Integrating LangChain w/ Logic-LM

General Overview:

The objective of Task 8 is to "integrate **LangChain** with an existing logical inference engine to demonstrate how a natural language query can be answered using formal logic, as enabled by retrieval-augmented generation (RAG)." More specifically, this task is based on the **Logic-LM approach**, according to which "a large language model (LLM) helps with natural language to logical query translation and explanation trace generation, but the actual truth evaluation is performed by a symbolic logic engine (Prolog)."

The final result of the system is:

1. A query (assessed as true or false), and

2. A logical inference trace that explains how the conclusion was reached from facts and rules.

Step 1: Setup of Environment

To avoid system-level conflicts with other Python packages, a **virtual environment** referred to as **venv** was created within a directory referred to as task_8.

```
python3 -m venv .venv
source .venv/bin/activate
```

Once the virtual environment had been activated, the required libraries were installed:

```
pip install langchain langchain-community langchain-openai chromadb python-dotenv
```

In addition to this, the **SWI-Prolog** library was installed and verified using the following command:

```
swipl --version
```

This was necessary because the Prolog library is required for executing logical queries.

Step 2: Constructing the Knowledge Base

A simple **Prolog knowledge base** was developed (**simpsons_kb.pl**) that includes the following:

- **20 facts** (gender, parent, sibling relationships), and
- **8 logical rules** (mother, father, grandparent, grandmother, grandfather, ancestor, aunt, related).

This knowledge base represents the ground truth for all logical inference. Notice that facts were kept small and human-readable so that inference traces could be easily explained.

Step 3: Retrieval-Augmented Generation (RAG)

To fulfill this requirement of querying the knowledge base using RAG, a retriever was created in rag_store.py. The Prolog facts were transformed into text documents and embedded using OpenAI embeddings. These embeddings were then stored in a Chroma vector store.

At runtime:

- A natural language query is entered.
- The retriever retrieves the most relevant facts from the knowledge base.
- The retrieved facts serve as context to the LLM.

This way, the reasoning of the LLM is **informed by the knowledge base** rather than relying on memory or hallucinations.

Step 4: LangChain Inference Chain (Logic-LM)

The main logic is implemented in chains.py with the help of LangChain. The inference pipeline is based on the Logic-LM approach:

1. The **natural language query** is entered by the user.
2. The **RAG context** (relevant facts) is fetched from the vector database.
3. The **LLM translates** the natural language query into a **Prolog goal** (e.g., mother(marge, bart)).

4. The **Prolog engine evaluates** the goal and returns a true/false answer (and bindings, if applicable).

5. The **LLM constructs a logical inference trace**, indicating which facts and rules were used.

6. A final **answer** (TRUE or FALSE) is returned.

The whole pipeline is managed by LangChain, and the logical correctness is ensured by Prolog.

Step 5: Running the Overall System

All parts are connected in the main.py file. Before executing the program, the Open AI API key is loaded securely using a .env file with the python-dotenv library.

The system can be run with a single command:

```
python main.py
```

This command:

- Builds the RAG retriever,
- Runs multiple example queries,
- Prints the retrieved context,
- Shows the generated Prolog goal,
- Shows the Prolog execution result,
- Shows a step-by-step inference trace,
- Shows a final true/false verdict.

Step 6: Demonstrating and Meeting the Task Requirements

The output of the command “python main.py” is already showing that all the requirements of Task 8 are being met since:

- The application of RAG is demonstrated in the section “RAG CONTEXT” since true facts are being extracted from the knowledge base.

- The application of LangChain is demonstrated since the LLM is translating natural language questions into Prolog goals correctly.

- The results of the logical inferences are demonstrated since the result of the logical inferences can be seen clearly.

Screenshots of the code's execution are meant to show:

1. The RAG context extraction,
2. The Prolog goal execution,
3. The inference trace and true/false answer.

Conclusion:

This task is successfully implemented to show how the LangChain framework can be coupled with a symbolic engine for logical inference using the Logic-LM approach. The system combines the capabilities of RAG, translation and explanation using LLM, and reasoning using Prolog to generate grounded, explainable, and verifiable logical conclusions. This is a great example of the power of neuro-symbolic hybrid models, where LLMs can be used to support reasoning without substituting logic.