LangChain Wikipedia Agent: Solution Architecture and Implementation Report

1. Solution Architecture

The implemented solution is an intelligent question-answering agent built using LangChain and Hugging Face's TinyLlama model. The architecture consists of the following key components:

- **LLM (Language Model):** TinyLlama-1.1B-Chat-v1.0 is used for natural language understanding and response generation.
- **LangChain Framework:** Provides tools to orchestrate interactions between the LLM and external APIs (Wikipedia).
- **Wikipedia API Integration:** LangChain's WikipediaAPIWrapper enables the agent to retrieve factual information when needed.
- **Agent with ReAct Framework:** The agent follows the Reasoning + Acting (ReAct) paradigm, deciding when to retrieve information from Wikipedia based on the query.
- **Agent Executor:** Manages interactions between the user, LLM, and Wikipedia API while handling errors and limiting iterations.

2. Use of LangChain and LLM

LangChain acts as the framework to seamlessly integrate the LLM with external tools. The LLM is wrapped using HuggingFacePipeline, which enables direct inference from the TinyLlama-1.1B-Chat-v1.0 model running locally on CPU. The PromptTemplate structures agent responses using a ReAct-style prompting approach, ensuring a logical reasoning process before deciding whether to use Wikipedia.

3. Integration with External Tools

The Wikipedia API is accessed using LangChain's WikipediaAPIWrapper, which fetches up to two search results with a maximum of 4000 characters per document. The integration works as follows:

1. The agent receives a user query.
2. It decides whether to answer using the LLM's internal knowledge or fetch external data.

3. If additional factual support is required, it invokes the WikipediaQueryRun tool.
4. The retrieved Wikipedia data is processed and incorporated into the final response.

4. Challenges and Solutions

Challenge 1: API Integration Issues

- Solution: Initially, the system did not correctly support API calls due to configuration errors. After debugging, adjustments were made to properly integrate LangChain's WikipediaAPIWrapper with the agent.

Challenge 2: Selecting the Right LLM

- Solution: Originally, OpenAI's GPT models were considered, but due to constraints, an open-source alternative (TinyLlama-1.1B-Chat-v1.0) was selected for local execution. This choice ensured accessibility and performance without external dependencies.

Challenge 3: Errors in API Processing

- Solution: The system encountered errors in API response parsing. Implementing handle_parsing_errors=True and wrapping API calls in try-except blocks significantly improved stability.

Challenge 4: Finding Reliable Learning Resources

- Solution: To resolve technical challenges, references from YouTube tutorials and online documentation were used to refine the implementation and enhance system stability.

5. Conclusion

This solution effectively combines a local LLM with Wikipedia's API, using LangChain to orchestrate the decision-making process. By leveraging the ReAct framework, the agent can dynamically decide when external knowledge is necessary, enhancing response accuracy while minimizing unnecessary API calls. Future improvements could include adding confidence thresholds to refine when Wikipedia is queried and integrating additional external knowledge sources.

CODE

```
import os

from langchain_community.llms import HuggingFacePipeline
from langchain.agents import AgentExecutor, create_react_agent
from langchain.prompts import PromptTemplate
from langchain_community.tools import WikipediaQueryRun
from langchain_community.utilities import WikipediaAPIWrapper
from transformers import pipeline

# Set up a local pipeline using a small model that can run on CPU
llm_pipeline = pipeline(
    "text-generation",
    model="TinyLlama/TinyLlama-1.1B-Chat-v1.0",  # A small model that
can run on CPU
    max_new_tokens=512,
    temperature=0.7,
)

# Create a LangChain wrapper around the pipeline
llm = HuggingFacePipeline(pipeline=llm_pipeline)

# Set up the Wikipedia tool
wikipedia = WikipediaAPIWrapper(
    top_k_results=2,
    doc_content_chars_max=4000
)
wikipedia_tool = WikipediaQueryRun(api_wrapper=wikipedia)

# Define the prompt template for our agent
prompt_template = """You are a helpful assistant that answers user
questions using your knowledge and Wikipedia when necessary.
You have access to the following tools:

{tools}

Use the following format:

Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [{tool_names}]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: I now know the final answer
Final Answer: the final answer to the original user question
```

```
Begin!

Question: {input}
{agent_scratchpad}
"""

prompt = PromptTemplate.from_template(prompt_template)

# Create the agent
agent = create_react_agent(
    llm=llm,
    tools=[wikipedia_tool],
    prompt=prompt
)

# Create the agent executor
agent_executor = AgentExecutor(
    agent=agent,
    tools=[wikipedia_tool],
    verbose=True,
    handle_parsing_errors=True,
    max_iterations=3
)

def process_query(user_query):
    """Process a natural language query through the agent."""
    try:
        result = agent_executor.invoke({"input": user_query})
        return result["output"]
    except Exception as e:
        return f"Error processing your query: {str(e)}"

# Example usage
if __name__ == "__main__":
    print("Welcome to the LangChain Wikipedia Agent!")
    print("Ask any question, and I'll use my knowledge and Wikipedia to
answer.")
    print("Type 'exit' to quit.\n")

    while True:
        user_input = input("\nYour question: ")
        if user_input.lower() == "exit":
            print("Goodbye!")
            break

        print("\nProcessing your query...")
        response = process_query(user_input)
        print(f"\nResponse: {response}")
```