**LangChain là gì?**

LangChain là một framework được viết bằng Python và JavaScript, nó cung cấp các công cụ để thao tác và xây dụng ứng dụng dựa trên LLMs. Bạn có thể xem hướng dẫn cài đặt LangChain tại [đây](https://python.langchain.com/docs/get_started/installation). LangChain hướng đến việc giải quyết các vấn đề khi làm việc với LLMs vì vậy dưới đây cũng chính là những core modules của LangChain.

**Model I/O**

Có rất nhiều LLMs để bạn sử dụng (OpenAI, Hugging Face…), LangChain cung cấp interface để bạn có thể tương tác với các model khác nhau mà không gặp bất kỳ khó khăn nào.

Bạn có thể sử dụng model LLMs từ các nhà cung cấp như OpenAI với API key:

from langchain\_openai import ChatOpenAI

from langchain\_openai import OpenAI

llm = OpenAI(openai\_api\_key="...")

chat\_model = ChatOpenAI()

Bạn cũng có thể tương tác trực tiếp với các model chạy dưới local qua việc sử dụng [Ollama](https://ollama.ai/). Đây là một dự án mã nguồn mở, được sử dụng để tương tác với LLMs mà không cần phải kết nối với các nhà cung cấp dịch vụ bên ngoài. Chúng ta sẽ tìm hiểu về Ollama trong một bài viết khác. Dưới đây là một ví dụ về việc sử dụng Ollama trong LangChain:

from langchain\_community.llms import Ollama

from langchain\_community.chat\_models import ChatOllama

llm = Ollama(model="llama2")

chat\_model = ChatOllama()

Với các đối tượng llm và chat\_model, bạn có thể bắt đầu tương tác với mô hình LLMs.

**Prompts**

LangChain cung cấp prompt template giúp bạn cấu trúc đầu vào cho LLMs một cách hiệu quả. Bạn có thể tạo ra một prompt động, chứa các tham số thay đổi tuỳ thuộc vào mục đích sử dụng.

prompt = PromptTemplate(

template="Tell me a joke in {language}", input\_variables=["language"]

)

print(prompt.format(language="spanish"))

'Tell me a joke in spanish'

**Output Parsers**

Đầu ra của một LLM là text, trong nhiều trường hợp bạn muốn kết quả trả về là một định dạng khác như JSON, CSV… output parsers sẽ giúp bạn thực hiện điều này. Hãy cùng xem qua ví dụ dưới đây:

template = "Generate a list of 5 {text}.\n\n{format\_instructions}"

chat\_prompt = ChatPromptTemplate.from\_template(template)

chat\_prompt = chat\_prompt.partial(format\_instructions=output\_parser.get\_format\_instructions())

chain = chat\_prompt | chat\_model | output\_parser

chain.invoke({"text": "colors"})

['red', 'blue', 'green', 'yellow', 'orange']

**Retrieval**

Thông thường, LLMs sẽ bị giới hạn bởi tập dữ liệu tại thời điểm huấn luyện. Giống như Chat-GPT sẽ không thể trả lời các câu hỏi liên quan đến những sự kiện diễn ra sau năm 2021. Nhiều trường hợp, bạn cũng cần model hiểu thêm những tài liệu khác mà bạn yêu cầu. RAG (Retrieval Augmented Generation) sinh ra để giải quyết các vấn đề này.

LangChain cung cấp các modules giúp bạn xây dựng một ứng dụng RAG hoàn chỉnh. Từ việc nhúng cho tới việc thao tác với các dữ liệu được thêm từ bên ngoài.

**Document Loaders**

Đây là module hỗ trợ việc load tài liệu từ các nguồn như Github, S3… cùng với nhiều định dạng khác nhau như .txt, .csv...:

from langchain\_community.document\_loaders import TextLoader

loader = TextLoader("./main.rb")

documents = loader.load()

[Document(page\_content='puts("Hello LangChain!")\n', metadata={'source': './main.rb'})]

**Text Splitting**

Dữ liệu sau khi load sẽ được xử lý để lấy ra các thông tin có ý nghĩa, sau đó được chia ra từng phần nhỏ trước khi chuyển đến bước tiếp theo.

from langchain.text\_splitter import Language

from langchain.text\_splitter import RecursiveCharacterTextSplitter

splitter = RecursiveCharacterTextSplitter.from\_language(

language=Language.RUBY, chunk\_size=2000, chunk\_overlap=200

)

documents = splitter.split\_documents(raw\_documents)

[Document(page\_content='puts("Hello LangChain!")', metadata={'source': './main.rb'})]

**Text Embedding Models**

Dữ liệu tiếp tục được chuyển đổi sang vector space và bạn cũng có thể cache lại chúng để tái sử dụng.

from langchain\_community.embeddings import OllamaEmbeddings

embeddings\_model = OllamaEmbeddings()

embeddings = embeddings\_model.embed\_documents(documents)

**Vector Stores**

Dữ liệu sau khi chuyển đổi sang vector có thể được lưu vào vector store. LangChain cung cấp một module để bạn thực hiện việc này:

from langchain\_community.vectorstores import Chroma

db = Chroma.from\_documents(documents, OllamaEmbeddings())

**Retrievers**

Bây giờ bạn đã có thể thao tác với dữ liệu ở trên thông qua retrievers.

from langchain.chains import RetrievalQA

from langchain.llms import Ollama

retriever = db.as\_retriever(search\_kwargs={"k": 1})

llm = Ollama(model="codellama:7b")

qa = RetrievalQA.from\_chain\_type(llm=llm, chain\_type="stuff", retriever=retriever)

qa.invoke("What's inside this document?")["result"]

'The text "Hello LangChain!" is a string of characters written in the Ruby programming language. It is not clear what "LangChain" refers to or what context it may be used in, so I cannot provide a helpful answer without more information.'

**Agents**

Với mỗi yêu cầu của người dùng, bạn cần thực hiện các bước khác nhau để trả về kết quả như mong muốn. Ví dụ với một câu hỏi đơn giản, người dùng chỉ cần một câu trả lời dạng text thông thường. Nhưng khi muốn hiển thị kết quả dạng bảng hoặc một yêu cầu export dữ liệu ra định dạng PDF, bạn sẽ cần thực hiện thêm các action khác để có được kết quả cuối cùng. LangChain Agents sẽ giúp bạn giải quyết vấn đề trên.

Mỗi một agent có thể coi là một tập hợp của nhiều tool, một tool sẽ bao gồm các thành phần chính:

* **Name**: Tên của tool
* **Description**: Là một mô tả ngắn gọn về mục đích sử dụng của tool
* **JSON schema**: Chứa thông tin đầu vào của tool
* **Function call**: Là nội dung chính sẽ được gọi khi chạy tool

Trong đó name, description và JSON schema là những thành phần quan trọng nhất, chúng được sử dụng ở tất cả các prompt.

**Built-In Tool**

Lang chain cung cấp rất nhiều built-in tool để bạn sử dụng. Dưới đây là một ví dụ:

from langchain\_community.tools import WikipediaQueryRun

from langchain\_community.utilities import WikipediaAPIWrapper

api\_wrapper = WikipediaAPIWrapper(top\_k\_results=1, doc\_content\_chars\_max=100)

tool = WikipediaQueryRun(api\_wrapper=api\_wrapper)

print("Name:", tool.name)

print("Description:", tool.description)

print("JSON schema:", tool.args)

Thông tin của tool:

Name: wikipedia

Description: A wrapper around Wikipedia. Useful for when you need to answer general questions about people, places, companies, facts, historical events, or other subjects. Input should be a search query.

JSON Schema: {'query': {'title': 'Query', 'type': 'string'}}

Bạn có thể sử dụng tool một cách đơn giản như sau:

tool.run({"query": "LangChain"})

'Page: LangChain\nSummary: LangChain is a framework designed to simplify the creation of applications '

Bằng việc sử dụng tool, bạn hoàn toàn có thể tương tác với dữ liệu bên ngoài và trên internet, qua đó sẽ phát huy được tối đa sức mạnh của model.

**Defining Custom Tools**

Không bị hạn chế bởi những built-in tool, LangChain cung cấp cho bạn cách để tự tạo tool phù hợp với bất kỳ mục đích nào. Để khai báo tool bạn cần sử dụng @tool decorator:

from langchain.tools import tool

@tool

def search(query: str) -> str:

"""Look up things online."""

return "LangChain"

@tool

def multiply(a: int, b: int) -> int:

"""Multiply two numbers."""

return a \* b

Sau đó bạn cần tạo agent từ những tool trên:

from langchain.agents import initialize\_agent, AgentType

from langchain.llms import Ollama

tools = [search, multiply]

agent = initialize\_agent(

tools,

Ollama(),

agent=AgentType.STRUCTURED\_CHAT\_ZERO\_SHOT\_REACT\_DESCRIPTION,

verbose=True,

)

Sau đó bạn có thể sử dụng:

agent.invoke("Multiply two numbers: 2 and 3")

> Entering new AgentExecutor chain...

Action: multiply(a: int, b: int) -> int

Action\_input: {"a": 2, "b": 3}

> Finished chain.

{'input': 'Multiply two numbers: 2 and 3', 'output': 'Action: multiply(a: int, b: int) -> int\nAction\_input: {"a": 2, "b": 3}\n'}

Thử với một input khác:

agent.invoke("Go online and search")

> Entering new AgentExecutor chain...

Action: search

Action\_input: {'query': {'title': 'Query', 'type': 'string'}}

> Finished chain.

{'input': 'Go online and search', 'output': "Action: search\nAction\_input: {'query': {'title': 'Query', 'type': 'string'}}\n\n"}

Agent đã hoạt động đúng như những gì chúng ta mong muốn.

**Conclusion**

LangChain là một framework rất mạnh mẽ giúp bạn dễ dàng tương tác và khai thác sức mạnh của LLMs. Bài viết cung cấp cho bạn những thông tin cơ bản về LangChain. Những phần sau chúng ta sẽ sử dụng LangChain để giải quyết các bài toán cụ thể hứa hẹn sẽ rất thú vị.