

Hiệu quả bộ nhớ và khả năng lưu trữ bộ nhớ trong Agent LangGraph

Trong quá trình xây dựng tác nhân AI với LangGraph, việc quản lý bộ nhớ hiệu quả và duy trì bộ nhớ lâu dài là hai yếu tố then chốt để đảm bảo ứng dụng vừa tiết kiệm chi phí vừa vận hành ổn định. **Hiệu quả bộ nhớ** giúp giảm lượng token tiêu thụ, tối ưu tốc độ phản hồi và chi phí sử dụng mô hình AI. Trong khi đó, **khả năng lưu trữ bộ nhớ lâu dài** (memory persistence) cho phép tác nhân ghi nhớ thông tin quan trọng giữa các phiên làm việc, tạo ra trải nghiệm thông minh và cá nhân hóa hơn cho người dùng. Bài viết này sẽ giới thiệu những kỹ thuật cơ bản và những lưu ý cần thiết khi triển khai hai yếu tố quan trọng này trong LangGraph.

- [Giới thiệu](#)
- [Tối Ưu Chi Phí và Quản Lý Bộ Nhớ Khi Xây Dựng Ứng Dụng AI Với OpenAI](#)
- [Tối ưu hóa bộ nhớ ngắn hạn bằng cách tóm tắt hội thoại](#)
- [Giới thiệu về Bộ nhớ Ngoài \(Persistent Memory\) với LangGraph](#)
- [Tóm tắt chương](#)

Giới thiệu

Trong phần này, chúng ta sẽ khám phá ba kỹ thuật rất thú vị.

Tuy nhiên, điều quan trọng nhất tôi muốn bạn ghi nhớ: **Đừng để mình bị phân tâm.**

Mục tiêu của tôi là bạn nắm chắc **ý tưởng** và **khái niệm cốt lõi** đằng sau mỗi kỹ thuật hoặc bài học này. Bạn không cần dành quá nhiều thời gian thực hành hoặc áp dụng ngay lập tức, bởi vì:

- Những kỹ thuật này sẽ **rất hữu ích** khi bạn bắt đầu xây dựng **ứng dụng thực tế nghiêm túc**.
- Khi đó, bạn sẽ nhớ rằng: "À, trước đây mình đã học về những kỹ thuật này", và đó chính là **thời điểm thích hợp** để bắt đầu thử nghiệm và ứng dụng.

Đây cũng là một **lời khuyên quan trọng** mà bạn có thể áp dụng nhiều lần khi học hoặc nghiên cứu tài liệu kỹ thuật — ví dụ như tài liệu của **LangGraph**.

Cẩn Thận Với Những "Sự Phân Tâm" Khi Học Tài Liệu LangGraph

Khi bạn đọc tài liệu của LangGraph, sẽ có rất nhiều **kỹ thuật nhỏ, thủ thuật đặc biệt** hiện ra và thu hút sự chú ý.

Nếu không cẩn thận, bạn có thể dễ dàng:

- Bị cuốn vào các chi tiết kỹ thuật phụ
- Mất động lực học các phần quan trọng hơn
- Chán nản và bỏ dở quá trình học

Điều quan trọng là:

- **Hãy ghi nhận sự tồn tại của các kỹ thuật đó, hiểu sơ qua, nhưng đừng để chúng làm bạn chệch hướng.**
- **Tiếp tục tập trung vào mục tiêu lớn: học cách xây dựng ứng dụng mạnh mẽ, thú vị và ấn tượng với LangGraph.**

Nội dung bạn sẽ học ở phần này

Chúng ta sẽ tìm hiểu **những kỹ thuật liên quan đến quản lý bộ nhớ trong ứng dụng LangGraph**, cụ thể:

1. Tối ưu hóa chi phí sử dụng LLM

Bạn sẽ học một vài phương pháp để:

- Giảm lượng **token** tiêu tốn trong các lần gọi LLM
- Giúp ứng dụng hoạt động **hiệu quả hơn** và **kinh tế hơn**.

2. Giới thiệu về bộ nhớ dài hạn (Long-term Memory)

Chúng ta sẽ bắt đầu thảo luận về cách:

- **Lưu trữ bộ nhớ** (memory) của ứng dụng vào **các cơ sở dữ liệu bên ngoài**
- **Ghi nhớ lâu dài** những thông tin quan trọng thay vì chỉ lưu trong phiên làm việc ngắn hạn.

Một lời nhắc quan trọng

Như tôi đã nhấn mạnh:

- Đây là những bài học rất thú vị và hữu ích.
- **Nhưng**, xin bạn **không dành quá nhiều thời gian thực hành lúc này**.
- **Hiểu khái niệm, nắm bắt nguyên lý, và tiếp tục tiến lên**.
- Khi bắt tay vào xây dựng những ứng dụng thực sự, bạn sẽ có dịp quay lại và khai thác những kỹ thuật này **đúng thời điểm**.

Hãy nhớ rằng: việc xây dựng một ứng dụng mạnh mẽ, thú vị và chuyên nghiệp đòi hỏi bạn phải **biết chọn lọc thông tin để tập trung**.

Đừng để những chi tiết phụ đánh lạc hướng bạn khỏi mục tiêu lớn hơn!

Tác giả: Đỗ Ngọc Tú
Công Ty Phần Mềm VHTSoft

Tối Ưu Chi Phí và Quản Lý Bộ Nhớ Khi Xây Dựng Ứng Dụng AI Với OpenAI

Nếu bạn từng đọc các chương trước của chúng tôi, bạn sẽ biết rằng việc thực hành các bài tập sử dụng OpenAI API không hề tốn kém.

Thông thường, bạn chỉ tốn khoảng **5 đô la** cho toàn bộ bootcamp trước, và con số này sẽ tương tự trong bootcamp này.

Tại sao chi phí lại thấp?

Vì chúng ta đang xây dựng các ứng dụng nhỏ gọn, số lượng người dùng hạn chế, tiêu thụ tài nguyên và token ở mức tối thiểu.

Tuy nhiên, khi bạn chuyển sang giai đoạn **sản xuất thực tế** với nhiều người dùng và ứng dụng lớn, chi phí có thể tăng **đáng kể**.

Bài học hôm nay sẽ giúp bạn:

- Hiểu vì sao phải kiểm soát chi phí khi xây dựng AI agent.
- Biết cách quản lý bộ nhớ ngắn hạn (**short-term memory**) để tiết kiệm token.
- Học hai kỹ thuật chính giúp giảm chi phí vận hành chatbot.

Chi Phí Và Bộ Nhớ Trong Ứng Dụng AI

Khi vận hành AI agent:

- **Chi phí** thường gắn liền với **quản lý bộ nhớ**.
- Bộ nhớ càng lớn → **Tốn nhiều token hơn** → **Chi phí cao hơn**.

Ví dụ:

- Nếu cuộc hội thoại có 100 tin nhắn, mỗi lần bạn gửi yêu cầu mới, bạn cũng phải gửi lại cả 100 tin nhắn trước đó.
- Điều này giống như việc bạn **đeo một ba lô** đầy tin nhắn trong suốt cuộc trò chuyện. Ba lô càng nặng → Chi phí càng cao.

OpenAI hay Open-Source?

Bạn có thể tự hỏi:

- "Tại sao không dùng mô hình mã nguồn mở để giảm chi phí?"

Bạn **có thể** làm vậy, đặc biệt ở giai đoạn thử nghiệm (beta, demo).

☐ Nhưng khi làm việc ở cấp độ chuyên nghiệp, **OpenAI** vẫn đang là tiêu chuẩn hiện tại (~99% doanh nghiệp dùng OpenAI).

Vì vậy, chúng tôi khuyên bạn nên thành thạo với OpenAI models.

Làm Thế Nào Để Giảm Chi Phí Bộ Nhớ?

Chúng ta sẽ học 2 kỹ thuật chính để **giảm dung lượng "ba lô"** của chatbot:

Kỹ thuật 1: Giới hạn số lượng tin nhắn lưu trong bộ nhớ

- Thay vì lưu **toàn bộ** cuộc hội thoại, chỉ lưu **khoảng 10 tin nhắn gần nhất**.
- Ví dụ:
 - Cuộc trò chuyện có 100 tin nhắn.
 - Chỉ lưu 10 tin nhắn mới nhất để gửi kèm với mỗi câu hỏi mới.

Ưu điểm: Giảm mạnh chi phí token.

Nhược điểm: Độ chính xác của chatbot có thể giảm nhẹ.

Kỹ thuật 2: Sử dụng bản tóm tắt thay cho toàn bộ tin nhắn

(Sẽ được học ở bài tiếp theo)

- Tóm tắt cuộc hội thoại thành 1 đoạn văn ngắn.
- Gửi đoạn tóm tắt thay vì gửi toàn bộ nội dung tin nhắn.

Ưu điểm: Bộ nhớ cực nhẹ, tiết kiệm chi phí.

Nhược điểm: Chatbot có thể không nhớ chi tiết đầy đủ.

Cách Thực Hiện Kỹ Thuật 1 Trong Thực Tế

Trong bài tập hôm nay, bạn sẽ thực hành:

- Tạo một danh sách mô phỏng các tin nhắn trước đó.

```
# Giả lập bộ nhớ tin nhắn
messages = [
    {"role": "user", "content": "Xin chào!"},
    {"role": "assistant", "content": "Chào bạn! Tôi có thể giúp gì?"},
    {"role": "user", "content": "Bạn có thể giải thích LangGraph không?"},
    {"role": "assistant", "content": "LangGraph là một framework xây dựng AI workflows."},
    {"role": "user", "content": "Làm sao để quản lý bộ nhớ trong LangGraph?"}
]
```

- Dùng **Python function** để xóa tất cả tin nhắn cũ, chỉ giữ lại **2 tin nhắn mới nhất** trong bộ nhớ.

```
def keep_last_two_messages(messages):  
    """  
    Hàm xóa tin nhắn cũ, chỉ giữ lại 2 tin nhắn mới nhất.  
    """  
  
    return messages[-2:] # Cắt danh sách, lấy 2 phần tử cuối
```

- Khi gửi yêu cầu tới ChatGPT, chỉ kèm **2 tin nhắn cuối cùng**.

```
import openai  
  
# Cập nhật danh sách tin nhắn, chỉ giữ lại 2 tin mới nhất  
messages_to_send = keep_last_two_messages(messages)  
  
# Gửi request tới ChatGPT  
response = openai.ChatCompletion.create(  
    model="gpt-4",  
    messages=messages_to_send  
)  
  
print(response['choices'][0]['message']['content'])
```

Kết quả:

Bộ nhớ (backpack) nhẹ hơn → Chi phí token thấp hơn.

Lưu Ý Khi Áp Dụng

- Những kỹ thuật giảm bộ nhớ này **rất hữu ích** ở giai đoạn:
 - Thử nghiệm (beta).
 - Demo với khách hàng.
- Khi vào sản xuất thực tế (production), bạn cần cân nhắc kỹ giữa:

Tác giả: Đỗ Ngọc Tú
Công Ty Phần Mềm VHTSoft

Tối ưu hóa bộ nhớ ngắn hạn bằng cách tóm tắt hội thoại

Trong bài học trước, bạn đã học cách **giảm số lượng tin nhắn trong bộ nhớ ngắn hạn** để tiết kiệm token khi làm việc với mô hình ngôn ngữ.

Trong bài học này, chúng ta sẽ tìm hiểu một **phương pháp thay thế: Tạo bản tóm tắt hội thoại** thay vì lưu toàn bộ danh sách tin nhắn.

Tóm tắt hội thoại: Ý tưởng chính

- Thay vì lưu trữ toàn bộ tin nhắn, chúng ta sẽ **tóm tắt nội dung cuộc trò chuyện**.
- Bộ nhớ ngắn hạn ("ba lô" mà chatbot mang theo) sẽ chỉ chứa **bản tóm tắt**, thay vì toàn bộ các tin nhắn.

Ưu điểm:

- Bản tóm tắt nhỏ gọn hơn nhiều so với danh sách tất cả các tin nhắn → **tiết kiệm token**.

Nhược điểm:

- Mức độ **chính xác** của phản hồi sẽ **giảm nhẹ**, vì thông tin chi tiết có thể bị mất trong quá trình tóm tắt.

Khi nào nên dùng tóm tắt hội thoại?

- **Giai đoạn phát triển / demo / beta**: Ưu tiên tiết kiệm chi phí và token → Dùng tóm tắt.
- **Giai đoạn sản phẩm chính thức (production)**: Ưu tiên hiệu suất và độ chính xác → Có thể dùng toàn bộ tin nhắn.

Ghi nhớ:

Hiệu suất và chi phí luôn cần **cân bằng** tùy vào mục tiêu dự án của bạn.

Cách triển khai

a. Cấu trúc State mới

Trong `state` của ứng dụng, ngoài khóa mặc định `messages`, bạn sẽ thêm một khóa mới:

```
state = {  
  "messages": [...],  
  "summary": "..."  
}
```

- `messages`: Lưu danh sách tin nhắn.
- `summary`: Lưu bản tóm tắt nội dung cuộc trò chuyện.

b. Logic hoạt động

1. **Bắt đầu cuộc trò chuyện** như bình thường.
2. Khi số lượng tin nhắn trong bộ nhớ **vượt quá ngưỡng** (ví dụ: 6 tin nhắn), **tạo hoặc cập nhật bản tóm tắt**.
3. Nếu chưa vượt ngưỡng, tiếp tục hội thoại bình thường.

Điều kiện:

Nếu `len(messages) > 6` → tạo / cập nhật bản tóm tắt.

c. Các thành phần cần lập trình

- **Conditional Edge**: Kiểm tra số lượng tin nhắn để quyết định có cần tóm tắt hay không.
- **Function để tóm tắt hội thoại**: Lấy danh sách tin nhắn, tạo ra một bản tóm tắt.
- **Function gửi yêu cầu tới ChatGPT**:
 - Nếu đã có `summary` → dùng tóm tắt làm context.
 - Nếu chưa có `summary` → dùng danh sách tin nhắn.

Lưu ý khi thực thi

- **Mỗi hội thoại / user / session** phải gắn với **một thread ID** riêng.
- Khi gửi request, bạn cần truyền `thread_id` để phân biệt từng cuộc trò chuyện khác nhau.

Tổng kết

- Đây là **cách thứ hai** để giảm lượng token tiêu thụ.
- Khác với cách giảm số lượng tin nhắn, ở đây ta **chuyển đổi toàn bộ nội dung thành một bản tóm tắt**.
- Bạn có thể quay lại tài liệu chi tiết và ví dụ khi cần áp dụng kỹ thuật này vào những ứng dụng thực tế.

Ghi nhớ

“Hiệu suất vs Chi phí luôn cần cân bằng. Hiểu kỹ các kỹ thuật ngay từ giai đoạn phát triển sẽ giúp bạn xây dựng các ứng dụng AI tối ưu và chuyên nghiệp hơn.”

Giới thiệu về Bộ nhớ Ngoài (Persistent Memory) với LangGraph

- Trong chương trình đến hiện tại, chúng ta mới chỉ làm việc với **bộ nhớ ngắn hạn** (short-term memory).
- Ứng dụng có thể ghi nhớ cuộc hội thoại đang diễn ra, nhưng **khi đóng ứng dụng và mở lại**, mọi thông tin sẽ **mất**.
- Hôm nay, chúng ta sẽ khám phá một khái niệm mới: **Bộ nhớ dài hạn** (Long-Term Memory) hay còn gọi là **Bộ nhớ ngoài** (Persistent Memory).

Tầm quan trọng của Bộ nhớ Ngoài

- Bộ nhớ ngoài giúp ứng dụng AI:
 - Ghi nhớ người dùng** qua nhiều phiên làm việc.
 - Lưu trữ sở thích, dữ liệu, lịch sử** người dùng.
- Đây là một **bước tiến rất quan trọng** trong việc xây dựng các ứng dụng AI thông minh hơn.

“ ✂ Ghi nhớ: Hôm nay chỉ là **giới thiệu ban đầu**, chi tiết sâu hơn sẽ học trong các bài học tới.

Mục tiêu bài học

Bạn sẽ học cách:

- Tạo một ứng dụng AI có **bộ nhớ ngoài**.
- Lưu trữ bộ nhớ** trong một **cơ sở dữ liệu SQLite** thay vì chỉ lưu trong RAM.

Các bước thực hiện

1. Cài đặt SQLite

- Xác nhận đã cài đặt **SQLite 3** trên máy.
- Nếu chưa có, làm theo hướng dẫn để cài đặt.

2. Tạo cơ sở dữ liệu bên ngoài

- Dùng **lệnh terminal** trong notebook (bằng `!`) để:
 - Tạo thư mục `state_db/`.
 - Tạo cơ sở dữ liệu `external_memory.db` trong thư mục này.

“ Lưu ý: Dấu `!` trong notebook có nghĩa là **chạy lệnh Terminal** ngay trong notebook.

3. Kết nối với cơ sở dữ liệu

- Sử dụng đối tượng **connection** (`conn`) để kết nối tới database.
- **Đường dẫn** tới database sẽ là: `state_db/external_memory.db`.

4. Thiết lập LangGraph với SQLite

- Trước đây ta dùng `MemorySaver` cho bộ nhớ ngắn hạn.
- Giờ ta sử dụng **SQLiteSaver** để:
 - Ghi nhớ cuộc hội thoại.
 - Lưu vào cơ sở dữ liệu ngoài (external memory).

```
from langgraph.checkpoints.sqlite import SQLiteSaver

checkpoint = SQLiteSaver(conn=conn)
```

`checkpoint` sẽ lưu giữ bộ nhớ trong file SQLite đã tạo.

Ứng dụng thực tế

- Ứng dụng chatbot trước đây sẽ được chỉnh sửa:
 - Không lưu tạm trên RAM nữa.
 - Mà lưu luôn trên cơ sở dữ liệu.
- Nếu bạn **tắt** và **mở lại notebook**, chatbot **vẫn nhớ** cuộc trò chuyện trước đó!

Ghi chú quan trọng

- **KHÔNG cần luyện tập sâu** với bài này ngay bây giờ.
- **Ghi nhớ ý tưởng**: ứng dụng có thể lưu bộ nhớ vào database.
- Khi xây dựng **ứng dụng nâng cao** sau này, bạn sẽ cần dùng kỹ thuật này nhiều hơn.

Tổng kết nhanh

- **Bộ nhớ ngoài** = Lưu trữ cuộc trò chuyện lâu dài.
- **SQLite** = Công cụ lưu trữ dữ liệu nhẹ, dễ dùng.

- **LangGraph** = Cung cấp module `SQLiteSaver` để kết nối và lưu dữ liệu dễ dàng.

Tác giả: Đỗ Ngọc Tú
Công Ty Phần Mềm VHTSoft

Tóm tắt chương

Hãy nhớ những gì tôi đã nói trong tài liệu về LangGraph, bạn sẽ gặp rất nhiều thứ mà chúng ta có thể gọi là "cạm bẫy".

Những cạm bẫy cần tránh khi học LangGraph

Chúng tôi gọi chúng là cạm bẫy vì chúng thực sự là những yếu tố gây xao nhãng lớn. Mục tiêu của bạn khi học LangGraph là để tạo ra những agent AI ấn tượng, mạnh mẽ và thú vị. Tuy nhiên, khi đọc tài liệu LangGraph, bạn sẽ gặp rất nhiều yếu tố gây xao nhãng trước khi đạt được mục tiêu đó.

Theo chúng tôi, điều này có thể gây bất lợi vì:

- Nếu dành quá nhiều thời gian cho những yếu tố phụ này, động lực học tập của bạn có thể giảm sút
- Bạn sẽ chỉ tập trung vào những thứ thứ yếu thay vì xây dựng ứng dụng thực tế
- Bạn không có cơ hội hoàn thiện ứng dụng bằng các kỹ thuật nâng cao

Lời khuyên học tập hiệu quả

Khi tiếp cận tài liệu LangGraph, bạn sẽ gặp rất nhiều cạm bẫy như vậy. Do đó, lời khuyên của chúng tôi là:

1. Đừng để bị xao nhãng
2. Nắm vững các khái niệm chính trước
3. Có thể quay lại nghiên cứu sâu các kỹ thuật sau khi đã xây dựng ứng dụng cơ bản

Ví dụ: Khi làm ứng dụng nâng cao, bạn có thể:

- Áp dụng kỹ thuật tiết kiệm chi phí đã học
- Sử dụng bộ nhớ ngoài với database
- Tối ưu hóa hệ thống

Các kỹ thuật đã học

1. **3 kỹ thuật đơn giản giảm token usage:**
 - Giảm số message lưu trong bộ nhớ ngắn hạn
 - Giảm số token sử dụng
 - Sử dụng các hàm reducer trong Python
2. **Phương pháp thay thế:**
 - Lưu bản tóm tắt conversation thay vì toàn bộ message
 - Có thể đặt điều kiện (vd: tạo summary khi có >6 messages)
3. **Bộ nhớ ngoài/lâu dài:**

- Ví dụ đơn giản với SQLite database
- Các kỹ thuật persistent memory

Lưu ý quan trọng: Đừng dừng lại quá lâu ở các kỹ thuật này! Hãy tiếp tục tiến lên và chỉ quay lại khi bạn thực sự cần áp dụng chúng để cải thiện ứng dụng cụ thể.

Chủ đề quan trọng sắp tới: Human-in-the-loop

Trong phần tiếp theo, chúng ta sẽ học về khái niệm cực kỳ quan trọng trong LangGraph: **Human-in-the-loop**. Bạn sẽ thấy:

- Agent AI và ứng dụng LLM có thể cải thiện vượt bậc nhờ feedback từ người dùng
- Trong nhiều trường hợp, ứng dụng có thể tự học mà không cần bạn can thiệp nhiều
- Đây là bước tiến quan trọng trong quá trình học tập

Lời nhắc: Hãy tập trung cao độ cho phần tiếp theo này vì nó sẽ thực sự nâng tầm khả năng phát triển agent AI của bạn! LangGraph thực sự là framework mạnh mẽ nhưng để học hiệu quả, bạn cần:

1. Xác định rõ mục tiêu (build agent cụ thể nào)
2. Học theo cách "top-down": dùng trước, tối ưu sau
3. Đừng sa đà vào micro-optimization quá sớm
4. Tập trung vào các pattern quan trọng: agent collaboration, human feedback, memory management

Tác giả: Đỗ Ngọc Tú
Công Ty Phần Mềm VHTSoft