assignment5

September 9, 2024

1 Getting started with Generative-AI

Submitted By: Jenish Twayana

Submitted Date: 9th September, 2024

1.1 Exercise 5: Realm of Router Chains

You are tasked with building a multi-function system using LangChain that handles three different types of user inputs:

1. Weather Inquiries: Users ask about the current weather or weather forecasts.

2. Job Queries: Users request queries on job openings and things

Objective:

1. Create a Router Chain that correctly routes these different types of inputs to the appropriate sub-chains. Implement each sub-chain and integrate them into the Router Chain.
2. Create a diagram illustrating how you connected the chains. You can hand-draw this diagram.

Specific Instructions:

Weather Inquiries
To handle weather inquiries, utilize custom agents/tools designed for weather information retrieval. Users can request the weather for any city.
Job Queries
For job-related queries, employ the built-in tools available on langchain.
Hint: You can use the "google-jobs" tool to find job listings and relevant information.
Joke Requests
For joke requests, apply prompt engineering techniques, principles or guidelines to generate jokes directly.

1.1.1 Import the necessary packages and libraries

[1496]:
```
import os
import requests
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain.agents.format_scratchpad.openai_tools import␣
 ↪format_to_openai_tool_messages
```

```
from langchain.agents.output_parsers.openai_tools import␣
 ↪OpenAIToolsAgentOutputParser
from langchain.agents import AgentExecutor
from langchain.agents import tool
from langchain_community.tools.google_jobs import GoogleJobsQueryRun
from langchain_community.utilities.google_jobs import GoogleJobsAPIWrapper
from langchain.schema.runnable import RunnableMap
from langchain.schema import StrOutputParser
```

1.1.2 Set up the environment variables for LangChain Tracing

It logs the tracing data in the LangSmith web interface.

[1497]:
```
os.environ["LANGCHAIN_TRACING_V2"]="true" # enables the tracing
os.environ["LANGCHAIN_ENDPOINT"]="https://api.smith.langchain.com"
os.environ["LANGCHAIN_API_KEY"]=os.getenv("LANGCHAIN_API_KEY")
os.environ["LANGCHAIN_PROJECT"]="assignment-5" #project name in the LangSmith␣
 ↪platform
```

1.1.3 Initialise the Chat Model

[1498]:
```
llm = ChatOpenAI(model="gpt-3.5-turbo", temperature=0)
```

1.2 1. Creating the Weather Agent

1.2.1 Custom Tool for fetching Weather Data

[1499]:
```
@tool
def get_weather_data(city: str) -> str:
    """Calls the Weather API and return the weather data
    Args:
        city: str
    Returns:
        str
    """
    url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={os.
 ↪getenv('WEATHER_API_KEY')}&units=metric"
    response = requests.get(url)
    return str(response.json())
```

[1500]:
```
tools = [ get_weather_data ]
```

[1501]:
```
prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            """
```

```
            You are very powerful weather data assistant equipped with multiple␣
 ↪tools.
              Here is the detailed instruction:
              1. Call the Weather API to get the weather data of the city.
              2. If the Weather API returns valid response with weather data␣
 ↪then, return the weather data in given output format.
              3. If the Weather API returns no weather data then, return the␣
 ↪response saying the weather data is not available.
              The desired output format for different scenarios are given below:
              Here is the weather data of the city <city_name>:
              <weather_data_in_bullet_form (dash separated)>
              """,

        ),
        ("user", "{input}"),
        MessagesPlaceholder(variable_name="agent_scratchpad"), # sequence of␣
 ↪messages that contain the previous agent tool invocations and the␣
 ↪corresponding tool outputs
    ]
)
```

1.2.2 Bind the Custom tools to the Chat Model

[1502]:
```
llm_with_tools = llm.bind_tools(tools)
```

1.2.3 Initialise the Agent

[1503]:
```
agent = (
    {
        "input": lambda x: x["input"],
        "agent_scratchpad": lambda x: format_to_openai_tool_messages(
            x["intermediate_steps"]
        ),
    }
    | prompt
    | llm_with_tools
    | OpenAIToolsAgentOutputParser()
)
```

[1504]:
```
weather_agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True)
```

1.3 2. Creating Prompt chain for Jokes

[1505]:
```
joke_prompt_template = """You are a funny assistant.
 You will provide one short and funny jokes on the given topic.
If the topic is not specified then, you can give your own jokes.
{input}
"""
```

[1506]:
```
joke_prompt = ChatPromptTemplate.from_template(joke_prompt_template)
```

[1507]:
```
joke_prompt_chain = joke_prompt | llm | StrOutputParser()
```

1.4 3. Creating Tool chain for Google Jobs

[1508]:
```
google_jobs_tool =␣
 ↪GoogleJobsQueryRun(api_wrapper=GoogleJobsAPIWrapper(serp_api_key=os.
 ↪getenv('SERPAPI_API_KEY')), verbose=True)
```

[1509]:
```
google_jobs_chain = (lambda x: x['input']) | google_jobs_tool
```

1.4.1 System prompt template

[1510]:
```
system_prompt_template = """
Determine the category of the following text into one of these three categories
and respond with the category only.
1. Weather
2. Joke
3. Job
If the question is not related to one of the categories,
respond with "Other".
Here is an example:
###
Question: What is the weather in XYZ city?
Weather
###

Question: {question}"""
```

[1511]:
```
system_prompt = ChatPromptTemplate.from_template(system_prompt_template)
```

1.4.2 Base Prompt chain for “Other” categories

[1512]:
```
base_prompt_template = """
Respond politely saying you can not answer the current question and only to ask␣
 ↪question within the categories of Weather, Joke or Job.
"""
```

[1513]:
```
base_prompt = ChatPromptTemplate.from_template(base_prompt_template)
```

[1514]:
```
base_chain = base_prompt | llm | StrOutputParser()
```

1.4.3 Function for selecting appropriate chain

[1515]:
```
def select_chain(output):
    if output["action"] == "Weather":
        return weather_agent_executor
    elif output["action"] == "Job":
        return google_jobs_chain
    elif output["action"] == "Joke":
        return joke_prompt | llm | StrOutputParser()
    else:
        return base_prompt | llm | StrOutputParser()
```

1.4.4 Router Chain

[1516]:
```
router_chain = system_prompt | llm | StrOutputParser()
```

[1517]:
```
chain = RunnableMap({
    "action": router_chain,
    "input": lambda x: x["question"]
})| select_chain

output = chain.invoke({"question":"Tell me a joke and weather in kathmandu"})
print(output if isinstance(output, str) else output["output"])
```

```
I'm sorry, but I can only answer questions within the categories of Weather,
Joke, or Job. Do you have a question in one of those categories that I can help
with?
```

2 Second Approach using MULTI_PROMPT_ROUTER_TEMPLATE

2.0.1 Almost similar to the above approach

[1518]:
```
from langchain_core.runnables import RunnableLambda
from langchain.chains.router.multi_prompt_prompt import␣
 ↪MULTI_PROMPT_ROUTER_TEMPLATE
```

[1519]:
```
chains_infos= [
    {
        'name': 'weather',
        'description': 'Provides the current weather data for a given city name␣
 ↪using the weather agent.',
    },
```

```
    {
        'name': 'job',
        'description': 'Fetches job listings using the Google Jobs tool within␣
 ↪Langchain based on given criteria.',
    },
    {
        'name': 'joke',
        'description': 'Generates a joke using a simple prompt chain for humor.
 ↪',
    }
]
```

2.0.2 Add Weather Agent, Google Jobs Tool chain and Joke chain in the destination_chain

[1520]:
```
destination_chains = {}
destination_chains['weather'] = weather_agent_executor
destination_chains['job'] = google_jobs_chain
destination_chains['joke'] = joke_prompt_chain
```

[1521]:
```
from operator import itemgetter
from typing import Literal
from typing_extensions import TypedDict
class RouteQuery(TypedDict):
    """Route query to destination."""
    destination: Literal["weather", "job", "joke"]
```

[ ]:
```
destinations = [f"{chain['name']}: {chain['description']}" for chain in␣
 ↪chains_infos]
destinations_str = "\n".join(destinations)
```

[1522]:
```
router_template = MULTI_PROMPT_ROUTER_TEMPLATE.format(
    destinations=destinations_str
)
router_prompt = ChatPromptTemplate.from_template(template=router_template)
```

2.0.3 Router Chain

[ ]:
```
router_chain = router_prompt | llm.with_structured_output(RouteQuery) |␣
 ↪itemgetter("destination")
```

[1523]:
```
chain = {
    "destination": router_chain,
    "question": lambda x: x["question"],
    "input": lambda x: x["question"],
} | RunnableLambda(
```

```
lambda x: destination_chains['weather'] if x["destination"] == "weather"
         else destination_chains['joke'] if x["destination"] == "joke"
         else destination_chains['job'] if x["destination"] == "job"
         else base_chain
)
```

[1525]:
```
print(chain.invoke({"question":"tell me the joke"}))
```

```
Why did the scarecrow win an award?
Because he was outstanding in his field!
```