Building AI Agents with LangChain

Learning Objectives

By the end of this lesson, you will be able to:

- Set up a LangChain environment to create AI agents.
- Build and configure an AI agent to interact with external data sources.
- Modify and extend an AI agent's behavior using reasoning and action-based decision-making.
- Troubleshoot and refine an AI agent using feedback loops.

Introduction

LangChain is an open-source framework that enables AI agents to interact dynamically with data, APIs, and external tools. In this lesson, you'll build an AI agent capable of retrieving and processing information, making decisions, and learning from interactions.

Setup: Environment & Dependencies

Step 1: Install LangChain and Required Libraries

Ensure you have the following dependencies installed:

Copy

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

Step 2: Load Your API Key

If using OpenAI, create a `.env` file and store your key securely:

Copy

```
OPENAI_API_KEY="your-api-key-here"
```

Then load it in Python:

Copy

```
import os
from dotenv import load_dotenv

load_dotenv()
api_key = os.getenv("OPENAI_API_KEY")
```

Step 1: Create a Basic LangChain Agent

This simple LangChain agent takes user input and interacts with a language model.

Copy

```
from langchain.chat_models import ChatOpenAI
from langchain.agents import initialize_agent, AgentType
from langchain.tools import Tool

# Define an AI agent using OpenAI's model
llm = ChatOpenAI(model="gpt-4", temperature=0.5)
```

```
# Define a basic agent with tool capabilities
agent = initialize_agent(
    tools=[],
    llm=llm,
    agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
    verbose=True
)

# Run the agent
response = agent.run("What is the capital of France?")
print(response)
```

Step 2: Add a Custom Tool to Your Agent

LangChain allows agents to use external tools (e.g., APIs, databases). Here's how to add a custom search tool:

Copy

```
def search_tool(query: str):
    """Simulates an external search tool"""
    knowledge_base = {
        "capital of France": "Paris",
        "largest planet": "Jupiter"
    }
    return knowledge_base.get(query.lower(), "I don't know")

# Wrap the function into a LangChain Tool
custom_tool = Tool(
    name="SimpleSearch",
    func=search_tool,
    description="Searches a local knowledge base."
)

# Initialize an agent with the tool
agent_with_tool = initialize_agent(
    tools=[custom_tool],
    llm=llm,
    agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
    verbose=True
)
```

```
# Test the agent with a query
response = agent_with_tool.run("What is the capital of France?")
print(response)
```

Step 3: Implement a Feedback Loop for Self-Correction

AI agents can improve over time by refining their answers based on user feedback.

Copy

```
user_feedback = {}

def feedback_tool(query: str, feedback: str):
    """Stores user feedback for agent responses."""
    user_feedback[query] = feedback
    return f"Feedback stored: {feedback}"

# Wrap the function into a LangChain Tool
feedback_storage = Tool(
    name="FeedbackCollector",
    func=feedback_tool,
    description="Collects user feedback on agent responses."
)

# Add feedback collection to the agent
agent_with_feedback = initialize_agent(
    tools=[custom_tool, feedback_storage],
    llm=llm,
    agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
    verbose=True
)

# Example usage
response = agent_with_feedback.run("What is the capital of France?")
print(response)

# User provides feedback
```

```
feedback_response = agent_with_feedback.run("FeedbackCollector: The answer should
print(feedback_response)
```

Coding Challenge: Modify & Expand Your Agent

Goal: Modify the agent to handle multiple data sources and improve reasoning accuracy.

Tasks:

1. Add another external tool (e.g., a real-time API).
2. Modify the feedback system to store multiple feedback entries per query.
3. Implement a basic reflection mechanism where the agent adjusts based on past responses.

< Previous

Next >