

# Ứng dụng LangGraph cơ bản với chatbot và công cụ

Mục tiêu của ứng dụng.



Sử dụng LangGraph quyết định phản hồi bằng chatbot hoặc sử dụng công cụ

Chúng ta sẽ xây dựng một ứng dụng cơ bản thực hiện 4 thao tác

Sử dụng chat messages làm **State**

Sử dụng chat model làm **Node**

Liên kết một công cụ với chat model

Thực hiện lệnh gọi công cụ trong **Node**

## I. Cài đặt

```
* cd project_name
* pyenv local 3.11.4
* poetry install
* poetry shell
* jupyter lab
```

tạo file .env

```
OPENAI_API_KEY=your_openai_api_key
* LANGCHAIN_TRACING_V2=true
* LANGCHAIN_ENDPOINT=https://api.smith.langchain.com
* LANGCHAIN_API_KEY=your_langchain_api_key
```

```
* LANGCHAIN_PROJECT=your_project_name
```

Bạn sẽ cần đăng ký tài khoản tại [smith.langchain.com](https://smith.langchain.com) để lấy API key.

Kết nối với file .env nằm trong cùng thư mục của notebook

```
#pip install python-dotenv
import os
from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv())
openai_api_key = os.environ["OPENAI_API_KEY"]
```

Cài LangChain

```
#!pip install langchain-openai

from langchain_openai import ChatOpenAI

chatModel35 = ChatOpenAI(model="gpt-3.5-turbo-0125")
chatModel4o = ChatOpenAI(model="gpt-4o")
```

## II. Xác định State schema

```
from typing_extensions import TypedDict
from langchain_core.messages import AnyMessage
from typing import Annotated
from langgraph.graph.message import add_messages

class MessagesState(TypedDict):
    messages: Annotated[list[AnyMessage], add_messages]
```

```
from typing_extensions import TypedDict
```

- `TypedDict` cho phép bạn định nghĩa một kiểu dictionary có cấu trúc rõ ràng, giống như `dataclass`, nhưng dành cho dict.
- `typing_extensions` được dùng để hỗ trợ các tính năng mới khi bạn đang dùng phiên bản Python cũ hơn

```
from langchain_core.messages import AnyMessage
```

- `AnyMessage` là một kiểu tin nhắn (message) tổng quát trong LangChain.

- Có thể là tin nhắn từ người dùng ( `HumanMessage` ), AI ( `AIMessage` ), hệ thống ( `SystemMessage` )

```
from typing import Annotated
```

- `Annotated` là một cách để **gắn thêm metadata** vào kiểu dữ liệu.
- Không thay đổi kiểu dữ liệu, nhưng cho phép các công cụ/phần mềm xử lý kiểu đó biết thêm thông tin phụ.

Metadata là gì? [Xem tại đây](#)

`Annotated` là gì? [Xem tại đây](#)

```
from langgraph.graph.message import add_messages
```

- `add_messages` là một **annotation handler** từ thư viện `langgraph`.
- Nó được dùng để **bổ sung logic** cho cách mà LangGraph xử lý trường `messages`

```
class MessagesState(TypedDict):  
    messages: Annotated[list[AnyMessage], add_messages]
```

đây là định nghĩa một State schema

### III. Xác định Node duy nhất của ứng dụng

```
def simple_llm(state: MessagesState):  
    return {"messages": [chatModel4o.invoke(state["messages"])]}
```

Đây là một **LangGraph Node Function**. Nó:

1. Nhận vào `state` (dạng dictionary có cấu trúc `MessagesState`)
2. Lấy danh sách tin nhắn từ `state["messages"]`
3. Gửi toàn bộ danh sách đó vào mô hình ngôn ngữ `chatModel4o`
4. Nhận lại một phản hồi từ LLM
5. Trả về phản hồi đó trong một dict mới dưới dạng `{"messages": [<message>]}`

Ví dụ minh họa

Giả sử:

```
state = {  
    "messages": [  

```

```
HumanMessage(content="Chào bạn"),
]
```

Gọi

```
simple_llm(state)
```

Trả về

```
{
  "messages": [
    AIMessage(content="Chào bạn! Tôi có thể giúp gì?")
  ]
}
```

Tóm lại

Thành phần	Vai trò
<code>state: MessagesState</code>	Trạng thái hiện tại (danh sách message)
<code>chatModel4o.invoke(...)</code>	Gửi prompt đến LLM
<code>return {"messages": [...]}</code>	Trả về message mới để nối thêm vào state

**Vì trong ví dụ này chúng ta chỉ có một nút nên chúng ta không cần phải xác định các cạnh.**

## IV. Biên dịch ứng dụng

```
from IPython.display import Image, display
from langgraph.graph import StateGraph, START, END

# Build graph
builder = StateGraph(MessagesState)

builder.add_node("simple_llm", simple_llm)

# Add the logic of the graph
builder.add_edge(START, "simple_llm")
builder.add_edge("simple_llm", END)

# Compile the graph
```

```
graph = builder.compile()

# Visualize the graph
display(Image(graph.get_graph().draw_mermaid_png()))
```

```
from IPython.display import Image, display
```

- Dùng trong Jupyter Notebook để hiển thị hình ảnh (ở đây là sơ đồ luồng graph).
- `display(...)` dùng để hiển thị hình ảnh.
- `Image(...)` nhận vào dữ liệu ảnh hoặc URL.

```
from langgraph.graph import StateGraph, START, END
```

- `StateGraph`: class để **xây dựng graph gồm các bước (nodes)** xử lý state.
- `START` và `END`: các node mặc định dùng để đánh dấu điểm **bắt đầu** và **kết thúc**.

Xây dựng Graph:

```
builder = StateGraph(MessagesState)
```

Khởi tạo 1 builder để tạo ra graph, với `MessagesState` là định nghĩa schema cho **state**.

```
builder.add_node("simple_llm", simple_llm)
```

- Thêm một node có tên `"simple_llm"` và hàm xử lý tương ứng là `simple_llm`.

**Kết nối các bước xử lý:**

```
builder.add_edge(START, "simple_llm")
builder.add_edge("simple_llm", END)
```

`START → simple_llm → END`: bạn đang định nghĩa **luồng chạy của graph** theo thứ tự:

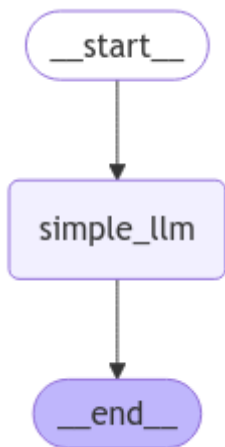
1. **Bắt đầu**
2. Chạy `simple_llm`
3. Kết thúc

**Biên dịch Graph:**

```
graph = builder.compile()
```

**Hiển thị sơ đồ:**

```
display(Image(graph.get_graph().draw_mermaid_png()))
```



## V. Chạy ứng dụng

```
from pprint import pprint
from langchain_core.messages import AIMessage, HumanMessage

messages = graph.invoke({"messages": HumanMessage(content="Where is the Golden Gate Bridge?")})

for m in messages['messages']:
    m.pretty_print()
```

- **Gọi Graph để xử lý một prompt** (giống như một chatbot).
- `graph.invoke(...)` là cách chạy luồng xử lý bạn đã định nghĩa bằng `StateGraph`.
- Bạn truyền vào một `dict` có key `"messages"` và giá trị là một **HumanMessage duy nhất**.

Trong thực tế, `"messages"` thường là **`list[BaseMessage]`**, nhưng LangGraph tự động chuyển đổi nếu bạn truyền một message duy nhất (nó sẽ biến nó thành `[HumanMessage(...)]`)

```
---### ```python
for m in messages['messages']:
    m.pretty_print()
```

## Kết quả in ra (ví dụ)

Giả sử AI trả lời như sau, bạn sẽ thấy:

```
Human: Where is the Golden Gate Bridge?
AI: The Golden Gate Bridge is located in San Francisco, California, USA.
```

## Diễn giải toàn bộ quy trình

1. Bạn tạo một **HumanMessage** hỏi AI một câu.
2. Gửi message đó vào graph (đã xây bằng LangGraph).
3. Graph chạy các node, ví dụ `simple_llm`, và thêm `AIMessage` phản hồi vào state.
4. Trả về danh sách messages mới (có cả user + AI).
5. In ra từng message bằng `.pretty_print()`.

## V. Bây giờ chúng ta hãy thêm một công cụ vào ChatModel4o

```
def multiply(a: int, b: int) -> int:
    """Multiply a and b.

    Args:
        a: first int
        b: second int
    """
    return a * b

chatModel4o_with_tools = chatModel4o.bind_tools([multiply])
```

Định nghĩa một hàm nhân 2 số nguyên `a` và `b`.

```
chatModel4o.bind_tools([multiply])
```

- Gắn hàm `multiply` vào mô hình `chatModel4o` như một **tool** (công cụ).
- Khi mô hình nhận câu hỏi phù hợp (ví dụ: "*What is 6 times 7?*"), nó có thể **gọi hàm** `multiply` để xử lý thay vì tự trả lời.

## VI. Bây giờ chúng ta hãy tạo một ứng dụng LangGraph thứ hai có thể quyết định xem ứng dụng đó có sử dụng chatbot LLM hay Công cụ Multiply để trả lời câu hỏi của người dùng hay không

```
# Node
def llm_with_tool(state: MessagesState):
    return {"messages": [chatModel4o_with_tools.invoke(state["messages"])]}
```

```
from IPython.display import Image, display
from langgraph.graph import StateGraph, START, END

# Build graph
builder = StateGraph(MessagesState)
```

```

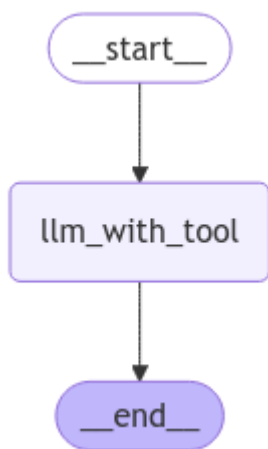
builder.add_node("llm_with_tool", llm_with_tool)

# Add the logic of the graph
builder.add_edge(START, "llm_with_tool")
builder.add_edge("llm_with_tool", END)

# Compile the graph
graph = builder.compile()

# Visualize the graph
display(Image(graph.get_graph().draw_mermaid_png()))

```



```

from pprint import pprint
from langchain_core.messages import AIMessage, HumanMessage

# The following two lines are the most frequent way to
# run and print a LangGraph chatbot-like app results.
messages = graph.invoke({"messages": HumanMessage(content="Where is the Eiffel Tower?")})

for m in messages['messages']:
    m.pretty_print()

```

```

===== Human Message
===== Where is the Eiffel Tower?
===== Ai Message
===== The Eiffel Tower is located in Paris,
France. It is situated on the Champ de Mars, near the Seine River.

```

```
from pprint import pprint
from langchain_core.messages import AIMessage, HumanMessage

# The following two lines are the most frequent way to
# run and print a LangGraph chatbot-like app results.
messages = graph.invoke({"messages": HumanMessage(content="Multiply 4 and 5")})

for m in messages['messages']:
    m.pretty_print()
```

```
===== Human Message
===== Multiply 4 and 5
===== Ai Message
===== Tool Calls: multiply
(call_QRyPmp5TdcllfEOyBrJ9E7V5) Call ID: call_QRyPmp5TdcllfEOyBrJ9E7V5 Args: a: 4 b: 5
```

**Tác giả: Đỗ Ngọc Tú**  
**Công Ty Phần Mềm VHTSoft**

---

Phiên bản #7

Được tạo 17 tháng 4 2025 03:27:41 bởi Đỗ Ngọc Tú

Được cập nhật 17 tháng 4 2025 08:55:37 bởi Đỗ Ngọc Tú