**LANGCHAIN & NLP TASK DOCUMENTATION**

**Objective**

The goal of this assignment is to build a question-answering (QA) system using LangChain and an open-source language model (LLM) such as GPT-2 or GPT-J (But here we are taking GPT-2). The system will accept user input in the form of questions and provide answers based on a provided context. The project involves setting up the environment, integrating the LLM, and designing prompt templates to enhance response quality.

**Task 1: Setting up the Environment**

**Steps Taken**

**1. Installing Required Packages:**

- To set up the Python environment with the necessary libraries, the following command was executed. This installs `langchain`, `transformers`, and other required packages:

python

!pip install langchain-community

!pip install transformers langchain

!pip install transformers

**2. importing Libraries:**

- After installing the packages, the following imports were made to utilize the models for text generation:

from transformers import pipeline

from transformers import AutoTokenizer, AutoModelForCausalLM

**3. Loading the GPT-2 Model:**

- The GPT-2 model was loaded using the `transformers` library. A generator pipeline is created for text generation tasks:

 generator = pipeline("text-generation", model="gpt2")

**4. Initializing Tokenizer and Model:**

- The pre-trained GPT-2 model and its tokenizer were initialized. This step is crucial for converting input text into tokens that the model can understand:

tokenizer = AutoTokenizer.from\_pretrained("gpt2")

model = AutoModelForCausalLM.from\_pretrained("gpt2")

**Environment Verification**

- After setting up, it is a good practice to verify that the installation was successful. This can be done by running a simple check:

  print("Packages installed successfully.")

**Task 2: LLM Integration with LangChain**

**System Implementation**

1. **Function to Generate Answers:**

- A function named `generate\_text` was defined to create a structured prompt, generate a response based on the context and question, and return the answer.

- The function includes steps to tokenize the input, generate the answer, and clean the output:

def generate\_text(context, question):

         input\_text = (

             f"Please answer the following question based on the context provided. "

             f"Keep the answer concise and relevant.\n\n"

             f"Context: {context}\n\n"

             f"Question: {question}\n"

             f"Answer:"

         )

         # Tokenize the input

         inputs = tokenizer(input\_text, return\_tensors="pt")

         # Generate the answer

         outputs = model.generate(

             \*\*inputs,

             max\_new\_tokens=100,  # Limit to a reasonable length

             do\_sample=True,

             temperature=0.7,

             top\_k=50,

             top\_p=0.9,

             num\_return\_sequences=1  # Generate only one answer

         )

         # Decode and clean up the output

         answer = tokenizer.decode(outputs[0], skip\_special\_tokens=True)

         # Extract only the answer from the generated text

         answer\_start\_index = answer.find("Answer:") + len("Answer: ")

         cleaned\_answer = answer[answer\_start\_index:].strip()

         return cleaned\_answer

**2. Defining Context and Questions:**

- The context provided was about APJ Abdul Kalam, including key details of his life and contributions. A list of relevant questions was also defined.

- Example of context and questions:

context = """APJ Abdul Kalam, often referred to as the "Missile Man of India," was a prominent scientist ..."""

     questions = [

         "Who is Dr. APJ Abdul Kalam?",

         "What role did Dr. Kalam play in India's missile development?",

         "What awards did Dr. Kalam receive during his lifetime?",

         "What was the main theme of Dr. Kalam's book 'India 2020'?",

         "How did Dr. Kalam inspire the youth of India?"

     ]

**3. Generating Answers:**

- A loop was created to iterate over the questions, using the `generate\_text` function to print the answers:

    for question in questions:

         answer = generate\_text(context, question)

         print(f"Question: {question}\nAnswer: {answer}\n")

**Example Output**

Question: Who is Dr. APJ Abdul Kalam?

Answer: APJ Abdul Kalam was a prominent scientist and the 11th President of India.

Question: What role did Dr. Kalam play in India's missile development?

Answer: He played a pivotal role in India's Integrated Guided Missile Development Program.

…….

**Task 3: Working with LangChain Prompt Templates**

**Custom Prompt Template**

1. **Creating a Custom Prompt:**

- A custom prompt template was defined using LangChain's `PromptTemplate` feature to enhance the response quality from the LLM. This template guides the model in formulating concise answers.

- Example of the custom prompt:

from langchain.prompts import PromptTemplate

     custom\_prompt = PromptTemplate(

         input\_variables=["context", "question"],

         template=(

             "You are a knowledgeable assistant. Based on the following context, please answer the question "

             "concise and accurately. Avoid repetition and focus on relevant information only.\n\n"

             "Context: {context}\n\n"

             "Question: {question}\n\n"

             "Answer:"

         )

     )

     ```

1. **Function with Memory:**

- A function was created to generate answers while storing conversation history using LangChain's memory capabilities. This allows the system to maintain context throughout the interaction.

- Example function:

     from langchain.memory import ConversationBufferMemory

     memory = ConversationBufferMemory()

     def generate\_answer\_with\_memory(context, question):

         # Save the current question in memory

         memory.save\_context({"input": question}, {"output": ""})

         # Retrieve the conversation history for contextual relevance

         conversation\_history = memory.load\_memory\_variables()

         # Format the prompt using the current context and conversation history

         prompt = custom\_prompt.format(context=context + "\n\n" + conversation\_history['input'], question=question)

         # Generate the answer (similar code as in the previous generate\_text function)

**3. Improving System Performance:**

- The custom prompt template improves the system's performance and accuracy by structuring the input. It provides clear instructions for the model, helping it generate answers that are not only relevant but also concise. By focusing on the context and the question, the likelihood of generating accurate responses increases significantly.

**Conclusion**

This documentation serves as a detailed guide to the steps taken to complete the assignment on building a question-answering system using LangChain and GPT-2. Each section provides insights into the processes and implementations that contribute to the final working system.