

Giới thiệu về ReAct Architecture

Kiến trúc **ReAct** là một mô hình rất quan trọng trong việc xây dựng các Agent dựa trên LLM. Tên “ReAct” là viết tắt của **Reasoning and Acting** – tức là kết hợp giữa *lập luận* và *hành động*. Đây là cách mà một Agent thông minh có thể thực hiện nhiều bước để hoàn thành một nhiệm vụ, thay vì chỉ trả lời một lần rồi kết thúc.

Giới thiệu về ReAct Architecture

ReAct cho phép các LLM (Large Language Models) hoạt động theo chu trình:

1. **act (hành động)**

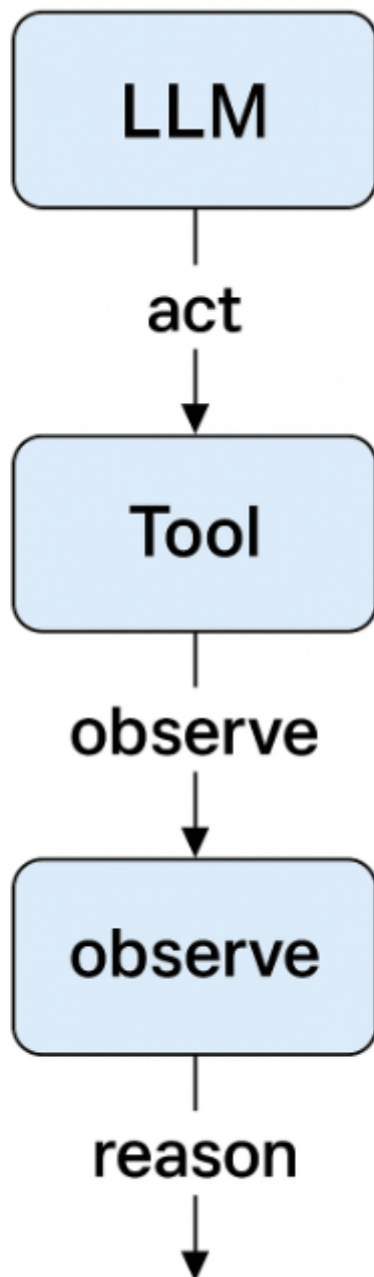
Mô hình quyết định gọi một công cụ (tool) để thực hiện một hành động cụ thể. Ví dụ: tìm kiếm thông tin, tính toán, tra dữ liệu từ API, v.v.

2. **observe (quan sát)**

Công cụ thực hiện hành động và trả về kết quả. Kết quả này sẽ được đưa trở lại LLM như một phần của cuộc hội thoại (hoặc “context”).

3. **reason (lập luận)**

LLM tiếp nhận kết quả, phân tích và quyết định bước tiếp theo: có thể gọi thêm một công cụ khác, tiếp tục hỏi người dùng, hoặc kết thúc bằng một câu trả lời.



ReAct là cách mà **AI (LLM)** suy nghĩ và hành động như một con người:

“ Nghĩ – Làm – Quan sát – Nghĩ tiếp – Làm tiếp...”

Câu chuyện đơn giản

Bạn nói với AI:

"Tính giúp tôi $3 + 5$ "

AI làm gì?

Bước	Hành động	Giải thích dễ hiểu
Act	"Tôi sẽ gọi công cụ máy tính để tính toán"	(Gọi hàm <code>calculator_tool</code>)
Tool	Máy tính nhận yêu cầu, tính ra kết quả <code>8</code>	
Observe	AI nhìn vào kết quả: <code>8</code>	(Cập nhật kết quả vào bộ nhớ)
Reason	AI tự hỏi: "Người dùng muốn tính tiếp không?"	Nếu có, lặp lại từ đầu

Ví dụ thực tế

Giả sử bạn hỏi:
"Hôm nay thời tiết ở TP Hồ Chí Minh như thế nào và tôi có nên mang ô không?"

Agent thực hiện:

- **act:** Gọi một công cụ thời tiết để lấy dữ liệu thời tiết tại **TP Hồ Chí Minh**.
- **observe:** Nhận được kết quả: "Trời có khả năng mưa vào chiều tối."
- **reason:** LLM suy luận: "Trời có thể mưa → nên mang ô → trả lời người dùng."

Trả lời cuối cùng:
"Hôm nay **TP Hồ Chí Minh** có khả năng mưa vào chiều tối, bạn nên mang theo ô."

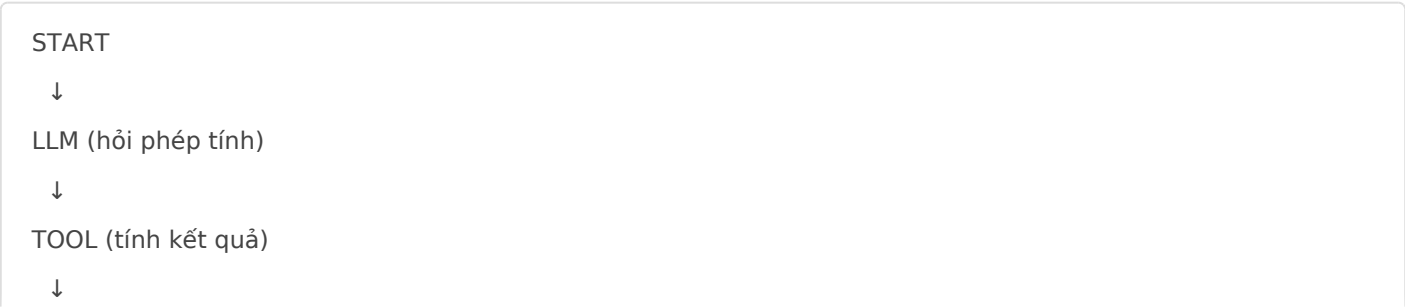
Ứng dụng của ReAct

- Xây dựng **multi-step agents**: Giải quyết các bài toán phức tạp cần nhiều hành động trung gian.
- Cho phép Agent có thể **tự khám phá**, điều chỉnh kế hoạch dựa trên kết quả trung gian.
- Hữu ích trong các hệ thống như **LangGraph, LangChain**, nơi bạn cần điều phối nhiều bước và công cụ.

Ví dụ cơ bản

Tạo một agent hỏi người dùng một phép tính, gọi một công cụ để tính, rồi tiếp tục hỏi đến khi người dùng muốn dừng.

Kiến trúc Agent như sau



LLM (xem kết quả, hỏi tiếp hay kết thúc?)

→ Nếu tiếp → quay lại TOOL

→ Nếu kết thúc → END

1. Định nghĩa **State**

```
from typing import TypedDict, List, Optional
```

```
class CalcState(TypedDict):
```

```
    history: List[str]
```

```
    next_action: Optional[str]
```

```
    question: Optional[str]
```

```
    answer: Optional[str]
```

2. Tạo Tool Node

```
def calculator_tool(state: CalcState) -> CalcState:
```

```
    question = state["question"]
```

```
    try:
```

```
        answer = str(eval(question)) # WARNING: chỉ dùng với ví dụ đơn giản!
```

```
    except:
```

```
        answer = "Không thể tính được."
```

```
    state["answer"] = answer
```

```
    state["history"].append(f"Q: {question}\nA: {answer}")
```

```
    return state
```

3. Tạo LLM Node

```
def llm_node(state: CalcState) -> CalcState:
```

```
    print("\n\n LLM: Tôi đang nghĩ...")
```

```
    print(f"Câu hỏi: {state.get('question')}")
```

```
    print(f"Kết quả: {state.get('answer')}")
```

```
    # Quyết định bước tiếp theo
```

```
    choice = input("Bạn muốn tiếp tục (yes/no)? ")
```

```
    if choice.lower() == "no":
```

```
        state["next_action"] = "end"
```

```
    else:
```

```
        new_q = input("Nhập phép tính mới: ")
```

```
        state["question"] = new_q
```

```
state["next_action"] = "calculate"
return state
```

4. Router

```
def calc_router(state: CalcState) -> str:
    return state["next_action"]
```

5. Kết nối các bước bằng LangGraph

```
from langgraph.graph import StateGraph

builder = StateGraph(CalcState)

builder.add_node("llm_node", llm_node)
builder.add_node("tool_node", calculator_tool)
builder.add_node("router", calc_router)

builder.set_entry_point("llm_node")
builder.add_edge("llm_node", "router")

builder.add_conditional_edges("router", {
    "calculate": "tool_node",
    "end": "__end__"
})

builder.add_edge("tool_node", "llm_node")
graph = builder.compile()
```

- Dòng 5,6,7 Chúng ta thêm 3 nodes: llm_node, calculator_tool, calc_router
- Dòng 9, llm_node là được đặt là node đầu tiên
- Dòng 10, chúng ta nối node có id là llm_node với node có id là `router` (**router là một node đặc biệt, node này không trả về một state mà trả về một chuỗi**)
- Dòng 12, router thực hiện rẽ nhánh dựa trên add_conditional_edges
 - "router" là id của node calc_router, nếu calc_router trả về "calculate" thì thực hiện tool_node, nếu calc_router trả về "end" thì thực hiện "__end__"
- Dòng 17, thực hiện nối "tool_node" với "llm_node" và tiếp tục

6. Chạy thử

```
state = {
    "history": [],
```

```
"next_action": None,  
"question": "2 + 3",  
"answer": None,  
}
```

```
graph.invoke(state)
```

Output mẫu

LLM: Tôi đang nghĩ...

Câu hỏi: 2 + 3

Kết quả: None

Nhập phép tính mới: 2 + 3

Q: 2 + 3

A: 5

Bạn muốn tiếp tục (yes/no)? yes

Nhập phép tính mới: 10 * 2

...

Tóm tắt kiến trúc ReAct

Pha	Vai trò	Ý nghĩa
Act	LLM gọi công cụ	LLM yêu cầu tính một biểu thức toán học
Observe	Công cụ trả kết quả	Công cụ <code>calculator_tool</code> xử lý phép tính và trả về kết quả
Reason	LLM quyết định bước tiếp theo	Dựa trên kết quả, LLM quyết định tiếp tục hay dừng
Tool	Công cụ được LLM gọi	Ở đây là hàm <code>calculator_tool</code> dùng <code>eval()</code> để tính toán

Phân tích chi tiết ví dụ trên

1. Act - LLM gọi công cụ

Trong hàm `llm_node`, nếu người dùng muốn tiếp tục:

```
state["next_action"] = "calculate"
```

LLM quyết định hành động tiếp theo là **gọi công cụ tính toán**.

2. Observe - Công cụ trả kết quả

```
def calculator_tool(state: CalcState) -> CalcState:
    ...
    state["answer"] = answer
```

Công cụ thực hiện tính toán và **cập nhật lại state** (trạng thái) với kết quả.
LLM sẽ **quan sát lại kết quả** ở bước tiếp theo.

3. Reason - LLM quyết định tiếp theo

Sau khi đã có kết quả trong `state["answer"]`, LLM đánh giá và hỏi:

```
choice = input("Bạn muốn tiếp tục (yes/no)? ")
```

Nếu **yes** thì act lần nữa (gọi lại công cụ).
Nếu **no** thì kết thúc (gọi `"end"`).

4. Tool - Công cụ độc lập

```
def calculator_tool(state: CalcState) -> CalcState:
    ...
```

Đây chính là một công cụ (Tool) – thực hiện công việc cụ thể do LLM yêu cầu.
Nó không thông minh, chỉ đơn thuần xử lý dữ liệu đầu vào.

Tổng kết dễ hiểu

Bước	Thành phần	Vai trò
1	<code>llm_node</code>	Act: Quyết định gọi tool
2	<code>calculator_tool</code>	Observe: Trả kết quả về
3	<code>llm_node</code>	Reason: Xem kết quả và quyết định hành động tiếp theo

Tác giả: Đỗ Ngọc Tú
Công Ty Phần Mềm VHTSoft

Phiên bản #7
Được tạo 19 tháng 4 2025 08:34:34 bởi Đỗ Ngọc Tú
Được cập nhật 21 tháng 4 2025 14:34:08 bởi Đỗ Ngọc Tú