Data Connections, Chains, and Agents - 2

One should look for what is and not what he thinks should be. (Albert Einstein)

Module completion checklist

Objective	Complete
Explore and use different chain types in LangChain	
Outline LangChain agents and how to use them	

Chains

- In LangChain, **chains** are **modular, reusable components** that facilitate the creation of complex workflows by combining language models with other tools and data sources
- Some key types of chain are:
  - LLM chain
  - Sequential chain
  - Router chain
  - Utility chain
- Visit ***python.langchain.com (link)*** to learn more about chains

LLM chain

- The most fundamental chain type, the **LLM chain**:
  - takes an input and applies a format using a prompt template
  - passes it to a language model
  - parses the output
- In short, `PromptTemplate + LLM = LLMChain`

LLM chain (cont'd)

- Try running a simple chain using the LLaMA model in `ctransformers`

```
!pip install ctransformers[gptq]
#if prompts for Restart Session , Do Restart and rerun from this code chunk again

from langchain_community.llms import CTransformers # Import necessary classes from langchain
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain

# Initialize the Llama model using C Transformers with the specific Llama-2-7B-Chat model
llm = CTransformers(model="TheBloke/Llama-2-7B-Chat-GGUF", model_type="llama")
# Create a prompt template for generating company names based on a product
prompt = PromptTemplate(input_variables=["product"],
    template="What is a good name for a company that specializes in {product}?")
```

```
# Create an LLMChain object with the specified language model and prompt
chain = LLMChain(llm=llm, prompt=prompt, verbose=True)

# Run the chain with an input and print the output
print(chain.run("gaming laptop"))
#While executing, Outputs may vary from those in slides due to model hallucination and the
stochastic nature of language model predictions based on dynamic input specificity. This is more
pronounced in locally loaded models compared to API-based models
```

```
.. ... .. . .. ... .. . .. ... ..
"PowerPlayTech"
```

Sequential chain

- A **sequential chain** executes multiple chains in a specific order
- It supports multiple inputs/outputs
- `SimpleSequentialChain` allows for a single input to undergo a series of coherent transformations, resulting in a refined output

Diagram of a simple sequential chain

Sequential chain (cont'd)

- Now try combining two chains into a sequential chain flow

```
from langchain.chains import LLMChain
from langchain.prompts import PromptTemplate
from langchain.prompts import ChatPromptTemplate
from langchain.chains import SimpleSequentialChain

# This is an LLMChain to write the first chain.
first_prompt = ChatPromptTemplate.from_template(
    "What is the best name to describe a company that makes {product}?")
chain_one = LLMChain(llm=llm, prompt=first_prompt)
```

Sequential chain (cont'd)

```
# This is an LLMChain to write the second chain
second_prompt = ChatPromptTemplate.from_template(
    "Write a 20 words description for the following company:{company_name}")
chain_two = LLMChain(llm=llm, prompt=second_prompt)
# LLM is already defined as Llama-2-7B-Chat-GGUF in previous slides

# Combining two chains in a simple sequential flow
overall_simple_chain = SimpleSequentialChain(chains=[chain_one, chain_two],verbose=True)
overall_simple_chain.run("gaming laptop")
#While executing, Outputs may vary from those in slides due to model hallucination and the
stochastic nature of language model predictions based on dynamic input specificity. This is more
pronounced in locally loaded models compared to API-based models
```

```
"EliteTech Gaming Laptops"

"Unleash your gaming potential with EliteTech's high-performance laptops designed for the
ultimate gaming experience. Elevate your gameplay today."
```

Router chain

- A **router chain** directs, or routes, inputs to the most suitable chain from a set of predefined options
- It is ideal for handling diverse inputs requiring specialized processing
- Key features of router chains include:
  - **Dynamic selection**: chooses the appropriate chain based on input criteria
  - **Versatility**: manages a variety of inputs efficiently
  - **Customization**: easily expandable with new chains for different tasks

Router chain exploration

- Experiment with router chains by defining personas with subject matter-specific expertise

```
# Importing necessary modules and classes
from langchain.chains.router import MultiPromptChain
from langchain.chains import ConversationChain
from langchain.chains.llm import LLMChain
from langchain.prompts import PromptTemplate
from langchain.chains.router.llm_router import LLMRouterChain, RouterOutputParser
from langchain.chains.router.multi_prompt_prompt import MULTI_PROMPT_ROUTER_TEMPLATE
```

```
# Defining personas for router decision-making
math_teacher = """Your name is Helen, you are a Math teacher...{input}"""

spanish_teacher = """Your name is Thomas, you are a Spanish teacher...{input}"""

calculus_teacher = """Your name is Godfrey, you are a Calculus teacher...{input}"""

# Setting up prompts for each persona
prompt_infos = [
    {"name": "Math Teacher", "description": "Good for answering questions about Math",
"prompt_template": math_teacher},
    {"name": "Spanish Teacher", "description": "Good for answering questions about Spanish",
"prompt_template": spanish_teacher},
    {"name": "Calculus Teacher", "description": "Good for answering questions about Calculus",
"prompt_template": calculus_teacher},]
```

Router chain exploration (cont'd)

```
# Mapping inputs to specific teacher chains
destination_chains = {}
for prompt_info in prompt_infos:
    prompt = PromptTemplate(template=prompt_info["prompt_template"], input_variables=["input"])
    chain = LLMChain(llm=llm, prompt=prompt)
    destination_chains[prompt_info["name"]] = chain

# Default chain for general conversations
default_chain = ConversationChain(llm=llm, output_key="text")

# Preparing descriptions for router options
destinations = [f"{p['name']}: {p['description']}" for p in prompt_infos]
destinations_str = "\n".join(destinations)
```

```
# Setting up router chain template
router_template = MULTI_PROMPT_ROUTER_TEMPLATE.format(destinations=destinations_str)
router_prompt = PromptTemplate(template=router_template, input_variables=["input"],
output_parser=RouterOutputParser())

# Creating the main router chain
router_chain = LLMRouterChain.from_llm(llm, router_prompt)

# Combining all chains into MultiPromptChain
chain = MultiPromptChain(router_chain=router_chain, destination_chains=destination_chains,
default_chain=default_chain, verbose=True)
```

Router chain exploration (cont'd)

```
# Testing the router chain with various inputs
print(chain.run("What is the meaning of average?"))
print(chain.run("What is the derivative of x dx"))
print(chain.run("Translate 'Hello world' to Spanish"))
#During execution, outputs may vary from those in slides due to model hallucination and the
stochastic nature of language model predictions based on dynamic input specificity. This is more
pronounced in locally loaded models compared to API-based models
```

```
> Entering new MultiPromptChain chain...
Math Teacher: {'input': 'What is the meaning of average?'}
Average refers to the sum of a set of numbers divided by the total number of numbers in that
set. .....


> Entering new MultiPromptChain chain...
Calculus Teacher: {'input': 'What is the derivative of x dx'}
The derivative of x dx is simply 1. This is because the derivative of x is 1 and the
derivative of dx is 0.....

> Entering new MultiPromptChain chain...
Spanish Teacher: {'input': "Translate 'Hello world' to Spanish"}
"Hola mundo"
```

Utility chain

- **Utility chains** help accomplish specialized tasks, including:
  - math computations
  - SQL commands
  - API calls
  - Bash commands
- They combine a large language model (LLM) with specific utilities in LangChain
- They include chains like MathChain, SQLChain, APIChain, and BashChain
- Key features of utility chains:
  - **Targeted functionality**: each chain is designed for specific operations
  - **Integration**: seamlessly integrate with LLMs for enhanced processing
  - **Customization**: tailor chains to unique application requirements
- Learn about even more types of chains by visiting ***python.langchain.com (link)***

Module completion checklist

Objective	Complete
Explore and use different chain types in LangChain	✔
Outline LangChain agents and how to use them	

LangChain agents

- LangChain **agents** are powerful components that bring flexibility and adaptivity to workflows
- They are an advancement from the more rigid “chain” system in LangChain, allowing users to create intelligent agents
- Key traits of these intelligent LangChain agents include:
  - **Dynamic decision-making**: Agents can choose their actions based on the context and user input, not just follow a pre-defined script
  - **External tool access**: Agents can be connected to APIs, databases, and other tools to gather information and complete tasks
  - **Memory capabilities**: Agents can store information from previous interactions, providing a sense of context and continuity
  - **Learning potential**: Advanced agents can adapt and improve their behavior based on feedback and experience

LangChain agents (cont'd)

- Use a LangChain agent to compose a 100-word essay on an input topic

```
!pip install -U duckduckgo-search #DuckDuckGo search library.
!pip install arxiv #library for accessing Arxiv papers.
!pip install wikipedia #Wikipedia data access library.

# Import required modules from LangChain
from langchain.tools import Tool, DuckDuckGoSearchRun, ArxivQueryRun, WikipediaQueryRun
from langchain.utilities import WikipediaAPIWrapper
from langchain.agents import initialize_agent, AgentType
from langchain.chains import LLMChain
from langchain.prompts import PromptTemplate

# Initialize the tools for the agent
search = DuckDuckGoSearchRun()
arxiv = ArxivQueryRun()
wiki = WikipediaQueryRun(api_wrapper=WikipediaAPIWrapper())
```

```
# Setup the essay generation tool

prompt_template = "Write an essay for the topic provided by the user with the help of following
content: {content}"
essay = LLMChain(
    llm=llm,
    prompt=PromptTemplate.from_template(prompt_template)
)
```

LangChain agents (cont'd)

```
# Define the tools the agent will use
tools = [
    Tool(
        name="Search",
        func=search.run,
        description="useful for when you need to answer questions about current events."),
    Tool(
        name="Arxiv",
        func=arxiv.run,
        description="useful when you need an answer about research papers from the arXiv
repository"),
    Tool(
        name="Wikipedia",
        func=wiki.run,
        description="useful when you need an answer about encyclopedic general knowledge"),
    Tool.from_function(
        func=essay.run,
        name="Essay",
        description="useful when you need to write an essay"),
]
```

```
# Initialize the agent with defined tools
agent = initialize_agent(tools, llm, agent=AgentType.OPENAI_FUNCTIONS, verbose=True)

# Run the agent to write an essay on given topic
prompt = "Write an essay in 100 words for the topic {input}, use the tools to retrieve the
necessary information"
input_topic = "Essay on Global Warming - Causes and Solutions"
```

LangChain agents (cont'd)

```
print(agent.run(prompt.format(input=input_topic)))
```

```
> Entering new AgentExecutor chain...
Invoking: `Essay` with `Essay on Global Warming - Causes and Solutions`

Global warming is a pressing issue that has garnered significant attention in recent years.
It refers to the long-term increase in Earth's average surface temperature due to human
activities, primarily the emission of greenhouse gases. This essay will explore the causes
of global warming and propose potential solutions to mitigate its effects.
. .. ... . .. ... . .. ... . .. ... . .. ... . .. ... . .. ... . .. ... . .. ... . .. ...
> Finished chain.
Global warming is a significant problem caused by human activities, primarily the burning of
fossil fuels, deforestation, and methane emissions. To address this issue, we need to
transition to renewable energy sources, such as solar and wind power, and reduce our
reliance on fossil fuels. Additionally, efforts to reduce deforestation and promote
reforestation are crucial in restoring the planet's capacity to absorb carbon dioxide. We
also need to implement better agricultural practices to reduce methane emissions from
livestock and organic waste. Finally, raising awareness and educating the public about the
importance of reducing greenhouse gas emissions is essential. By taking these steps, we can
work towards mitigating the effects of global warming and creating a sustainable future.
```

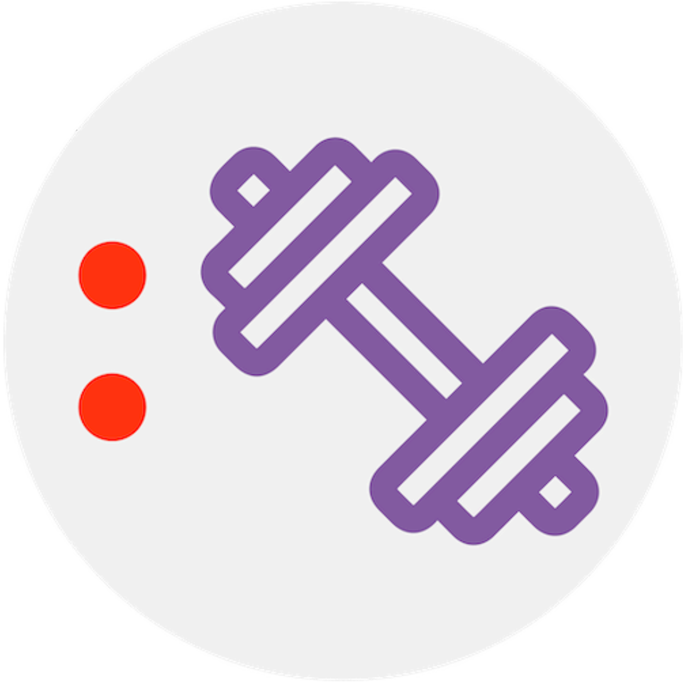
LangChain agent uses

- Additional use cases of LangChain agents include:
  - **Adaptive automation**: Automate workflows with agents that can handle unexpected situations and adjust their actions based on real-time data
  - **Interactive chatbots**: Build dynamic chatbots that understand user intent and respond accordingly
- Agents can be customized for specific tasks and domains
- They can be integrated into larger applications for a more interactive and adaptable user experience
- LangChain also provides various tools and utilities to simplify agent development

Knowledge check

Link: ***Click here to complete the knowledge check***

Exercise



You are now ready to try Tasks 4-5 in the Exercise for this topic

Module completion checklist

Objective	Complete
Explore and use different chain types in LangChain	✔
Outline LangChain agents and how to use them	✔

Data connections, Chains, and Agents: Topic summary

In this part of the course, we have covered:

- LangChain data connections processes and components
- Different chain types available in LangChain
- How to use LangChain agents

Congratulations on completing this module!