CTSE LLM Chatbot: A Retrieval-Based Q&A System for CTSE Notes

1. Introduction

The **CTSE LLM Chatbot** project is designed to help students access CTSE notes more interactively using a retrieval-based AI system. Leveraging the power of **LangChain**, **HuggingFace**, and **ChromaDB**, this chatbot enables natural language querying over educational PDFs, delivering precise and concise answers. It aims to bridge the gap between static learning material and modern AI-powered study aids.

2. Objectives

- Convert PDF-based CTSE notes into an interactive knowledge base.
- Allow users to query these notes in natural language and receive relevant answers.
- Leverage open-source and lightweight LLMs for effective, local inference.
- Provide a terminal-based interface for rapid Q&A interactions.

3. Justification of LLM Choice

For this project, the language model selected was **google/flan-t5-base**, accessed via the HuggingFace pipeline. This model was chosen based on several considerations:

- **Lightweight & Efficient**: It supports local inference, making it suitable for resource-constrained environments such as laptops.
- **Instruction-Tuned**: The model is trained to follow prompts effectively, making it ideal for question-answering tasks.
- **Open-Source & Accessible**: Easily integrable through the HuggingFace ecosystem without needing GPU-accelerated hosting services.
- **Proven Performance**: Strong performance in multiple natural language understanding tasks, especially in low-latency Q&A.

Other larger models such as GPT-based APIs were considered, but they required cloud access and raised data privacy concerns. Hence, flan-t5-base was deemed the most balanced and practical choice.

4. Justification of Development Approach

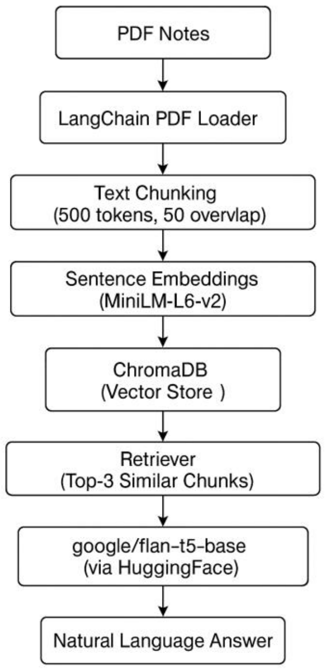
The project adopts a **retrieval-based Q&A system** over traditional fine-tuned models for the following reasons:

- **Scalability**: Easy to update the knowledge base by reloading or embedding new PDFs.
- **Efficiency**: Eliminates the need for expensive and time-consuming model training or fine-tuning.
- **Accuracy**: Combines the precision of semantic retrieval with the generation capabilities of LLMs.
- **Simplicity**: The modular design allows for clear separation between document processing, retrieval, and response generation.

This architecture was implemented using **LangChain** for chaining logic, **ChromaDB** for vector storage, and **HuggingFace** models for embeddings and generation.

5. System Design

- The system design follows this logical architecture:



This pipeline ensures that user questions are answered using only relevant chunks from the original PDF, maximizing both relevance and efficiency.

6. Challenges and Lessons Learned

Challenges Encountered

- **PDF Quality Variance**: Some PDFs had formatting issues that led to improper text extraction.
- **Chunk Overlap Tuning**: Finding the right chunk size and overlap required multiple trials to balance context completeness and redundancy.
- **Latency**: While flan-t5-base is efficient, response times can be improved with GPU acceleration or model quantization.
- **Testing Accuracy**: Manual evaluation was needed to assess the relevance of answers due to a lack of labeled datasets.

Lessons Learned

- A retrieval-based system allows for greater flexibility and lower computational cost than fully generative systems.
- Embedding quality and chunk granularity significantly impact final output accuracy.
- LangChain's modularity provides excellent control over each stage of the pipeline.

7. Technology Stack

Component	*Tool/Library Used*
Programming Language	**Python**
Notebook Interface	**Jupyter Notebook (.ipynb)**
PDF Loader	**LangChain – PyPDFLoader**
Text Chunking	**LangChain – RecursiveCharacterTextSplitter**
Embeddings	**HuggingFaceEmbeddings – all-MiniLM-L6-v2**
Vector Store	**ChromaDB**
Language Model (LLM)	**google/flan-t5-base via HuggingFace pipeline**
Retrieval Chain	**LangChain RetrievalQA**
CLI Interaction	**Python's built-in input () function**

8. Implementation Steps

Step 0: Install Dependencies

```
!pip install langchain langchain-community langchain-huggingface chromadb
transformers unstructured sentence-transformers
```

Step 1: Load PDF Document

- Load ctse_notes.pdf using LangChain's PyPDFLoader.

```
from langchain_community.document_loaders import PyPDFLoader
loader = PyPDFLoader("data/ctse_notes.pdf")
documents = loader.load()
```

Step 2: Split Text into Chunks

- Use RecursiveCharacterTextSplitter with chunk_size=500 and chunk_overlap=50.

```
from langchain.text_splitter import RecursiveCharacterTextSplitter
text_splitter = RecursiveCharacterTextSplitter(chunk_size=500,
chunk_overlap=50)
docs = text_splitter.split_documents(documents)
```

Step 3: Generate Embeddings

- Generate semantic embeddings using Sentence Transformers.

```
from langchain_huggingface import HuggingFaceEmbeddings
embedding_model = HuggingFaceEmbeddings(model_name="sentence-
transformers/all-MiniLM-L6-v2")
```

Step 4: Store Vectors in ChromaDB

- Save embedded vectors to a persistent vector store.

```
from langchain_community.vectorstores import Chroma
persist_directory = "./ctse_db"
vectordb = Chroma.from_documents(documents=docs, embedding=embedding_model,
persist_directory=persist_directory)
```

Step 5: Load Language Model

- Loads google/flan-t5-base via HuggingFace’s pipeline.

```
from transformers import pipeline
from langchain_huggingface import HuggingFacePipeline
```

```
hf_pipeline = pipeline("text2text-generation", model="google/flan-t5-base",
tokenizer="google/flan-t5-base", max_length=512, do_sample=False)
llm = HuggingFacePipeline(pipeline=hf_pipeline)
```

Step 6: Setup RetrievalQA Chain

- Combine retriever with LLM using RetrievalQA for intelligent answers.

```
from langchain.chains import RetrievalQA
retriever = vectordb.as_retriever(search_type="similarity",
search_kwargs={"k": 3})
qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever)
```

Step 7: Interactive Terminal Loop

- Ask questions interactively in a Jupyter-compatible loop.

```
def interactive_ctse_bot():
    while True:
        query = input("\nAsk a question (or type 'exit' to quit): ")
        if query.lower().strip() == "exit":
            print("Exiting the chatbot session.")
            break
        answer = qa_chain.invoke({"query": query})
        print("Answer:", answer["result"])

interactive_ctse_bot()
```

9. Additional Enhancement: CTSE_Question_Set.txt

To validate the chatbot, a file named CTSE_Question_Set.txt was added containing 25 real-world CTSE-related questions. These questions span various AWS concepts, CAP theorem, and scalability topics. This file is available in the project directory and can be used to test the robustness and accuracy of the chatbot.

10. Sample Test Case

User Query:

"What is software architecture?"

Expected Output (Generated by the chatbot):

A concise and relevant explanation on software architecture, based on the content in ctse_notes.pdf, typically including its definition, components, and importance.

11. Project Files

File	***Description***
CTSE_LLM_Chatbot.ipynb	Jupyter Notebook containing the chatbot code
ctse_notes.pdf	Source knowledge base
./ctse_db/	Persisted Chroma vector database
CTSE_Question_Set.txt	List of test queries to validate chatbot
CTSE_Assignment02_Report.pdf	This finalized report document

12. Conclusion

The CTSE LLM Chatbot is a successful demonstration of how modern AI tools can be used to create intelligent Q&A systems over static learning resources. With retrieval-based architecture and light LLM, the project is both scalable and efficient for local use. It simplifies student interaction with academic notes and enhances accessibility through natural language querying.