

5 Khái Niệm Trụ Cột

LangGraph là một framework mạnh mẽ để xây dựng các ứng dụng LLM có logic phức tạp theo mô hình **graph-based agent**. Để làm chủ LangGraph, bạn cần hiểu rõ 5 thành phần chính: **State**, **Schema**, **Node**, **Edge**, và **Compile**.

1. State = Memory (Trạng thái là bộ nhớ)

Trong LangGraph, **State** là nơi lưu trữ toàn bộ thông tin mà ứng dụng của bạn cần ghi nhớ trong suốt quá trình chạy.

- Nó có thể chứa input từ người dùng, kết quả từ LLM, thông tin truy xuất từ tool, v.v.
- Mỗi node trong graph có thể **đọc và ghi vào state**.
- State giúp ứng dụng **suy nghĩ nhiều bước**, ghi nhớ dữ kiện, và sử dụng lại ở bước sau.

Ví dụ:

```
{ "question": "Chủ tịch nước Việt Nam là ai?", "llm_response": "..." }
```

State giống như **RAM** của agent – càng tổ chức tốt, agent càng thông minh và hiệu quả.

2. Schema = Data Format (Định dạng dữ liệu)

LangGraph yêu cầu bạn **định nghĩa Schema cho State** để đảm bảo dữ liệu được quản lý nhất quán, an toàn và rõ ràng.

- Schema là một **class Python** (thường dùng `pydantic.BaseModel` hoặc `TypedDict`)
- Nó xác định rõ những gì tồn tại trong state, loại dữ liệu, và bắt buộc hay không.

Ví dụ:

```
from typing import TypedDict

class MyState(TypedDict):
    question: str
    answer: str
```

Schema = bản thiết kế dữ liệu, giúp bạn tránh lỗi, dễ debug và mở rộng.

3. Node = Step (Một bước xử lý)

Node là một bước trong đồ thị – nơi một phần công việc được thực hiện.

- Có thể là một LLM call, một hành động với tool, một quyết định logic, v.v.
- Mỗi node nhận `state`, xử lý và trả lại `state` đã được cập nhật.

Ví dụ node đơn giản:

```
def call_llm(state):  
    response = llm.invoke(state["question"])  
    state["answer"] = response  
    return state
```

4. Edge = Kết nối giữa các bước(Operation Between Steps)

Edge là mối liên kết giữa các node – giúp bạn định nghĩa luồng đi của chương trình.

- Một node có thể có **nhiều edge** đi tới các bước khác nhau (ví dụ dựa vào kết quả).
- Có thể định nghĩa **logic rẽ nhánh (branching)**, **lặp lại (loop)** hoặc **kết thúc** tại đây.

Ví dụ:

```
graph.add_edge("ask_user", "call_llm")  
graph.add_conditional_edges("call_llm", {  
    "need_more_info": "search_tool",  
    "done": END  
})
```

Edge quyết định luồng suy nghĩ và logic quyết định của agent.

5. Compile = Đóng gói đồ thị để sử dụng(Pack)

Sau khi bạn đã định nghĩa đầy đủ các **node**, **edge** và **schema**, bước cuối cùng là **compile** – để tạo ra một graph sẵn sàng chạy.

- `compile()` sẽ kiểm tra sự nhất quán, ràng buộc, và đóng gói toàn bộ graph.
- Sau đó, bạn có thể dùng `.invoke()` để chạy agent một cách mượt mà.

Ví dụ:

```
builder = StateGraph(MyState)
# add node, edge...
graph = builder.compile()
graph.invoke({"question": "..."})
```

Compile = **chuyển bản thiết kế sang sản phẩm hoàn chỉnh có thể vận hành.**

Tóm lại

Thành phần	Vai trò	Ví dụ dễ hiểu
State	Bộ nhớ dùng để truyền dữ liệu giữa các bước	“Tôi nhớ câu hỏi người dùng”
Schema	Định nghĩa cấu trúc và loại dữ liệu trong state	“State phải có ‘question’ dạng chuỗi”
Node	Một bước xử lý như LLM call, tool call, v.v.	“Gọi GPT để trả lời”
Edge	Điều khiển luồng di chuyển giữa các node	“Nếu xong thì kết thúc, nếu thiếu thì tìm tiếp”
Compile	Đóng gói toàn bộ graph để vận hành	“Biến các bước rời rạc thành một chương trình”

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

Phiên bản #2
Được tạo 15 tháng 4 2025 02:11:33 bởi Đỗ Ngọc Tú
Được cập nhật 21 tháng 4 2025 03:13:15 bởi Đỗ Ngọc Tú