

# Thực hành – Node LLM quyết định bước tiếp theo

Chúng ta sẽ xây dựng lại ứng dụng trước đó, nhưng lần này sẽ bổ sung hành vi mang tính "agent" điển hình. Theo cách gọi của nhóm LangGraph, đây sẽ là **Agent đầu tiên** của chúng ta.

## Những điểm chính:

- **Hành vi agent điển hình:** node công cụ (tool node) sẽ phản hồi lại cho LLM, và sau đó LLM sẽ quyết định bước tiếp theo.
- Node bao gồm chatbot, công cụ (tool), và thông điệp hệ thống (system message).
- Các thao tác gọi công cụ theo trình tự (sequential tool-calling operations).

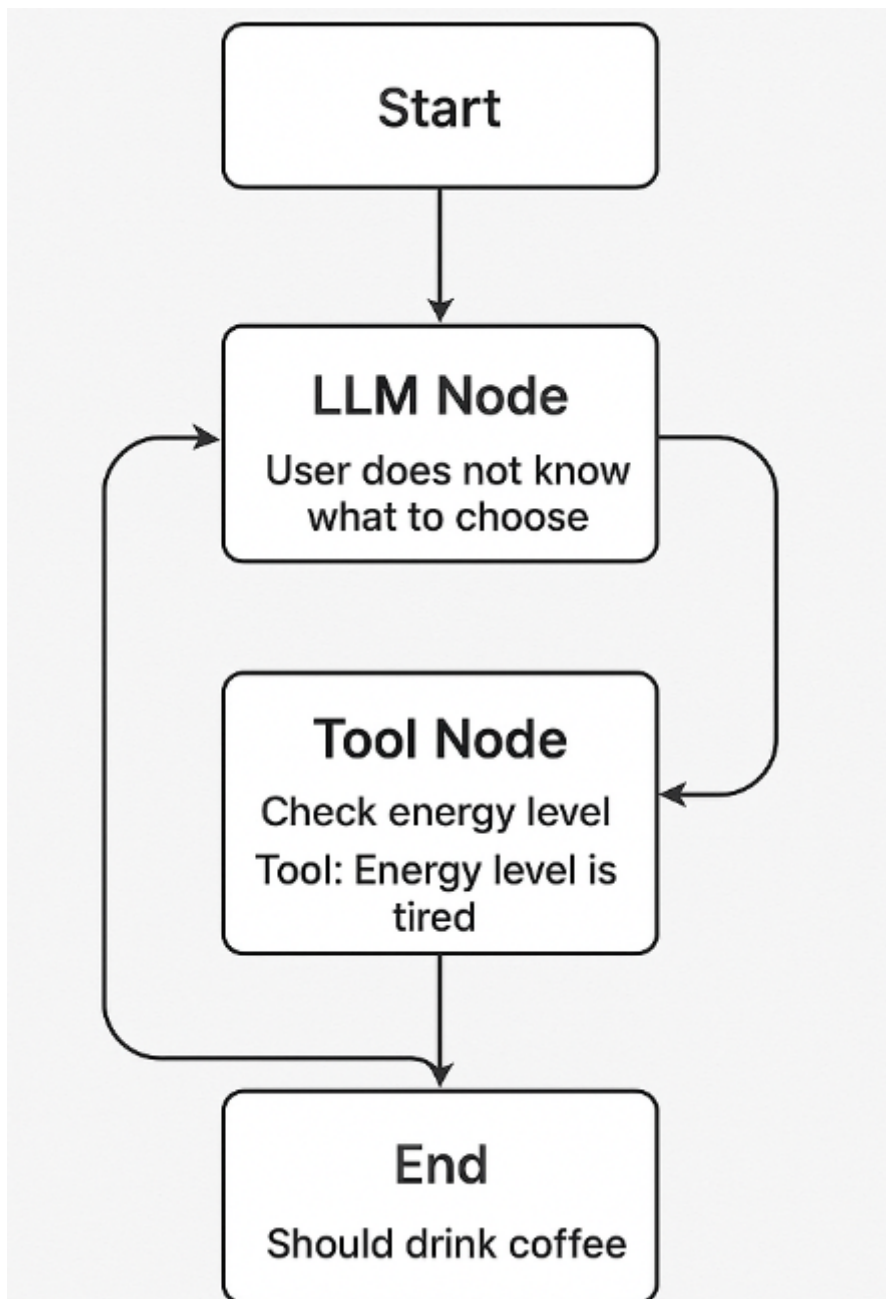
## Agent Cơ Bản

- **Mục tiêu:** Chúng ta sẽ xây dựng một ứng dụng tương tự như trong bài thực hành trước. **Điểm khác biệt chính** là trong ứng dụng lần này, **node công cụ (tools node) sẽ phản hồi trở lại cho assistant (LLM) thay vì đi thẳng đến node KẾT THÚC (END Node)**.
- Điều này thể hiện kiến trúc được gọi là **ReAct**: mô hình LLM sẽ tiếp tục sử dụng các công cụ cho đến khi nhận được phản hồi thỏa đáng.
  - **act** – LLM gọi một công cụ.
  - **observe** – công cụ trả kết quả lại cho LLM.
  - **reason** – LLM quyết định bước tiếp theo (gọi công cụ khác hoặc trả lời trực tiếp).

## Thực hành

Tạo một agent sử dụng kiến trúc ReAct với quy trình:

1. Người dùng không biết chọn gì → LLM hỏi ý kiến.
2. LLM gọi một tool để kiểm tra trạng thái năng lượng người dùng.
3. Tool trả lời về mức năng lượng.
4. Dựa vào đó, LLM đưa ra lời khuyên nên uống trà hay cà phê.



## 1. Cài đặt và chuẩn bị

```
pip install langgraph openai
```

## 2. Định nghĩa State

```
from typing import TypedDict, Literal, Optional

class DrinkState(TypedDict):
    preference: Literal["coffee", "tea", "unknown"]
    energy_level: Optional[str]
    messages: list[dict]
```

### 3. Tạo công cụ kiểm tra năng lượng

```
def check_energy_level(state: DrinkState) -> DrinkState:
    # Giả sử công cụ xác định người dùng đang "mệt"
    energy = "tired"
    state["energy_level"] = energy
    state["messages"].append({"role": "tool", "content": f"Energy level is {energy}"})
    return state
```

### 4. Tạo node LLM để đưa ra quyết định

```
from langchain.chat_models import ChatOpenAI

llm = ChatOpenAI(temperature=0)

def llm_node(state: DrinkState) -> DrinkState:
    messages = state["messages"]
    response = llm.invoke(messages)
    state["messages"].append({"role": "assistant", "content": response.content})
    if "coffee" in response.content.lower():
        state["preference"] = "coffee"
    elif "tea" in response.content.lower():
        state["preference"] = "tea"
    return state
```

### 5. Tạo Graph

```
from langgraph.graph import StateGraph

builder = StateGraph(DrinkState)

builder.add_node("llm", llm_node)
builder.add_node("check_energy", check_energy_level)

builder.set_entry_point("llm")

# LLM gọi tool rồi quay lại chính nó
builder.add_edge("llm", "check_energy")
builder.add_edge("check_energy", "llm")
```

```
# Thiết lập điểm kết thúc
builder.set_finish_point("llm")

graph = builder.compile()
```

## 6. Chạy thử

```
initial_state = {
    "preference": "unknown",
    "energy_level": None,
    "messages": [{ "role": "user", "content": "Tôi không biết nên uống gì hôm nay" }],
}

final_state = graph.invoke(initial_state)
print("Gợi ý của Agent:", final_state["preference"])
```

# Giải thích luồng hoạt động

1. Người dùng không đưa ra lựa chọn cụ thể.
2. LLM phân tích và quyết định cần kiểm tra mức năng lượng.
3. Tool `check_energy` trả về trạng thái "mệt".
4. LLM đưa ra khuyến nghị uống cà phê để tăng tỉnh táo.
5. Kết thúc.

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

---

Phiên bản #2  
Được tạo 19 tháng 4 2025 08:06:18 bởi Đỗ Ngọc Tú  
Được cập nhật 21 tháng 4 2025 05:50:54 bởi Đỗ Ngọc Tú