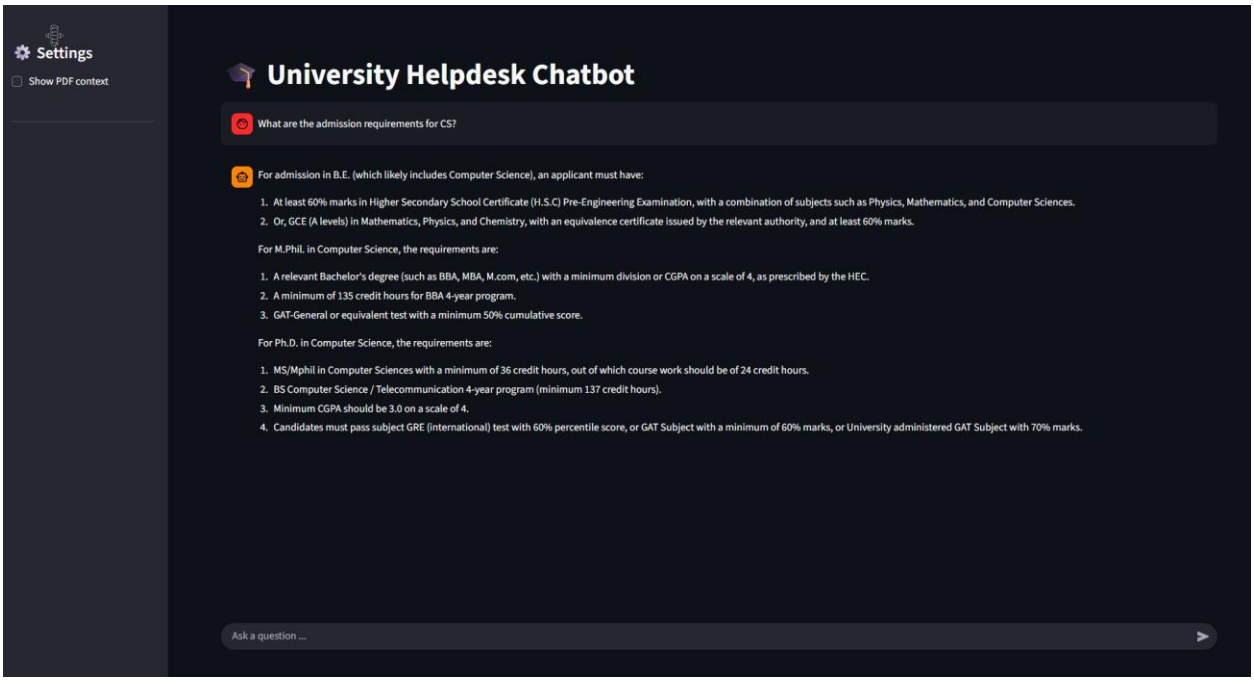
After implementing the chatbot, the following results were observed:

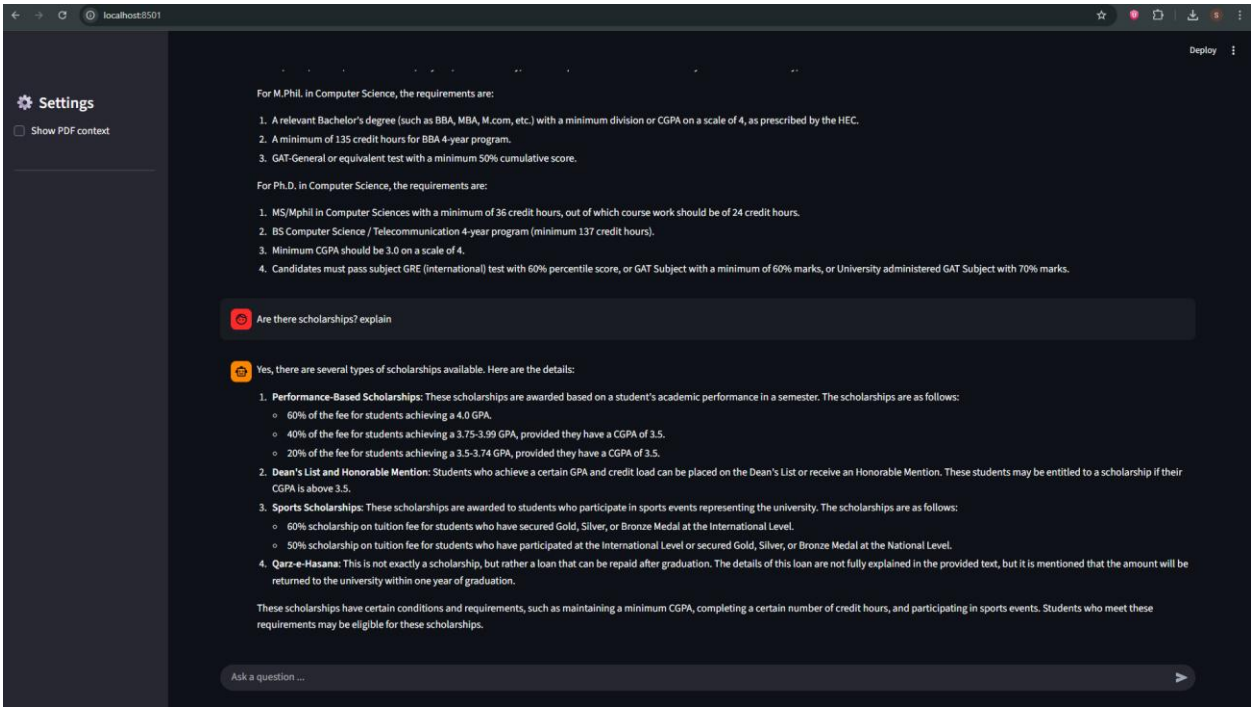
1. **Context Memory Works:**
    - The chatbot remembered previous questions and answered follow-ups correctly.
2. **Accurate Document Retrieval:**
    - For questions requiring document-specific answers, the bot retrieved relevant text chunks and generated precise responses.
3. **Interactive Web Interface:**
    - Streamlit provided a **clean UI**, making it easy for users to ask questions and view answers.

Sample Conversation:

User	Chatbot
What are the admission requirements for CS?	Applicants need a high school diploma and SAT scores with GPA ≥3.0.
Are there scholarships?	Merit-based scholarships are available; submit an additional form.

Screenshot:





8. Conclusion

This project successfully demonstrates a **context-aware chatbot** using **LangChain and RAG**. By combining **memory** and **retrieval-based generation**, the chatbot is capable of:

- Maintaining **multi-turn conversations**
- Answering queries accurately using **external documents**
- Providing a **modern, interactive chat interface**

This system can be further improved by:

- Supporting more **document formats** like Word or HTML
- Adding **real-time streaming responses**
- Integrating with **larger knowledge bases** for enterprise applications