Phản hồi từ người dùng(Human Feedback)

Human Feedback, giúp bạn cải thiện agent theo thời gian, học từ hành vi và đánh giá của con người.

Giới thiệu

Phần này chúng ta sẽ học những khái niệm cực kỳ quan trọng, nhưng xin lưu ý ngay từ đầu:

Lời khuyên học tập quan trọng

⚠ Đây vẫn là phần kiến thức nền tảng trước khi xây dựng ứng dụng hoàn chỉnh. Tôi khuyên bạn không nên dành quá nhiều thời gian thực hành các bài tập trong phần này ngay lúc này, vì:

- Dễ bị xao nhãng khỏi mục tiêu chính
- Hiện tại quan trọng nhất là nắm vững khái niệm cốt lõi
- Các kỹ thuật này sẽ trở nên thực sự hữu ích khi bạn bắt đầu phát triển agent AI hoàn chỉnh với LangGraph

Khi đó bạn sẽ tự nhiên cảm thấy:

- Mong muốn áp dụng các kỹ thuật này
- Hứng thú thử nghiệm để cải thiện agent của mình
- Hiểu sâu hơn về cách vận hành thực tế

Vấn đề phổ biến khi học framework mới

Hầu hết tài liệu kỹ thuật (bao gồm cả LangGraph) đều chứa đầy "cạm bẫy" khiến bạn:

- Dễ bị phân tâm
- Bối rối không biết bắt đầu từ đâu
- Nhanh chán nản và bỏ cuộc

Khác biệt của cách tiếp cận này:

1. Đơn giản hóa tài liệu gốc
2. Định hướng lộ trình học đúng đắn
3. Tập trung vào ứng dụng thực tế thay vì lý thuyết suông

2 điểm trọng tâm của phần này

1. Chuyển từ `.invoke()` sang `.stream()`

- Giống như trong khóa học về LangChain trước đây
- Bước chuyển đổi này mở ra rất nhiều khả năng mới
- Là một trong những điểm quan trọng nhất của phần này

Ví dụ so sánh:

```
# Cách cũ
result = agent.invoke({"input": "Hello"})

# Cách mới
for chunk in agent.stream({"input": "Hello"}):
    # Xử lý từng phần
    print(chunk)
```

2. Xử lý stream data cho Human-in-the-loop

Bí mật lớn: Ứng dụng AI hiện nay vẫn chưa thực sự "thông minh" hoàn toàn. Chúng cần con người để đạt hiệu suất tối đa (thường tăng thêm 5-20% chất lượng).

3 hướng tác động chính của Human-in-the-loop:

1. **Phê duyệt/ từ chối** các bước tiếp theo của ứng dụng
2. **Điều chỉnh trạng thái** hệ thống
3. **Gỡ lỗi** ứng dụng trong thời gian thực

Ví dụ thực tế:

```
for chunk in agent.stream(conversation):
    if needs_human_review(chunk):
        human_feedback = get_human_input()
        agent.adjust_based_on_feedback(human_feedback)
```

Bổ sung quan trọng từ tôi

LangGraph thực sự tỏa sáng khi kết hợp:

- Khả năng xử lý luồng dữ liệu (.stream())
- Cơ chế phản hồi của con người
- Kiến trúc có trạng thái (stateful)

Lời khuyên:

1. Tập trung hiểu cơ chế stream
2. Ghi chú các trường hợp cần human feedback
3. Đừng sa đà vào tối ưu hóa chi tiết
4. Chuẩn bị tinh thần cho phần human-in-the-loop sắp tới

Tác giả: Đỗ Ngọc Tú

Công Ty Phần Mềm VHTSoft

Streaming và Human-in-the-loop trong LangGraph

Streaming = "Truyền liên tục", nghĩa là agent hoặc LLM sẽ trả về **kết quả dần dần**, **theo dòng**, **chứ không phải chờ tính toán xong mới gửi hết**.

- **Streaming Values:** bạn nhận kết quả từng bước nhỏ (thường là dạng text hoặc dữ liệu).
- **Streaming Update:** bạn vừa nhận được giá trị ban đầu, vừa có thể **nhận bản cập nhật** mới hơn về sau (khi tác vụ tiến triển).

Thuật ngữ	Ý nghĩa	Ví dụ thực tế
Streaming Values	Nhận các giá trị từng phần	Từng đoạn text của câu trả lời
Streaming Updates	Các bản cập nhật của 1 giá trị ban đầu	Đang xử lý... Đã hoàn thành

Ví dụ đơn giản:

Giả sử bạn có một AI Agent thực hiện tóm tắt 1 tài liệu lớn.

- Ban đầu, nó đọc tài liệu và trả về **"Đang đọc chương 1..."** (streaming value)
- Khi đọc thêm, nó **cập nhật** thành **"Đã đọc chương 1, đang đọc chương 2..."** (streaming update)

Bạn nhận được **dòng text** đầu tiên nhanh chóng, sau đó **nhiều bản cập nhật liên tiếp**.

Ví dụ cụ thể bằng Python (giả lập):

```
import time

# Streaming giá trị ban đầu
def stream_value():
    print("□□ Loading document...")
    time.sleep(1)
    yield "□  Step 1: Read Chapter 1"

    time.sleep(2)
    yield "□  Step 2: Read Chapter 2"
```

```
    time.sleep(1)
    yield "□ Step 3: Summarizing Chapters"

# Xử lý streaming
for value in stream_value():
    print(f"□□ Received update: {value}")
```

Kết quả

```
Loading document...
Received update: □ Step 1: Read Chapter 1
Received update: □ Step 2: Read Chapter 2
Received update: □ Step 3: Summarizing Chapters
```

Ví dụ đơn giản về cách streaming câu trả lời trong chatbot

Tình huống:

Người dùng hỏi: **"Tóm tắt tác phẩm Dế Mèn Phiêu Lưu Ký"**

Ta sẽ cho chatbot trả lời dần dần (streaming) theo từng câu hoặc từng đoạn văn bản.

Ví dụ Python (giả lập):

```
import time

def chatbot_streaming_response():
    yield "Dế Mèn Phiêu Lưu Ký là tác phẩm nổi tiếng của nhà văn Tô Hoài."
    time.sleep(1)

    yield "Tác phẩm kể về hành trình phiêu lưu của chú dế Mèn thông minh, gan dạ và thích khám phá."
    time.sleep(1)

    yield "Trong hành trình đó, dế Mèn đã học được nhiều bài học quý giá về tình bạn, lòng dũng cảm và sự trưởng thành."
    time.sleep(1)

    yield "Tác phẩm không chỉ hấp dẫn thiếu nhi mà còn chứa đựng nhiều triết lý sống sâu sắc."
    time.sleep(1)

# Giả lập chatbot gửi từng đoạn
```

```
for chunk in chatbot_streaming_response():
    print(f"□□ {chunk}")
```

Kết quả mô phỏng trên terminal:

```
□□ Dế Mèn Phiêu Lưu Ký là tác phẩm nổi tiếng của nhà văn Tô Hoài.
□□ Tác phẩm kể về hành trình phiêu lưu của chú dế Mèn thông minh, gan dạ và thích khám phá.
□□ Trong hành trình đó, dế Mèn đã học được nhiều bài học quý giá về tình bạn, lòng dũng cảm và sự trưởng thành.
□□ Tác phẩm không chỉ hấp dẫn thiếu nhi mà còn chứa đựng nhiều triết lý sống sâu sắc.
```

Sự khác biệt chính giữa `.invoke()` và `.stream()`

Đặc điểm	`.invoke()`	`.stream()`
Cách hoạt động	Gọi toàn bộ chain hoặc agent và chờ kết quả	Trả về kết quả **từng phần một** (token, chunk, etc.)
Tốc độ phản hồi	Trả lời sau khi xử lý xong toàn bộ	Trả lời gần như **ngay lập tức**, theo thời gian thực
Trường hợp sử dụng	Tốt cho các xử lý ngắn hoặc không cần realtime	Lý tưởng cho **chatbot**, **UI realtime**, hoặc truyền dữ liệu
Trả về	Toàn bộ output cùng lúc (ví dụ một string)	Một generator (yield từng phần kết quả)
Tích hợp UI	Không thể hiện đang gõ	Có thể hiện đang gõ như ChatGPT

Lời khuyên thực tế khi triển khai

1. **Ưu tiên dùng stream_mode="values"** khi phát triển để dễ debug
2. Chỉ chuyển sang "updates" khi ứng dụng đã ổn định và cần tối ưu hiệu suất
3. Luôn thiết kế các "điểm kiểm soát" (checkpoints) cho human-in-the-loop tại:
   - Bước ra quyết định quan trọng
   - Khi cần xác nhận tính chính xác
   - Khi agent không chắc chắn
4. Xử lý lỗi thông minh:

```
try:
    for chunk in agent.stream(...):
        process(chunk)
except HumanInterventionRequired as e:
    handle_exception(e)
```

```
    wait_for_human_decision()
```

Bổ sung kiến thức quan trọng

Tại sao .stream() mạnh mẽ hơn?

- Cho phép xây dựng **agent có khả năng tự sửa lỗi**
- Tạo điều kiện cho **học tăng cường từ con người** (RLHF)
- Hỗ trợ **xử lý tác vụ dài** mà không bị timeout
- Cho phép **hiển thị tiến trình thời gian thực** cho người dùng

Pattern hay gặp:

```
# Hybrid approach - Kết hợp cả invoke và stream khi cần
if is_simple_task(task):
    return agent.invoke(task)
else:
    return process_stream(agent.stream(task))
```

Hãy coi `.stream()` như "cửa sổ vào bộ não" của agent - bạn có thể quan sát và can thiệp vào quá trình xử lý theo cách chưa từng có với `.invoke()` thông thường!

Tác giả: Đỗ Ngọc Tú

Công Ty Phần Mềm VHTSoft

Breakpoints và Human-in-the-loop trong LangGraph

Breakpoint là gì?

> Một **breakpoint** trong LangGraph cho phép bạn **tạm dừng workflow tại một node** (bước), từ đó bạn có thể:

- Kiểm tra input/output của node đó
- Thay đổi trạng thái trước khi tiếp tục
- Dùng trong quá trình **debug**, **testing**, hoặc **human-in-the-loop**

Cách hoạt động

Khi một node có breakpoint, LangGraph sẽ **trả về trạng thái tạm thời** (trạng thái dừng), và bạn có thể tiếp tục thực thi bằng cách gọi `.invoke()` hoặc `.stream()` lại với trạng thái đã cập nhật.

Ví dụ đơn giản: Breakpoint ở một bước LLM

1. Khởi tạo LangGraph

```
from langgraph.graph import StateGraph, END
from langchain_core.runnables import RunnableLambda

# Bước 1: Xử lý văn bản
def process_text(state):
    text = state["input"]
    return {"processed": text.upper()}

# Bước 2: Tổng kết
def summarize(state):
    return {"summary": f"Summary: {state['processed']}"}

graph = StateGraph()
```

```
# Thêm node và breakpoints
graph.add_node("process", RunnableLambda(process_text))
graph.add_node("summarize", RunnableLambda(summarize))

graph.set_entry_point("process")
graph.add_edge("process", "summarize")
graph.add_edge("summarize", END)

# Gắn breakpoint vào bước "summarize"
graph.add_conditional_edges("summarize", lambda x: "__break__")

app = graph.compile()
```

2. Gọi `invoke()` và dừng tại breakpoint

```
state = app.invoke({"input": "Hello LangGraph!"})
print(state)
```

Output sẽ không kết thúc workflow, mà dừng tại node `"summarize"` vì có breakpoint.

3. Kiểm tra hoặc chỉnh sửa rồi tiếp tục

```
# Bạn có thể chỉnh sửa state nếu muốn
state["processed"] = "HELLO LANGGRAPH! (edited)"

# Tiếp tục từ breakpoint
final = app.invoke(state)
print(final)
```

Khi nào nên dùng breakpoints?

Tình huống	Có nên dùng không?
Debug từng bước của graph	Có
Thêm bước xác nhận từ người dùng	Có
Gửi yêu cầu API cần kiểm tra trước	Có
Tự động hóa hoàn toàn không kiểm tra	Không cần

Tác giả: Đỗ Ngọc Tú

Công Ty Phần Mềm VHTSoft