Project Report: HealthCare Chatbot

Student Name: Aman Kumar **UID: 24MCA20154**
Branch: MCA **Section/Group: 3A**
Semester: 1 st **Date of Performance: 29-Oct-2024**
Subject Name: PYTHON PROGRAMMING LAB **Subject Code: 24CAH-606**

1. **Aim of the project:** The aim of this project is to develop an intelligent healthcare chatbot using Python, designed to assist users in obtaining accurate, timely information on health-related topics, self-care advice, and guidance on minor symptoms.

2. **Software Requirements:**
   a) **Operating System:**
      - Windows: Specify supported versions (e.g., Windows 10, Windows 11, Windows Server 2019).
      - Linux: Mention supported distributions and versions (e.g., Ubuntu 20.04 LTS, CentOS 8, Debian 10).
      - macOS: List the supported versions (e.g., macOS 11.0 Big Sur, macOS 12.0 Monterey).

   b) **Python Installation:**
      - Python Version: Python 3.6 or later. Go to the Official Website: Visit the Python official website.

   c) **Install Anaconda and Jupyter Notebook:**
      - Visit the Anaconda Website: Go to the Anaconda distribution page.
      - Download the Installer: Click on the "Download" button for the Python 3.x version that matches your system architecture (64-bit or 32-bit).
      - License Agreement: Read and accept the license agreement by clicking "I Agree".
      - Installation Type:
      - Just Me: Recommended for personal use (installs Anaconda only for your user account).
      - All Users: Requires administrator privileges (installs Anaconda for all users on the system).

d) Python Packages:

- Chainlit
- Langchain
- Sentence-Transformers
- Faiss
- PyPDF2

3. Program Logic:

1. Document Loading and Vector Database Creation
DirectoryLoader and PyPDFLoader: Used to load PDF documents from a specified directory.
Text Splitting: Uses RecursiveCharacterTextSplitter to divide long text documents into smaller, manageable chunks to optimize for embedding.
Embedding: HuggingFaceEmbeddings with the model sentence-transformers/all-MiniLM-L6-v2 generates embeddings for each chunk of text.
Vector Store: A FAISS (Facebook AI Similarity Search) index is created from these embeddings, enabling efficient document retrieval. The vector store is saved locally for later use.
2. Retrieval QA Setup
Custom Prompt Template: Defines the structure of the response. It sets a context-based answer structure, instructing the model to only respond with accurate information or acknowledge when it doesn't know an answer.
Retrieval QA Chain: A RetrievalQA chain type retrieves relevant document chunks using the FAISS index and the embeddings to answer user queries. The retriever is set to return the top two most relevant document chunks based on similarity.
Language Model (LLM): The CTransformers model TheBloke/Llama-2-7B-Chat-GGML (a variant of LLaMA) processes the retrieved text to generate answers.
3. QA Bot and Final Result Functions
qa_bot: This function combines the language model, embeddings, vector store, and custom prompt to create the QA bot.
final_result: Takes a query and uses qa_bot to generate and return an answer.
4. Chainlit Integration for User Interaction
@cl.on_chat_start: Initializes the chatbot when the chat session begins, sending a welcome message.
@cl.on_message: Processes each message. When the user sends a query, the retrieval_qa_chain generates an answer with callback support to display the answer in real-time.
Logical Flow Summary
Document Preparation: PDF documents are loaded, split into chunks, and embedded for similarity search.
User Query Handling: The user's question is processed by retrieving relevant document chunks and generating an answer with the language model.
Interactive Chat: Real-time interaction is provided via Chainlit, sending responses to the user and showing sources when available.
Key Libraries and Their Functions
LangChain Community Extensions: HuggingFaceEmbeddings, FAISS, PyPDFLoader, DirectoryLoader
FAISS: Efficient similarity search and document retrieval.

CTransformers: For running the LLaMA model for text generation.

Chainlit: Enables a chat-based interface and user session handling.

4. Code:

A) ingest.py

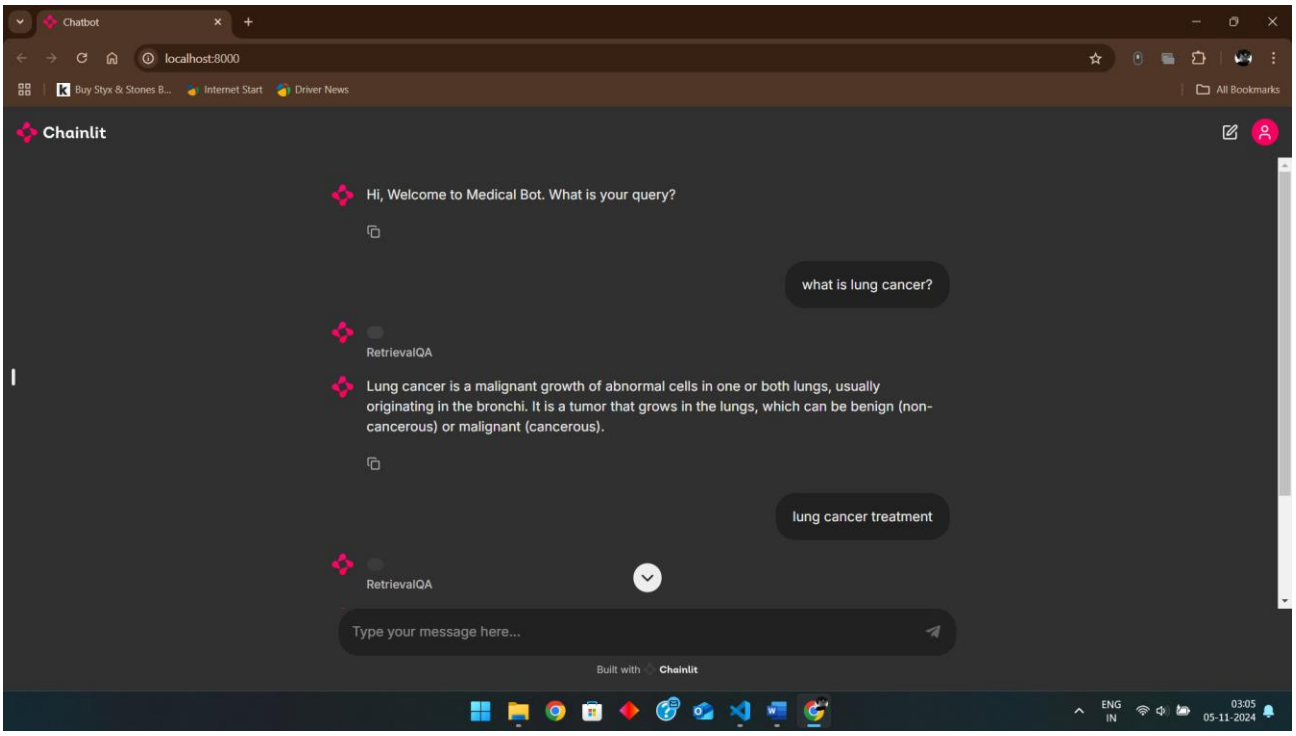
```
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
from langchain_community.document_loaders import PyPDFLoader, DirectoryLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter

DATA_PATH = 'data/'
DB_FAISS_PATH = 'vectorstore/db_faiss'

# Create vector database
def create_vector_db():
    loader = DirectoryLoader(DATA_PATH,
                             glob='*.pdf',
                             loader_cls=PyPDFLoader)

    documents = loader.load()
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=500,
                                                   chunk_overlap=50)
    texts = text_splitter.split_documents(documents)

    embeddings = HuggingFaceEmbeddings(model_name='sentence-transformers/all-MiniLM-L6-v2',
                                       model_kwargs={'device': 'cpu'})

    db = FAISS.from_documents(texts, embeddings)
    db.save_local(DB_FAISS_PATH)

if __name__ == "__main__":
    create_vector_db()
```

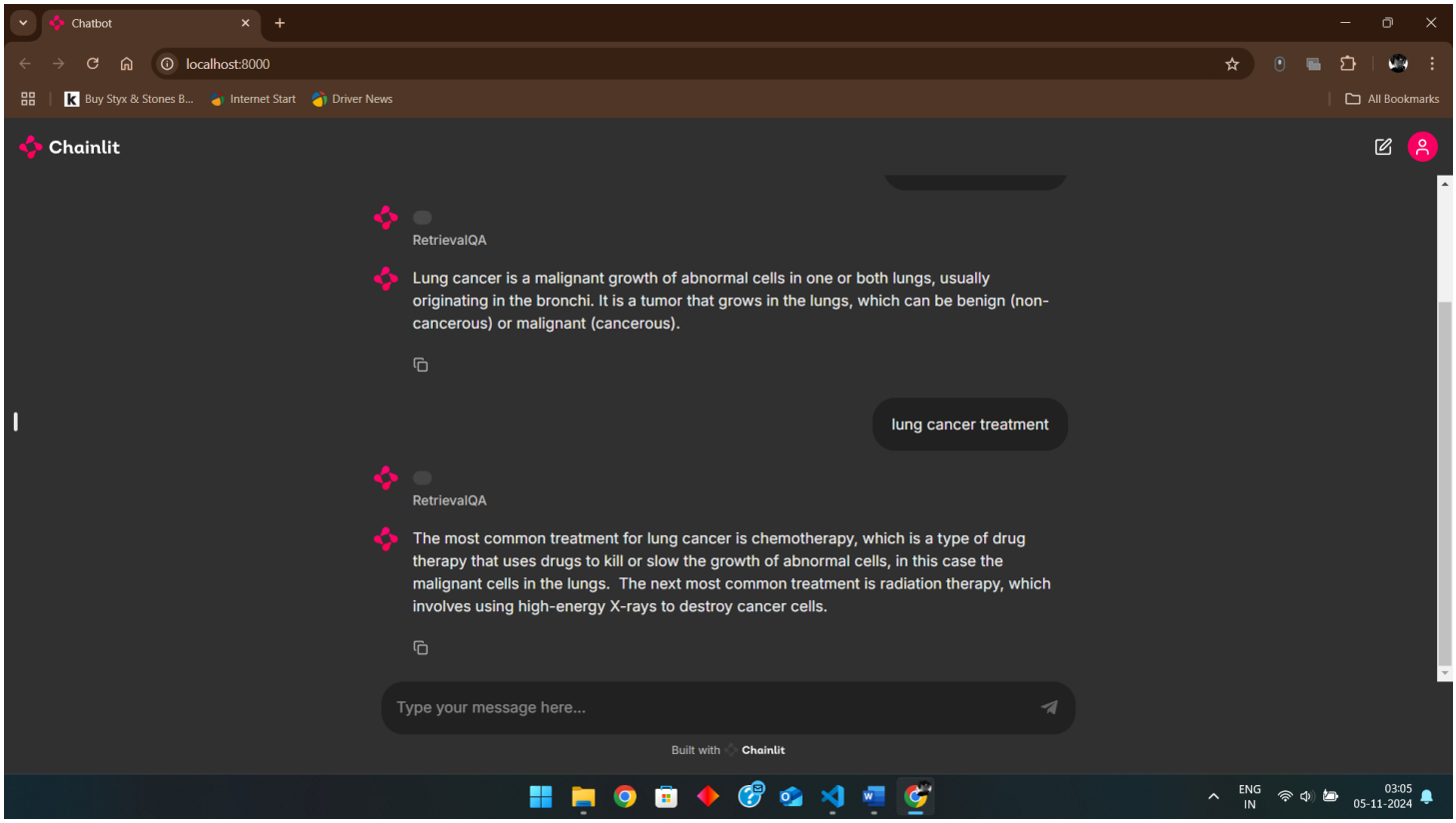
B) model.py

```
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
from langchain_community.document_loaders import PyPDFLoader, DirectoryLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter

DATA_PATH = 'data/'
```

```
DB_FAISS_PATH = 'vectorstore/db_faiss'

# Create vector database
def create_vector_db():
    loader = DirectoryLoader(DATA_PATH,
                             glob='*.pdf',
                             loader_cls=PyPDFLoader)

    documents = loader.load()
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=500,
                                                   chunk_overlap=50)
    texts = text_splitter.split_documents(documents)

    embeddings = HuggingFaceEmbeddings(model_name='sentence-transformers/all-MiniLM-L6-v2',
                                       model_kwargs={'device': 'cpu'})

    db = FAISS.from_documents(texts, embeddings)
    db.save_local(DB_FAISS_PATH)

if __name__ == "__main__":
    create_vector_db()
```

5. Result:





6.Learning outcomes (What I have learnt):

1.Vectorization and Embeddings for Document Retrieval
Understand how to use sentence-transformer models to convert text documents into vector representations, allowing for efficient semantic search and information retrieval.

2.Building a Vector Database with FAISS
Gain proficiency in using FAISS to construct and manage a vector database, ensuring quick and accurate retrieval of information from large datasets.

3.Implementation of Language Models for Contextual QA
Develop the skills to load and deploy language models (such as LLaMA) locally, and use them for context-sensitive question-answering.

4.Designing Prompt Templates for Effective Responses
Learn prompt engineering techniques to create customized prompts that guide language models in generating coherent, context-relevant responses.

5.Creating Document Loading and Processing Pipelines
Gain experience in designing pipelines to efficiently load, preprocess, and split large document sets (such as PDFs) for use in the chatbot.

6.Knowledge of Text Splitting for Improved Model Performance

Understand the importance of splitting documents into smaller chunks for better performance in QA tasks and apply techniques like recursive character splitting.

7.Integrating Custom Models and Embeddings in Applications

Explore the integration of custom models and embeddings (e.g., sentence transformers) to personalize and enhance chatbot responses.

8.Real-time Chat Interface Development with Chainlit

Build expertise in using Chainlit to create real-time chat interfaces, making AI-driven tools more user-friendly for end-users.

9.Session Management and Asynchronous Processing in Chatbots

Learn how to manage user sessions and process asynchronous requests within a chatbot, improving user interaction flow and response times.

10.Developing a Healthcare-specific Knowledge Bot

Acquire the ability to design and implement a chatbot tailored to healthcare, equipping it to handle medical queries reliably and retrieve relevant health information accurately.