

Subject: Large Language Model

Name: Dipen Prajapati

Net ID: dp1435

Project 2 Topic:

Building a LangChain Research Assistant using Local HuggingFace Models

Introduction

This project report presents the development and operation of a light Research Assistant leveraging LangChain, HuggingFace pipelines, and Chroma vector databases. The assistant is capable of performing retrieval-based question answering (QA), explanations, summarizations, and conversational responses, all built on top of a local flan-t5-base model and Sentence Transformers embeddings. The goal is to demonstrate how an AI assistant can be built modularly by chaining a series of tasks with open-source models.

Research Assistant:

Objective

Building a modular Research Assistant capable of:

- Answering user queries from a custom document database (QA system),
- Providing simple explanations,
- Summarizing inputs,
- Engaging in conversational memory-based dialogue.

Methodology

1. Loading and Splitting Documents:

- A document (sample_doc.txt) is loaded and split into smaller chunks using a custom load and split docs function.
- This enables efficient embedding and retrieval at the chunk level.

2. Embedding and Vector Store Creation:

- A HuggingFace Sentence Transformer (all-MiniLM-L6-v2) is used to generate dense vector embeddings of the document chunks.
- These embeddings are stored and indexed in a Chroma vector database, allowing semantic retrieval of relevant chunks during QA.

3. Setting up Local HuggingFace Model:

- A text-to-text generation pipeline is initialized using the google/flan-t5-base model from HuggingFace.
- Key parameters like maximum token length (300), sampling (enabled), temperature (0.7), and top-p filtering (0.95) are configured to encourage diverse yet coherent generations.

4. Setting up Local HuggingFace Model:

Four different chains were created to modularize the assistant's capabilities:

- **RetrievalQA Chain:** Answers user queries based on relevant context retrieved from ChromaDB.
- **Simple Explanation Chain:** Uses a prompt to reframe any topic in easy-to-understand language.
- **Summary Chain:** Summarizes longer user inputs or retrieved information.
- **Conversational Memory Chain:** Maintains conversation history for more human-like dialogue continuation.

5. Running the Assistant:

- A continuous input loop was created where the user can query the assistant.
- For each query, the assistant responds with outputs from all four chains:
 - Retrieval-based answer
 - Simplified explanation
 - Summary
 - Conversational continuation

Results & Analysis

- **Question Answering:** RetrievalQA chain effectively fetches the most relevant content chunks from Chroma DB and uses flan-t5-base to generate coherent answers.
- **Simple Explanation:** The assistant can explain complex inputs in layman's terms reliably, showcasing the power of prompt engineering.
- **Summarization:** The summary chain gives a concise view of longer queries, useful for information distillation.
- **Conversational Memory:** The system maintains conversational continuity, remembering previous context between turns.

```

/Users/dipen/Desktop/LLM/LLMP2/main.py:45: LangChainDeprecationWarning: The class `LLMChain` was deprecated in LangChain 0.1.17 and will
be removed in 1.0. Use :meth:`~RunnableSequence`, e.g., `prompt | llm` instead.
  llm_chain = LLMChain(llm=llm, prompt=explain_prompt)
/Users/dipen/Desktop/LLM/LLMP2/main.py:59: LangChainDeprecationWarning: Please see the migration guide at: https://python.langchain.com/d
ocs/versions/migrating_memory/
  memory = ConversationBufferMemory()
● LangChain Research Assistant is ready!

Ask a question (or type 'exit'): What is machine learning. explain

📘 Answer from QA Chain:
Machine learning is the simulation of human intelligence processes by machines, especially computer systems.

💡 Simplified Explanation:
Machine learning is a type of artificial intelligence that tries to predict the behavior of humans.

📄 Summary Style:
This is the basic information that a machine learner must gather and use to solve a problem.

💬 Conversation Response:
Machine learning is a technology that is based on the use of machine learning algorithms in computers.

Ask a question (or type 'exit'): 

```

Conclusion

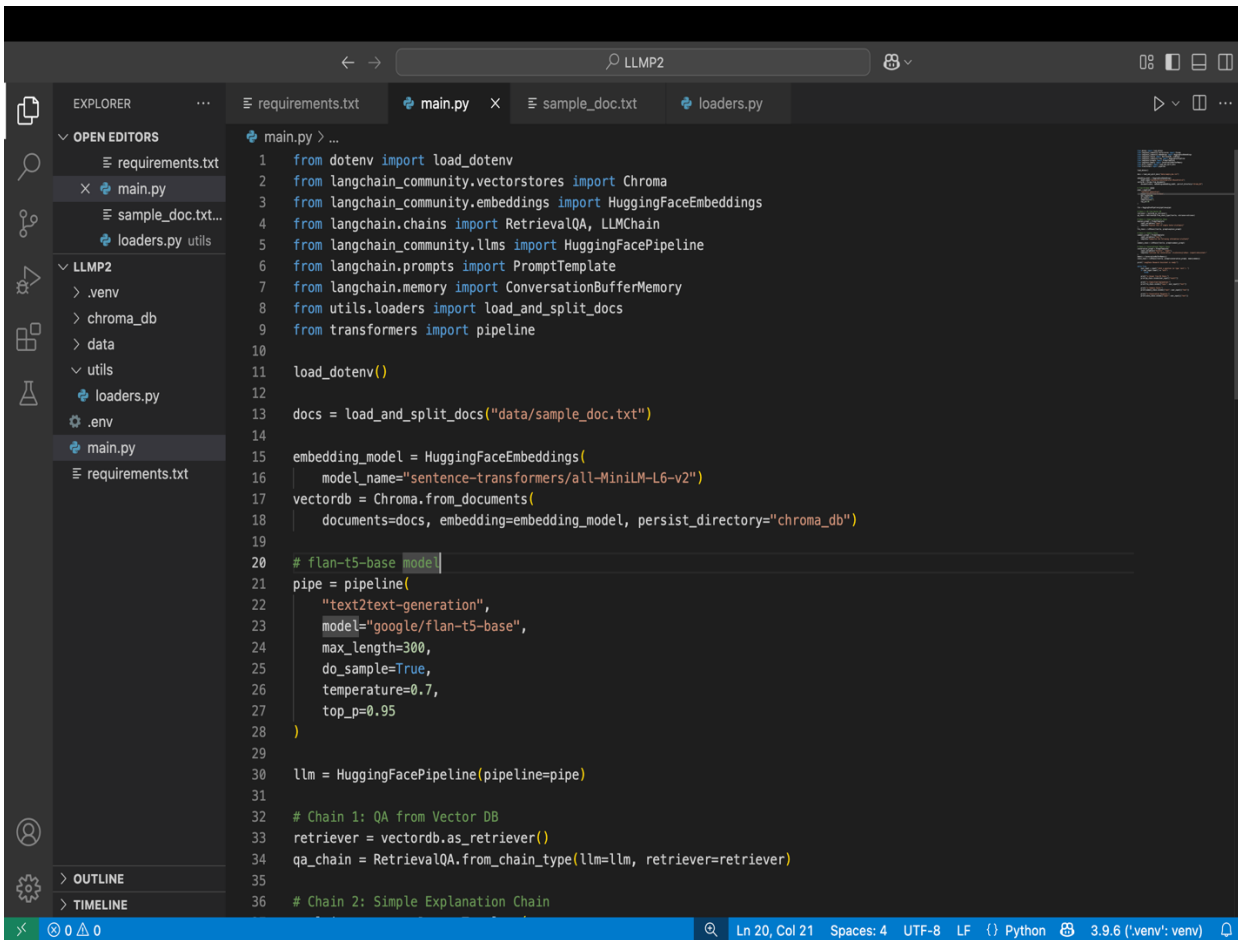
This project is a great example of developing a multi-purpose Research Assistant using open-source language models, LangChain chains, and a vector store database. Each task, from retrieval to explanation, summarization, and conversation, was modularized using different chains, providing flexible and scalable architecture.

The system highlights that even rather humble models like flan-t5-base can deliver strong performance if they are used in conjunction with careful prompting and retrieval enhancement.

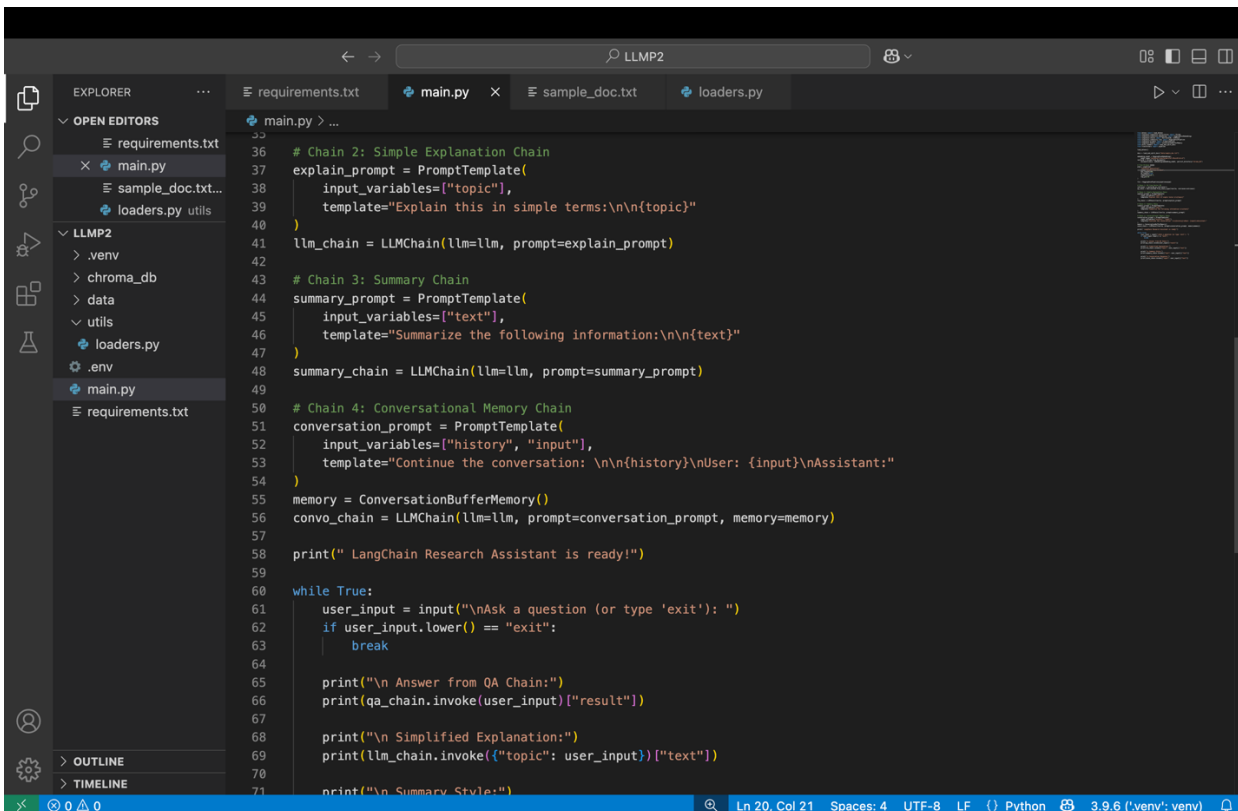
For future work:

- Larger or domain-specific models could be used for more nuanced answers.
- Enhancing conversation memory with summarization techniques for long-term chats.
- Adding more chains like translation, question reformulation, and opinion generation could make the assistant even more powerful.

The code are attached here below in this pdf.



```
1 from dotenv import load_dotenv
2 from langchain_community.vectorstores import Chroma
3 from langchain_community.embeddings import HuggingFaceEmbeddings
4 from langchain.chains import RetrievalQA, LLMChain
5 from langchain_community.llms import HuggingFacePipeline
6 from langchain.prompts import PromptTemplate
7 from langchain.memory import ConversationBufferMemory
8 from utils.loaders import load_and_split_docs
9 from transformers import pipeline
10
11 load_dotenv()
12
13 docs = load_and_split_docs("data/sample_doc.txt")
14
15 embedding_model = HuggingFaceEmbeddings(
16     model_name="sentence-transformers/all-MiniLM-L6-v2")
17 vectordb = Chroma.from_documents(
18     documents=docs, embedding=embedding_model, persist_directory="chroma_db")
19
20 # flan-t5-base model
21 pipe = pipeline(
22     "text2text-generation",
23     model="google/flan-t5-base",
24     max_length=300,
25     do_sample=True,
26     temperature=0.7,
27     top_p=0.95
28 )
29
30 llm = HuggingFacePipeline(pipeline=pipe)
31
32 # Chain 1: QA from Vector DB
33 retriever = vectordb.as_retriever()
34 qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever)
35
36 # Chain 2: Simple Explanation Chain
```



```
36 # Chain 2: Simple Explanation Chain
37 explain_prompt = PromptTemplate(
38     input_variables=["topic"],
39     template="Explain this in simple terms:\n\n{topic}"
40 )
41 llm_chain = LLMChain(llm=llm, prompt=explain_prompt)
42
43 # Chain 3: Summary Chain
44 summary_prompt = PromptTemplate(
45     input_variables=["text"],
46     template="Summarize the following information:\n\n{text}"
47 )
48 summary_chain = LLMChain(llm=llm, prompt=summary_prompt)
49
50 # Chain 4: Conversational Memory Chain
51 conversation_prompt = PromptTemplate(
52     input_variables=["history", "input"],
53     template="Continue the conversation: \n\n{history}\nUser: {input}\nAssistant:"
54 )
55 memory = ConversationBufferMemory()
56 convo_chain = LLMChain(llm=llm, prompt=conversation_prompt, memory=memory)
57
58 print(" LangChain Research Assistant is ready!")
59
60 while True:
61     user_input = input("\nAsk a question (or type 'exit'): ")
62     if user_input.lower() == "exit":
63         break
64
65     print("\n Answer from QA Chain:")
66     print(qa_chain.invoke(user_input)["result"])
67
68     print("\n Simplified Explanation:")
69     print(llm_chain.invoke({"topic": user_input})["text"])
70
71     print("\n Summary Style:")
```

LLMP2

EXPLORER

main.py

requirements.txt

sample_doc.txt

loaders.py

main.py > ...

requirements.txt

main.py

sample_doc.txt...

loaders.py

utils

LLMP2

.venv

chroma_db

data

utils

loaders.py

.env

main.py

requirements.txt

OUTLINE

TIMELINE

```
51 conversation_prompt = PromptTemplate(
52     input_variables=["history", "input"],
53     template="Continue the conversation: \n\n{history}\nUser: {input}\nAssistant:"
54 )
55 memory = ConversationBufferMemory()
56 convo_chain = LLMChain(llm=llm, prompt=conversation_prompt, memory=memory)
57
58 print(" LangChain Research Assistant is ready!")
59
60 while True:
61     user_input = input("\nAsk a question (or type 'exit'): ")
62     if user_input.lower() == "exit":
63         break
64
65     print("\n Answer from QA Chain:")
66     print(qa_chain.invoke(user_input)["result"])
67
68     print("\n Simplified Explanation:")
69     print(llm_chain.invoke({"topic": user_input})["text"])
70
71     print("\n Summary Style:")
72     print(summary_chain.invoke({"text": user_input})["text"])
73
74     print("\n Conversation Response:")
75     print(convo_chain.invoke({"input": user_input})["text"])
76
```

Ln 20, Col 21 Spaces: 4 UTF-8 LF Python 3.9.6 (.venv: venv)