Gen-AI-LAB 8

Experiment 8- Custom Prompt Template with LangChain and Cohere

1. Install langchain, cohere (for key), langchain-community.
2. Get the API key (By logging into Cohere and obtaining the cohere key).
3. Load a text document from your google drive.
4. Create a prompt template to display the output in a particular manner.

AIM:

To install LangChain and Cohere, obtain a Cohere API key, load a text file from Google Drive, and use a LangChain prompt template to display the output in a structured format.

Introduction:

LangChain and Cohere are tools that help us work with large language models (LLMs). A large language model is an advanced AI system that can read, understand, and generate human-like text based on input. These models can answer questions, summarize content, and even carry out conversations.

Cohere is a company that provides access to such models via an API key. Think of an API key as a password that lets us connect to their service. Once connected, we can send text to their model and get intelligent responses.

LangChain is a Python based framework that allows us to easily structure interactions with LLMs. One of its features is the **PromptTemplate** that helps us decide exactly how we want our input or output text to look.

A **custom prompt template** is a **user-defined format** used to instruct a language model (LLM) to generate output in a specific way. It is built using the PromptTemplate class in LangChain and allows you to:

- **Control how the model responds** (e.g., as bullet points, Q&A, tables, etc.).
- Insert **dynamic content** (like document text) into a fixed structure using **placeholders** such as {content}.
- Make prompts **clear, reusable, and structured** to improve the quality and consistency of model output.

For example, we can ask the model to summarize text in bullet points or explain a topic in a question answer format.

In this lab program, we will learn how to:

- Install the required libraries.

- Use your Google Drive to load a document.

- Enter your Cohere API key securely.

- Create a template to control the output format from the model.

- Generate a response using Cohere's language model and LangChain.

Objectives of the Program

1. Install the required libraries (langchain, cohere, langchain-community).

2. Load a text document from Google Drive.

3. Configure the Cohere API key securely in Colab.

4. Use LangChain's PromptTemplate to format and display content from the document in a structured way.

Step 1: Install LangChain, Cohere, and LangChain-Cohere Plugin

```
!pip install langchain cohere langchain-community langchain-cohere –quiet
```

Step 2: Import required Python libraries

```
from langchain import PromptTemplate
from langchain_community.llms import Cohere
from google.colab import drive
import os
from getpass import getpass
```

Step 3: Mount your Google Drive to access text files

```
drive.mount('/content/drive')
```

Step 4: Load and Read the text file content from Google Drive

```
file_path = "/content/drive/MyDrive/input.txt"
with open(file_path, 'r') as file:
    document_text = file.read()
print("Document loaded, length:", len(document_text), "characters")
```

Step 5: Set your Cohere API key securely (input will be hidden)

```
os.environ["COHERE_API_KEY"] = getpass("Enter your Cohere API key: ")
print("Cohere API key configured.")
```

To obtain a Cohere API key, follow these steps:

1. **Create a Cohere Account:**
    - Visit the Cohere Dashboard : https://dashboard.cohere.com/welcome/login
    - If you don't have an account, sign up by providing the required details.
    - Verify your email address to activate your account.
2. **Access the API Keys Section:**
    - After logging in, navigate to the API Keys page.
3. **Generate a New API Key:**
    - Click on “New Trial Key”.
    - Assign a name to your key for easy identification (e.g., “MyFirstKey”).
    - Click on “Generate Trial Key”.
    - Your new API key will be displayed. **Copy and store it securely**, as you won't be able to view it again later.

Step 6: Define the prompt format using LangChain's PromptTemplate

```
template = """
Summarize the following document in three bullet points highlighting the key ideas:

{content}

Bullet Point Summary:
"""
prompt = PromptTemplate(input_variables=["content"], template=template)
```

Step 7: Use the Cohere LLM to generate a response from the formatted prompt

```
llm = Cohere(max_tokens=150, temperature=0)  # temperature=0 for consistent output
formatted_prompt = prompt.format(content=document_text)
response = llm(formatted_prompt)

# Displaying the formatted response
print("Formatted Output:\n", response)
```

Output

```
Here are the key ideas from the document about AI in education:
1. AI in education primarily impacts how knowledge is delivered, absorbed, and assessed through personalized learning, AI-powered chatbots, and virtual assistants, and AI-graded assessments.
2. AI also increases accessibility and inclusivity in education through language translation services and disability assistance technology to break down communication barriers and support diverse needs.
3. However, the integration of AI in education comes with challenges such as data privacy concerns, the risk of AI biases, and the need for educator training. Institutions are urged to adopt ethical AI frameworks for optimal outcomes and equitable learning environments.
```

Final Note:

In this lab, we learned how to use LangChain and Cohere to create a custom prompt template for summarizing text. We used Google Drive to load a sample text file, entered our API key securely, and generated output using the LLM with a prompt structure we designed. This experiment demonstrates the power of using AI to format and control output, a key requirement in many real-world applications.

Appendix

Appendix A: Comparison of Cohere and Hugging Face

Cohere and Hugging Face are both widely used in Natural Language Processing (NLP) and AI development. While they offer similar functionalities like text generation, summarization, and embedding, they differ in purpose, access, and flexibility. The comparison below highlights how they align and differ:

Similarities:

- Both provide large language models (LLMs) for NLP tasks like summarization, text generation, classification.
- Both offer easy-to-use APIs for integrating language models into applications.
- Both can be integrated with LangChain for prompt-based LLM workflows.
- Both support common NLP use cases such as embeddings, Q&A, and more.
- Widely used by developers, researchers, and enterprises in AI applications.

Differences:

Feature	Cohere	Hugging Face
Focus	Offers proprietary hosted LLMs	Hosts open-source models from various contributors
Platform Type	Closed-source API platform	Open-source model hub and ecosystem
Model Access	Access Cohere's hosted models via API	Access, share, or download thousands of models
Community Sharing	Limited public sharing of models	Supports public model sharing and uploads
Main Offering	Text generation, classification, embeddings	Diverse models (BERT, GPT, T5, etc.) from many sources

Use Cohere when you want a scalable, proprietary model with stable API support. Use Hugging Face when you need access to a large variety of open-source models, community support, and more flexibility for experimentation.

Appendix B: LangChain Workflow Manager for LLMs

LangChain **acts like a** middleman **or** orchestrator that manages the entire interaction with the language model (Cohere).

What LangChain Does in this Program:

1. **Prepares the Input**
    - Using PromptTemplate, it formats the user input (e.g., a text document) into a structured message.
    - Example: “Summarize the following text in bullet points:\n {content}”
2. **Sends the Prompt to the LLM (Cohere)**
    - LangChain uses the integration (langchain-cohere) to pass your formatted message to Cohere's model.
3. **Gets the Response from the LLM**
    - Cohere processes the prompt and returns a response (like a summary in bullet points).
4. **Delivers the Response**
    - LangChain takes that output and makes it available to your Python program, which can then display it or process it further.

In Short:

LangChain handles the workflow like:

"Prepare the prompt → Talk to the LLM → Get the result → Show it to the user "

We do not need to manually handle API calls, input formatting, or response parsing, LangChain takes care of that.