Training/Internship Project

Apollo RAG Chatbot: An AI-Powered Solution for Hospital Query Resolution

Submitted by

Anvesh Mishra B.Tech CSE, Lovely Professional University, Punjab

Under the supervision of

Saurabh Mehrotra, Computer Centre, IIT Kanpur

LOVELY PROFESSIONAL UNIVERSITY

JULY 2025

EXECUTIVE SUMMARY

The Apollo Hospital Query Assistant project is a domain-specific chatbot system designed to answer queries related to hospital services, doctors, test packages, and other healthcare-related information, with a focus on Apollo Hospitals. Built using **LangChain**, **FAISS**, and a **locally hosted LLaMA 3 language model via Ollama**, the solution transforms static hospital brochures, doctor profiles, and service-related documents into an interactive, intelligent assistant. Users can upload multiple PDF documents and query them in natural language through a clean, responsive **Streamlit** interface.

The system follows a **Retrieval-Augmented Generation (RAG)** architecture without relying on cloud-based or paid APIs, ensuring privacy, offline capability, and cost efficiency. Uploaded PDF documents are parsed using **PyMuPDF**, split into manageable text chunks via the **RecursiveCharacterTextSplitter**, and embedded into semantic vector representations using **HuggingFace's MiniLM-L6-v2** embedding model. These vectors are indexed in **FAISS** for fast similarity-based retrieval.

When a user asks a question, the system embeds the query, searches for the top five most relevant text chunks, and passes them to the local **LLaMA 3** model for response generation. This approach ensures that each answer is contextually accurate and grounded in the uploaded hospital materials.
The application includes:

- **Multiple PDF upload support** from the sidebar.
- **One-time PDF processing** to optimize performance.
- **Persistent chat history** for tracking previous interactions.
- **Real-time response timing** for performance transparency.

By combining an open-source retrieval pipeline with a fully local language model, this project ensures data confidentiality — a critical factor in healthcare applications — while also providing adaptability for other medical or institutional domains.

In summary, the Apollo Hospital Query Assistant demonstrates the power of **local AI-driven RAG systems** in delivering efficient, accurate, and privacy-preserving information access in healthcare. It integrates document processing, semantic search, and local language modeling into a cohesive tool that can be easily extended to other specialized knowledge bases.

TABLE OF CONTENTS

INTRODUCTION

1.1 Background

In the healthcare sector, accurate and easily accessible information is essential for both patients and hospital staff. Hospitals like Apollo regularly produce a wide range of documents—such as doctor profiles, medical service brochures, and diagnostic test package details—most often stored in static formats like PDF. While these documents contain valuable information, locating specific answers by manually searching through them can be slow and frustrating, especially when time is critical.

Traditional support systems, such as hospital front desks or telephone helplines, often face high query volumes, leading to delays and inconsistent responses. As hospitals continue to expand their services and patient base, the demand for fast, reliable, and automated information delivery systems has become increasingly important.

Recent advancements in Artificial Intelligence, particularly in Large Language Models (LLMs) and tools like LangChain, have made it possible to create intelligent assistants capable of understanding natural language queries and retrieving relevant information directly from source documents. By combining these models with vector search technologies like FAISS and advanced text embeddings, static hospital documents can be transformed into dynamic, searchable knowledge systems.

This project applies these innovations to develop an AI-powered chatbot designed specifically for Apollo Hospitals. By enabling users to upload official hospital PDFs and ask questions in plain language, the system can quickly return precise, document-based answers. This not only enhances user experience but also reduces the workload on hospital support staff, offering a scalable and cost-effective solution for healthcare information access.

1.2 Motivation

In a hospital environment, timely and accurate information can directly impact decision-making, patient satisfaction, and overall service efficiency. However, the conventional methods of answering queries—such as in-person helpdesks, phone calls, or manual browsing of brochures—are often slow, require human resources, and are prone to inconsistencies in the information provided.

Apollo Hospitals, being a large healthcare provider with a wide range of services, faces the challenge of managing large volumes of repetitive questions from patients and visitors. Many of these queries relate to basic but essential information, such as doctor

specializations, consultation timings, available diagnostic packages, or cost details. Responding to each query manually is neither scalable nor efficient.

With the growing acceptance of AI in daily life, there is a strong motivation to create a solution that combines **speed**, **accuracy**, and **ease of access**. By leveraging modern AI technologies like Large Language Models (LLMs), LangChain, and FAISS-based document retrieval, it is now possible to develop a system that can interpret a user's question in plain language and provide a reliable, instant answer sourced directly from official hospital documents.

This project is motivated by the need to:

- Reduce the time taken for patients and staff to find accurate information.
- Minimize dependency on human-operated query handling systems.
- Improve the accessibility of hospital information for anyone, anywhere, at any time.
- Demonstrate how AI-powered Retrieval-Augmented Generation (RAG) systems can be applied in real-world healthcare settings.

1.3 Scope of the Project

The Apollo Hospital Query Assistant is designed as an AI-powered information retrieval system capable of understanding natural language queries and providing precise answers based on official hospital documents. The scope of the project covers the **design, development, and demonstration** of a Retrieval-Augmented Generation (RAG) chatbot that works entirely with locally processed data and a locally hosted LLaMA 3 model via Ollama, ensuring privacy and cost efficiency.

Key aspects within the scope include:

- **Multi-PDF Document Support** – The system allows users to upload multiple Apollo Hospital-related PDFs, such as brochures, doctor profiles, and test package details, in a single session.
- **One-Time Document Processing** – Uploaded files are processed only once per session, ensuring faster response times for repeated queries.
- **Natural Language Query Handling** – Users can ask questions in plain English without needing technical keywords or exact document phrases.
- **Accurate, Source-Based Responses** – All answers are grounded in the uploaded hospital documents to ensure reliability and trust.
- **User-Friendly Interface** – A web-based interface built with Streamlit provides intuitive PDF upload, chat history, and response timing features.
- **Local Model Execution** – All AI processing is done using a local LLaMA 3 model via Ollama, avoiding reliance on external APIs and protecting sensitive hospital data.

PROJECT DESCRIPTION AND GOALS

2.1 Objective

The main objective of the Apollo Hospital Query Assistant is to design and implement an AI-powered chatbot capable of providing accurate and instant responses to hospital-related queries using official Apollo Hospital documents as its primary knowledge base.
The main objective of the Apollo Hospital Query Assistant is to design and implement an AI-powered chatbot capable of providing accurate and instant responses to hospital-related queries using official Apollo Hospital documents as its primary knowledge base.

Specific objectives include:

1. **To simplify information access** by enabling patients, visitors, and staff to retrieve relevant hospital details without manually searching through lengthy PDFs.
2. **To ensure accuracy** by grounding all chatbot responses in authentic Apollo Hospital documentation.
3. **To improve efficiency** by minimizing repetitive manual query handling by hospital support staff.
4. **To develop a privacy-preserving system** that works entirely on local infrastructure using a locally hosted LLaMA 3 model via Ollama.
5. **To create a user-friendly interface** that allows PDF uploads, interactive chatting, and chat history viewing.
6. **To design a scalable architecture** that can be extended to other healthcare institutions or related domains in the future.

2.2 Problem Statement

Patients, visitors, and even hospital staff often face difficulties in quickly obtaining accurate information about hospital services, doctors, and diagnostic packages. Although this information is available in brochures and PDF documents, manually searching through them is time-consuming and inefficient.

Traditional support channels such as phone-based helplines or hospital front desks often experience high traffic, resulting in long waiting times, inconsistent responses, and increased workload for staff. Furthermore, in situations where quick decisions are needed—such as emergency visits or urgent test bookings—delays in accessing information can be problematic.

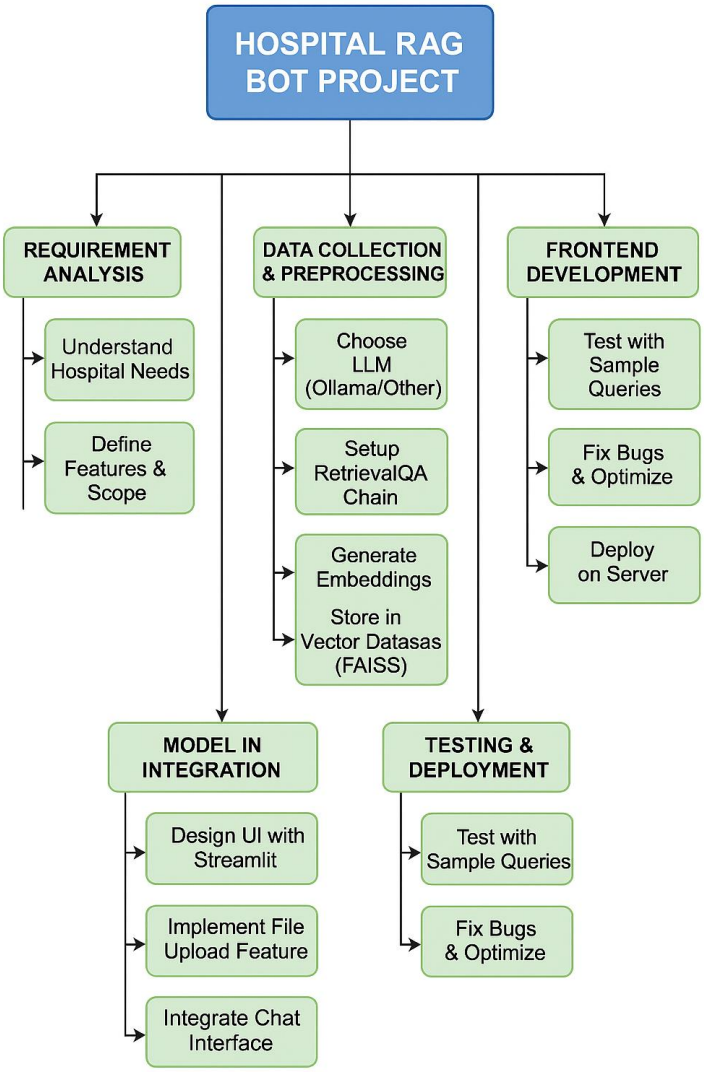
There is a clear need for an **automated, intelligent, and reliable** solution that:

- Accepts user queries in natural language.

- Retrieves only relevant and accurate details from official hospital documents.
- Operates without depending on cloud-based services, ensuring **privacy** and **cost-efficiency**.

The Apollo Hospital Query Assistant addresses this need by implementing a Retrieval-Augmented Generation (RAG) chatbot that combines semantic search, vector embeddings, and a locally hosted LLaMA 3 language model to deliver real-time, document-based answers.

2.3 Work Breakdown Structure



TECHNICAL SPECIFICATIONS

3.1 Requirements

3.1.1 Functional Requirements

The system must:

- **Accept multiple PDF file uploads** through an intuitive **Streamlit sidebar interface**, supporting drag-and-drop and file selection for ease of use.

- **Parse and read PDFs** using PyMuPDFLoader, ensuring accurate extraction of both text and layout while handling diverse document structures.

- **Segment extracted text** into **512-character chunks** with a **128-character overlap** using RecursiveCharacterTextSplitter to maintain context continuity across boundaries.

- **Generate semantic embeddings** using the Hugging Face model **all-MiniLM-L6-v2**, optimized for short-to-medium text similarity tasks.

- **Store vector embeddings** in a **FAISS vector database**, enabling efficient and scalable similarity search across potentially thousands of document chunks.

- **Perform similarity-based retrieval** to identify the **top 5 most relevant document chunks** for each user query, improving answer accuracy.

- **Use a locally hosted LLaMA3 model** (via Ollama integration) to generate **context-aware answers** that directly reference retrieved content

- **Maintain a persistent chat history** within the session state so the user can review past interactions without losing context.

- **Display query responses with a time indicator** showing how long the model took to process and respond, improving transparency and performance monitoring.

3.1.2 Non-Functional Requirements

- **Operate entirely offline, ensuring that sensitive data never leaves the local environment, which is critical in healthcare and confidential domains.**

- **Be lightweight and efficient, capable of running on mid-range hardware without requiring cloud computing resources.**

- **Maintain response times between 3–5 seconds for small to moderately sized document sets, with scalability considerations for larger datasets.**

- **Process multiple PDFs in one session without reloading, leveraging in-memory caching and session storage to avoid redundant computation.**

- **Deliver a clean, responsive user interface via Streamlit, ensuring accessibility for non-technical staff and minimizing training needs.**

- **Provide error handling for invalid or corrupted PDFs to prevent crashes and improve user experience.**

3.2 Feasibility Study

3.2.1 Technical Feasibility

- Implemented entirely with **Python** and **open-source frameworks** (LangChain, FAISS, HuggingFace, Streamlit).
- Uses **Ollama** for local LLaMA3 execution, avoiding cloud costs and network dependency
- All components are fully compatible and integrated via LangChain's modular design.

3.2.2 Economical Feasibility

- No licensing fees — all libraries are open-source.
- Hardware cost is a **one-time investment**; minimal operational costs thereafter.
- Reduces dependency on human staff for repetitive hospital-related queries.

3.2.3 Social Feasibility

- Improves efficiency in **hospital information access**.
- Frees up staff time, allowing focus on critical healthcare tasks.
- Builds trust in AI tools by maintaining **full data confidentiality**.

3.3 System Specifications

3.3.1 Hardware Specifications

- **CPU:** Intel i7 / AMD Ryzen 7 or higher
- **RAM:** 16 GB minimum
- **Storage:** 512 GB SSD
- **GPU (Optional):** NVIDIA RTX 3060 or above for faster LLaMA3 inference
- **Network:** Local LAN or standalone operation

3.3.2 Software Specifications

- **OS:** Windows 11 / Ubuntu 22.04 LTS
- **Programming Language:** Python 3.10+
- **Libraries & Frameworks:**
- `Streamlit` (UI)
- `LangChain` (orchestration)
- `PyMuPDF` (PDF text extraction)
- `HuggingFaceEmbeddings` (`all-MiniLM-L6-v2`)
- `FAISS` (vector storage & retrieval)
- `Ollama` (LLaMA3 model execution)
- **Model:** LLaMA3 (7B or 13B)
- **Other Tools:** `tempfile`, `os`, `time`, `ThreadPoolExecutor` for background processing

DESIGN APPROACH AND DETAILS

4.1 System Architecture

The Hospital RAG Bot is built on a modular architecture that integrates document processing, semantic search, and conversational AI to deliver accurate, context-aware responses from hospital-related PDF documents. The system is organized into **three main layers**:

1. ***Data Ingestion Layer:***

- ***PDF Upload Interface: Users upload one or more hospital-related PDF documents via the Streamlit-based interface.***
- ***Temporary Storage: Uploaded PDFs are stored temporarily for processing using Python's tempfile module.***
- ***Document Loader: The PyMuPDFLoader extracts text content from the uploaded files while preserving document structure.***

2. ***Processing & Knowledge Representation Layer:***

- ***Text Chunking: Extracted text is segmented into overlapping chunks using RecursiveCharacterTextSplitter (chunk size 512, overlap 128) to optimize retrieval relevance.***
- ***Embeddings Generation: Each chunk is converted into a high-dimensional vector using the HuggingFaceEmbeddings model (all-MiniLM-L6-v2), enabling semantic similarity search.***
- ***Vector Storage: The embeddings are stored in a FAISS vector database for fast and efficient nearest-neighbor retrieval.***

3. ***Retrieval & Response Generation Layer:***

- ***Retriever: Queries are matched with the most relevant chunks from FAISS using similarity search (k=5).***
- ***LLM Integration: Retrieved context is passed to the Large Language Model (LLM) served via Ollama (llama3), integrated into a LangChain RetrievalQA chain.***
- ***User Interface: The Streamlit chat interface displays user queries and AI responses in real-time, stores conversation history, and shows processing indicators such as time taken and progress spinners.***

METHODOLOGY AND TESTING

5.1 Methodology

The development of the Hospital RAG Bot followed an **iterative and modular methodology** to ensure efficiency, maintainability, and scalability. The workflow was divided into sequential phases, each with clear deliverables, while allowing for feedback and refinement.

Phase 1: Requirement Analysis

- Defined **functional requirements** (chat interface, PDF processing, vector search, LLM integration).
- Defined **non-functional requirements** (speed, accuracy, ease of use).

Phase 2: Data Collection & Preparation

- Gathered relevant hospital PDFs for the official Apollo Website (policy manuals, patient information documents, etc.).
- Used **PyMuPDFLoader** to extract text from PDF files.
- Split text into **512-character chunks** with **128-character overlap** using RecursiveCharacterTextSplitter to preserve context.
- Generated **semantic embeddings** using HuggingFace all-MiniLM-L6-v2 model.

Phase 3: Backend Development

- Created a **vector database** using **FAISS** for efficient semantic search.
- Configured **Retriever** with top-5 similarity matches to optimize accuracy.
- Integrated the **Ollama LLaMA 3 model** via LangChain's RetrievalQA pipeline.

Phase 4: Frontend Development

- Developed a user-friendly **Streamlit** web interface.
- Implemented PDF upload functionality in the sidebar.
- Added **real-time chat interface** with conversational history.
- Included system response time tracking for performance insights.

Phase 5: Integration & Testing

- Connected frontend to backend pipeline for seamless user interaction.
- Conducted **unit testing** for:
    - PDF loading and processing.
    - Embedding creation and retrieval accuracy.
    - Model integration.
- Performed **user acceptance testing** with sample hospital staff queries.

Phase 6: Optimization & Deployment

- Optimized text splitting and retrieval parameters for speed and accuracy.
- Deployed locally on a machine meeting the specified minimum hardware requirements.
- Documented system usage and troubleshooting steps.

5.2 Testing

To ensure the chatbot performs accurately and reliably, multiple forms of testing were conducted:

- Functional Testing:
  a) PDF Upload: Confirmed successful loading and parsing of multiple PDF files.
  b) Embedding Process: Verified that embeddings are correctly generated and indexed in FAISS.
  c) Query-Response Flow: Tested various types of queries including direct, indirect, and context-heavy questions

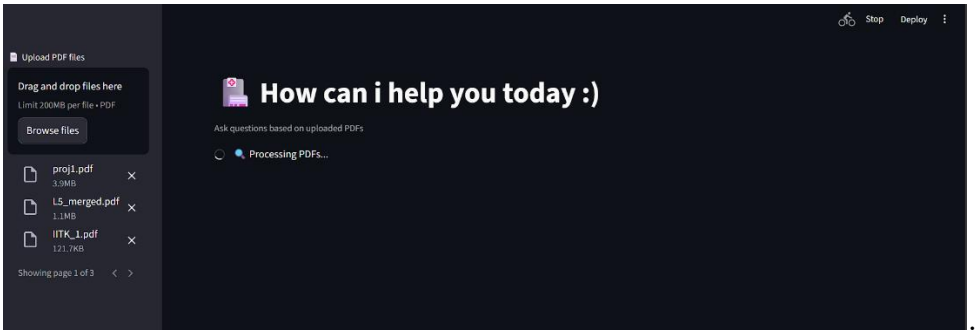


Fig 5.2.1-Ensuring PDF upload and embedding process

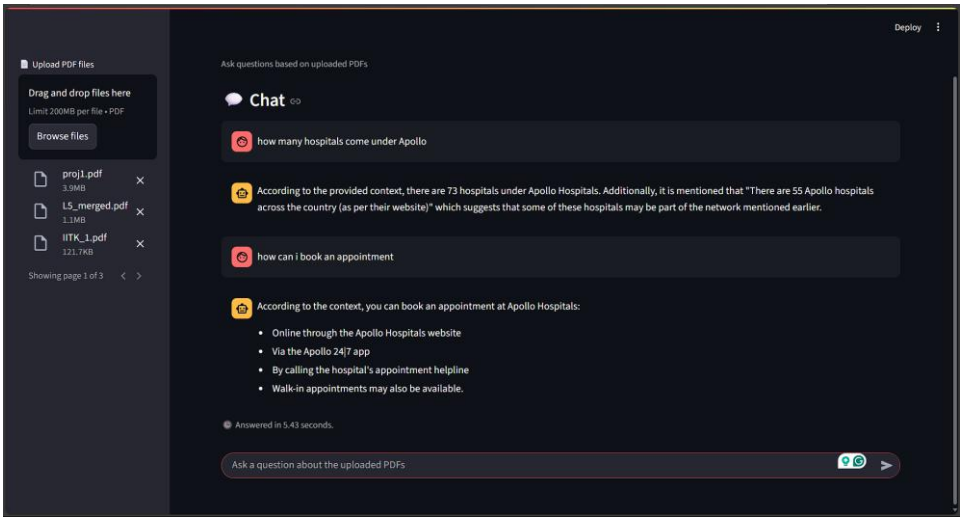


Fig 5.2.3 -Ensuring Query Response Flow

- Performance Testing:
    a) Measured average response time per query under different model sizes and concurrent loads.
    b) Ensured that even large PDFs did not crash the interface and that memory usage remained efficient.
- Accuracy Testing:
    a) Manually compared the responses of the chatbot against the actual content in the PDFs to evaluate retrieval and generation accuracy.
    b) Fine-tuned chunk size and retrieval k value to optimize relevance of responses.
- User Testing
    a) Simulated real-world use cases.
    b) Feedback was collected to iterate on prompt design and user experience.

PROJECT DEMONSTRATION

The project demonstration offers a practical showcase of the system's ability to accurately retrieve and respond to hospital-related queries using a LangChain-powered document-based chatbot. This section outlines the working flow, demonstrates core functionalities, and emphasizes the reliability and efficiency of the solution in addressing queries related to hospital services, doctor profiles, and medical test packages.

6.1 System Overview

The proposed Hospital RAG Bot comprises the following core components:

- **PDF Text Extraction Engine:** Extracts raw text from hospital-related PDFs such as brochures, doctor profiles, and test packages using PyMuPDF for downstream processing.
- **Text Chunking & Embedding Generator:** Breaks down the extracted text into manageable chunks and converts them into dense vector representations using HuggingFace Embeddings.
- **FAISS Vector Retrieval Engine:** Performs fast similarity-based retrieval to fetch contextually relevant document chunks in response to user queries.
- **Language Model Response Generator:** An Ollama-powered LLM generates coherent, natural language responses by synthesizing the retrieved context.
- **Streamlit-Based Chat Interface:** A user-friendly web interface allowing users to upload PDFs, ask hospital-related questions, and receive conversational responses with memory persistence.

6.2 Working Flow

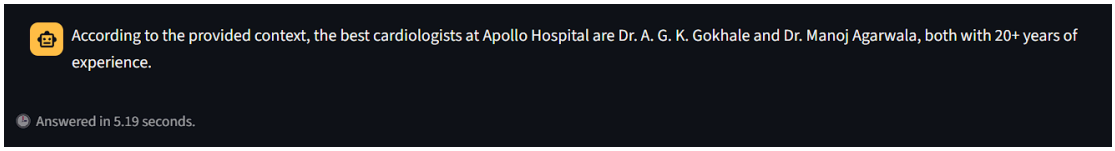
- **User Query Input:** A user types a question into the chatbot interface related to hospital information (e.g., “What tests are included in the full body checkup package?”).
- **Text Embedding Generation:** The user’s question is transformed into a vector embedding using the HuggingFace embeddings model, capturing the semantic meaning of the query.
- **Document Retrieval:** FAISS compares the query embedding with the stored embeddings from the uploaded hospital-related PDF chunks (such as brochures, doctor profiles, and test packages) and retrieves the most relevant sections.
- **Response Generation:** The retrieved chunks are passed, along with the original query, to the Ollama LLM through LangChain’s RetrievalQA chain. The model generates a context-aware response based on the retrieved content.
- **Response Display:** The final answer is displayed in the Streamlit chat interface in a conversational format. Users can continue asking follow-up questions, with the chatbot retaining context for the ongoing session.

6.3 Demonstration

During the demonstration phase, several test cases were used to validate the system’s performance. Below is an example scenario:

6.3.1 Test Query 1

Output:



Conclusion and Future Enhancements

The Hospital RAG Bot successfully demonstrates the integration of Retrieval-Augmented Generation (RAG) with LangChain to provide accurate, context-aware responses based on hospital-related documents. By leveraging PDF-based data sources such as brochures, doctor profiles, and test packages, the system ensures that users receive precise information without having to manually search through lengthy documents. The combination of FAISS for efficient vector-based retrieval and an Ollama-powered LLM for natural language responses results in a reliable, interactive chatbot experience. This solution not only improves accessibility to hospital information but also showcases the potential of AI-driven tools in streamlining patient and visitor queries.

Future Enhancements

While the current implementation meets the primary objectives, several improvements can be pursued:

- **Multi-document Type Support** – Extend support beyond PDFs to include Word documents, Excel sheets, and images with OCR.
- **Voice Query Integration** – Allow users to ask questions verbally and receive spoken responses for better accessibility.
- **Multi-language Support** – Implement multilingual capabilities to cater to diverse patient demographics.
- **Real-time Data Updates** – Integrate with hospital databases or APIs for dynamic information such as doctor availability, appointment slots, or updated test prices.
- **Advanced Context Memory** – Enhance long-term session memory to handle extended conversations without losing context.
- **User Interface Improvements** – Develop a mobile-friendly version or integrate the chatbot into the hospital's official website and app.

These enhancements would not only improve user experience but also scale the system for broader institutional deployment, potentially across departments or other academic campuses.