SYLLABUS

NGHIỆP VỤ QUY HOẠCH VÀ XÂY DỰNG AGENTS TRÊN HỆ SINH THÁI CLAUDE COWORK

Phiên bản 1.0

Ngày ban hành: 20/02/2026

GIỚI THIỆU KHÓA HỌC

Tài liệu này là syllabus đào tạo nghiệp vụ dành cho các chuyên viên, nhà quản lý và kỹ thuật viên muốn ứng dụng hệ sinh thái Claude Cowork (Anthropic) vào công việc thực tế. Trong bối cảnh AI đang chuyển dịch mạnh mẽ từ mô hình chatbot đơn giản sang các hệ thống agent tự chủ (agentic systems), việc nắm vững phương pháp quy hoạch và xây dựng agents trở thành năng lực cốt lõi của người làm việc với AI.

Claude Cowork là ứng dụng desktop agent cho phép người dùng không cần lập trình vẫn có thể thiết kế, triển khai và vận hành các agent thực hiện công việc đa bước một cách tự chủ — từ xử lý tài liệu văn phòng, phân tích dữ liệu, đến điều phối quy trình nghiệp vụ phức tạp. Khóa học này được xây dựng dựa trên tài liệu kỹ thuật chính thức của Anthropic, bao gồm nghiên cứu “Building Effective Agents” và tiêu chuẩn Agent Skills, kết hợp với các best practices từ cộng đồng AI engineering quốc tế.

Đối tượng học viên

Chuyên viên nghiệp vụ, quản lý dự án, kỹ thuật viên AI không chuyên lập trình, và những ai muốn tự động hóa công việc bằng agent AI trong môi trường doanh nghiệp.

Kết quả đầu ra kỳ vọng

Sau khi hoàn thành khóa học, học viên có thể:

1. Hiểu rõ kiến trúc và triết lý thiết kế agent của Anthropic.
2. Quy hoạch được các luồng công việc agent phức tạp, phù hợp với nghiệp vụ thực tế.
3. Tự tay xây dựng và triển khai các **Skills** (kỹ năng) tùy chỉnh cho agent.
4. Vận hành, giám sát và tối ưu hóa hiệu suất của agents trong môi trường làm việc thực tế.

MỤC LỤC

Chương 1: Tổng quan hệ sinh thái Agentic AI & Claude Cowork

1. Giới thiệu: Sự chuyển dịch từ Chatbot sang Agent

Trong bối cảnh phát triển nhanh chóng của Trí tuệ Nhân tạo (AI), chúng ta đang chứng kiến một sự chuyển dịch đáng kể từ các hệ thống chatbot truyền thống sang các AI agent có khả năng tự chủ và thực hiện hành động phức tạp. Chatbot thường được thiết kế để tương tác và cung cấp thông tin dựa trên các quy tắc hoặc kịch bản được xác định trước, chủ yếu tập trung vào việc tạo ra phản hồi văn bản [1]. Ngược lại, AI agent là các hệ thống AI tiên tiến có khả năng hiểu ngữ cảnh, đưa ra quyết định, lập kế hoạch và thực hiện các chuỗi hành động để đạt được mục tiêu cụ thể với sự giám sát hạn chế [2]. Sự khác biệt cốt lõi nằm ở khả năng tự chủ và thực thi: trong khi chatbot chỉ trả lời câu hỏi, agent có thể thực sự *làm việc* [3].

Sự chuyển dịch này mở ra những tiềm năng to lớn cho việc tự động hóa các tác vụ phức tạp, từ hỗ trợ khách hàng đến phát triển phần mềm. Anthropic, một trong những công ty hàng đầu trong lĩnh vực AI, đã và đang đóng vai trò quan trọng trong việc định hình và thúc đẩy sự phát triển của Agentic AI thông qua các nghiên cứu, sản phẩm như Claude Cowork và triết lý thiết kế độc đáo của mình.

2. Agentic AI là gì?

Agentic AI (hay AI tác nhân) là một dạng trí tuệ nhân tạo tiên tiến tập trung vào khả năng ra quyết định và hành động tự chủ. Một AI agent là một hệ thống học máy có thể hoàn thành một mục tiêu cụ thể với sự giám sát hạn chế. Nó bao gồm các AI agent – các mô hình học máy được trang bị khả năng hiểu ngữ cảnh, đưa ra quyết định và thực hiện hành động để đạt được mục tiêu [1].

Các đặc điểm chính của Agentic AI bao gồm:

- **Tự chủ (Autonomy)**: Khả năng hoạt động độc lập mà không cần sự can thiệp liên tục của con người.

- **Lập kế hoạch (Planning)**: Khả năng phân tích mục tiêu, chia nhỏ thành các tác vụ con và xác định trình tự hành động để đạt được mục tiêu đó.
- **Sử dụng công cụ (Tool Use)**: Khả năng tích hợp và sử dụng các công cụ bên ngoài (API, cơ sở dữ liệu, phần mềm khác) để mở rộng khả năng của mình.
- **Bộ nhớ (Memory)**: Khả năng lưu trữ và truy xuất thông tin từ các tương tác trước đó để cải thiện hiệu suất trong tương lai.
- **Phản hồi và Tự sửa lỗi (Feedback and Self-correction)**: Khả năng đánh giá kết quả hành động, học hỏi từ sai lầm và điều chỉnh kế hoạch khi cần thiết.

Anthropic phân loại các hệ thống này thành **Agentic Systems**, và chia nhỏ hơn thành **Workflows** và **Agents** [4]:

- **Workflows**: Là các hệ thống nơi các Mô hình Ngôn ngữ Lớn (LLM) và các công cụ được điều phối thông qua các đường dẫn mã được xác định trước. Chúng cung cấp tính dự đoán và nhất quán cho các tác vụ được định nghĩa rõ ràng.
- **Agents**: Là các hệ thống nơi LLM tự động điều hướng các quy trình và việc sử dụng công cụ của chúng, duy trì quyền kiểm soát cách chúng hoàn thành nhiệm vụ. Agents phù hợp hơn khi cần sự linh hoạt và ra quyết định dựa trên mô hình ở quy mô lớn.

3. Kiến trúc Agentic System

Kiến trúc của một hệ thống Agentic AI thường được thiết kế để phân tách một mục tiêu lớn thành các tác vụ con nhỏ hơn và gán mỗi tác vụ con cho một agent chuyên dụng với một kỹ năng cụ thể [5]. Các thành phần chính trong kiến trúc Agentic System bao gồm:

- **Lớp Công cụ (Tool Layer)**: Cung cấp các công cụ và API mà agent có thể sử dụng để tương tác với môi trường bên ngoài, truy xuất thông tin hoặc thực hiện hành động.
- **Lớp Suy luận (Reasoning Layer)**: Chịu trách nhiệm cho việc lập kế hoạch, ra quyết định và suy luận logic của agent.
- **Lớp Hành động (Action Layer)**: Thực thi các hành động được quyết định bởi lớp suy luận, thường thông qua việc gọi các công cụ trong lớp công cụ.

- **Thành phần Điều phối (Orchestration Components)**: Quản lý luồng công việc giữa các agent, phân phối tác vụ và tổng hợp kết quả.
- **Lớp Quan sát (Observability Layer)**: Giám sát hoạt động của agent, thu thập dữ liệu về hiệu suất và giúp gỡ lỗi.

Anthropic đã xác định một số mẫu kiến trúc (workflows) phổ biến cho các hệ thống agentic, từ đơn giản đến phức tạp [4]:

Mẫu kiến trúc	Mô tả	Khi nào sử dụng	Ví dụ
Augmented LLM	LLM được tăng cường với khả năng truy xuất, công cụ và bộ nhớ.	Khối xây dựng cơ bản cho mọi hệ thống agentic.	Claude có thể tự tạo truy vấn tìm kiếm, chọn công cụ phù hợp.
Prompt Chaining	Phân tách nhiệm vụ thành chuỗi các bước, mỗi LLM xử lý đầu ra của bước trước.	Nhiệm vụ có thể phân tách rõ ràng thành các tác vụ con cố định, cần độ chính xác cao hơn.	Tạo nội dung marketing rồi dịch; Viết dàn ý rồi phát triển nội dung.
Routing	Phân loại đầu vào và định tuyến đến tác vụ chuyên biệt.	Nhiệm vụ phức tạp với các danh mục riêng biệt, cần xử lý chuyên biệt.	Định tuyến yêu cầu hỗ trợ khách hàng (hoàn tiền, kỹ thuật); Định tuyến câu hỏi đến các mô hình Claude khác nhau (Haiku cho dễ, Sonnet cho khó).
Parallelization	LLM làm việc đồng thời trên các tác vụ, tổng hợp kết quả.	Cần tăng tốc độ xử lý hoặc cần nhiều góc nhìn/nỗ lực để có kết quả tin cậy.	**Sectioning**: Guardrails (một model xử lý query, một model sàng lọc nội dung); **Voting**: Đánh giá lỗ hổng mã, đánh giá nội dung.
Orchestrator-workers	LLM trung tâm phân tách, ủy quyền cho LLM worker và tổng hợp kết quả.	Nhiệm vụ phức tạp không thể dự đoán trước các tác vụ con.	Agent mã hóa thay đổi nhiều tệp; Agent tìm kiếm thông tin từ nhiều nguồn.
Evaluator-optimizer	Một LLM tạo phản hồi, một LLM khác đánh giá và cung cấp phản hồi trong vòng lặp.	Có tiêu chí đánh giá rõ ràng, tinh chỉnh lặp lại mang lại giá trị.	Dịch văn học (LLM dịch, LLM đánh giá); Tìm kiếm phức tạp cần nhiều vòng lặp.

4. Triết lý Simple & Composable của Anthropic

Anthropic nhấn mạnh rằng các triển khai Agentic AI thành công nhất thường sử dụng các mẫu đơn giản, có thể kết hợp (Simple & Composable) thay vì các framework phức tạp [4]. Triết lý này được thể hiện qua ba nguyên tắc cốt lõi khi xây dựng agent:

1. **Duy trì sự đơn giản trong thiết kế của agent**: Tránh thêm độ phức tạp không cần thiết. Bắt đầu với giải pháp đơn giản nhất có thể và chỉ tăng độ phức tạp khi nó chứng minh được việc cải thiện kết quả.

2. **Ưu tiên tính minh bạch**: Hiển thị rõ ràng các bước lập kế hoạch và suy luận của agent. Điều này giúp gỡ lỗi, hiểu hành vi của agent và xây dựng niềm tin cho người dùng.

3. **Thiết kế cẩn thận giao diện agent-máy tính (ACI)**: Thông qua tài liệu công cụ kỹ lưỡng và thử nghiệm. Giao diện công cụ cần rõ ràng, dễ hiểu cho LLM, giống như việc viết docstring tốt cho một nhà phát triển mới.

Triết lý này khuyến khích các nhà phát triển bắt đầu bằng cách sử dụng trực tiếp các API của LLM, vì nhiều mẫu có thể được triển khai chỉ với vài dòng mã. Nếu sử dụng framework, cần đảm bảo hiểu rõ mã nguồn bên dưới để tránh các giả định sai lầm [4].

5. Claude Cowork: Sự hiện thực hóa Agentic AI của Anthropic

Claude Cowork là một sản phẩm của Anthropic, được giới thiệu như một bước tiến quan trọng trong việc đưa sức mạnh thực thi của AI đến mọi người [3]. Ban đầu, Claude Code đã thay đổi cách các nhà phát triển xây dựng phần mềm. Giờ đây, Cowork mang khả năng thực thi tương tự đó đến với người dùng phổ thông, cho phép Claude không chỉ trả lời câu hỏi mà còn thực sự *làm việc* cùng với họ [3].

Cowork sử dụng cùng kiến trúc agentic đã cung cấp năng lượng cho Claude Code, giờ đây có thể truy cập được trong Claude Desktop và không cần mở terminal [6]. Điều này cho phép người dùng trải nghiệm khả năng của Agentic AI trong các quy trình làm việc đa bước, bao gồm tổng hợp nghiên cứu, tạo tài liệu và trích xuất dữ liệu [3]. Cowork thể hiện rõ ràng sự chuyển dịch từ mô hình chatbot chỉ cung cấp thông tin sang mô hình agent có khả năng thực hiện các tác vụ phức tạp, tương tác với môi trường máy tính của người dùng.

6. Best Practices trong xây dựng Agentic AI

Dựa trên kinh nghiệm của Anthropic và các nhà phát triển, dưới đây là một số best practices quan trọng khi xây dựng Agentic AI [4]:

1. **Bắt đầu đơn giản và tăng độ phức tạp dần dần**: Luôn tìm kiếm giải pháp đơn giản nhất trước. Chỉ thêm các thành phần agentic (như công cụ, bộ nhớ, lập kế hoạch) khi có bằng chứng rõ ràng rằng chúng cải thiện hiệu suất hoặc giải quyết một vấn đề cụ thể. Tránh sử dụng các framework phức tạp nếu một vài dòng mã đơn giản có thể đạt được mục tiêu tương tự.

2. **Thiết kế công cụ rõ ràng và mạnh mẽ (Robust Tool Design)**: Công cụ là trái tim của một agent. Tài liệu hóa công cụ một cách kỹ lưỡng, bao gồm ví dụ sử dụng, trường hợp biên, yêu cầu định dạng đầu vào và ranh giới rõ ràng với các công cụ khác. Đảm bảo rằng các tham số công cụ được đặt tên và mô tả một cách trực quan. Thử nghiệm rộng rãi cách LLM sử dụng công cụ của bạn và lặp lại để cải thiện. Áp dụng nguyên tắc Poka-yoke (chống sai sót) bằng cách thay đổi các đối số để khó mắc lỗi hơn (ví dụ: luôn yêu cầu đường dẫn tuyệt đối thay vì tương đối).

3. **Quản lý ngữ cảnh hiệu quả (Effective Context Management)**: Ngữ cảnh là một tài nguyên quan trọng nhưng hữu hạn đối với AI agent. Phát triển các chiến lược để quản lý và sắp xếp ngữ cảnh một cách hiệu quả, đảm bảo agent có đủ thông tin cần thiết mà không bị quá tải. Điều này bao gồm việc chọn lọc thông tin liên quan, tóm tắt các đoạn dài và sử dụng bộ nhớ ngoài khi cần.

4. **Kiểm tra và lặp lại liên tục (Continuous Testing and Iteration)**: Agentic AI là một lĩnh vực đang phát triển nhanh chóng. Việc thử nghiệm liên tục trong môi trường sandbox, đặc biệt là với các trường hợp biên và kịch bản lỗi, là rất quan trọng. Sử dụng các công cụ đánh giá tự động và phản hồi của con người để cải thiện agent theo thời gian.

5. **Tích hợp giám sát và can thiệp của con người (Human Oversight and Intervention)**: Mặc dù agent có khả năng tự chủ, việc có các điểm kiểm tra (checkpoints) để con người có thể xem xét, cung cấp phản hồi hoặc can thiệp khi cần là rất quan trọng. Điều này giúp đảm bảo an toàn, độ chính xác và sự phù hợp với mục tiêu của người dùng, đặc biệt trong các tác vụ quan trọng.

7. Step-by-step Guide: Xây dựng một Agent đơn giản với Prompt Chaining

Để minh họa triết lý Simple & Composable, chúng ta sẽ xem xét một hướng dẫn từng bước để xây dựng một agent đơn giản sử dụng mẫu Prompt Chaining để tạo và dịch nội dung marketing:

Mục tiêu: Tạo một đoạn văn marketing ngắn bằng tiếng Anh, sau đó dịch nó sang tiếng Việt.

Bước 1: Xác định các tác vụ con

- Tác vụ 1: Tạo nội dung marketing bằng tiếng Anh.
- Tác vụ 2: Dịch nội dung marketing từ tiếng Anh sang tiếng Việt.

Bước 2: Thiết kế Prompt cho từng tác vụ

- **Prompt cho Tác vụ 1 (Marketing Content Generation)**:

```
Bạn là một chuyên gia marketing. Hãy viết một đoạn văn ngắn (khoảng 50 từ) để quảng bá sản phẩm "AI Agent Platform" mới của chúng tôi. Sản phẩm này giúp tự động hóa các tác vụ phức tạp và tăng cường hiệu suất làm việc. Đoạn văn cần hấp dẫn và tập trung vào lợi ích cho người dùng.
```

- **Prompt cho Tác vụ 2 (Translation)**:

```
Bạn là một chuyên gia dịch thuật. Hãy dịch đoạn văn tiếng Anh sau sang tiếng Việt một cách tự nhiên và chính xác, giữ nguyên ý nghĩa gốc:


[NỘI DUNG TIẾNG ANH TỪ TÁC VỤ 1]
```

Bước 3: Thực thi chuỗi Prompt

1. Gửi Prompt Tác vụ 1 đến LLM (ví dụ: Claude). LLM sẽ tạo ra một đoạn văn marketing tiếng Anh.
2. Lấy kết quả từ Tác vụ 1. Đây sẽ là `[NỘI DUNG TIẾNG ANH TỪ TÁC VỤ 1]`.

3. Chèn `[NỘI DUNG TIẾNG ANH TỪ TÁC VỤ 1]` vào Prompt Tác vụ 2.
4. Gửi Prompt Tác vụ 2 đã được điền đầy đủ đến LLM. LLM sẽ trả về đoạn văn marketing đã được dịch sang tiếng Việt.

Bước 4: Kiểm tra và tinh chỉnh

- Đánh giá chất lượng của đoạn văn marketing tiếng Anh gốc. Nếu cần, điều chỉnh Prompt Tác vụ 1.
- Đánh giá chất lượng của bản dịch tiếng Việt. Nếu cần, điều chỉnh Prompt Tác vụ 2 hoặc cung cấp thêm ví dụ trong prompt (few-shot prompting).

Đây là một ví dụ đơn giản, nhưng nó minh họa cách các tác vụ phức tạp có thể được chia nhỏ và xử lý tuần tự bởi các lệnh gọi LLM, với mỗi bước xây dựng dựa trên kết quả của bước trước.

8. Ví dụ thực tế (Use Cases)

Agentic AI đang được ứng dụng trong nhiều lĩnh vực, mang lại hiệu quả đáng kể:

1. **Hỗ trợ khách hàng tự động (Automated Customer Support)** [4]:
    - **Mô tả**: Các agent AI có thể xử lý các truy vấn của khách hàng, truy xuất thông tin từ cơ sở dữ liệu kiến thức, lịch sử đơn hàng và thậm chí thực hiện các hành động như hoàn tiền hoặc cập nhật trạng thái vé. Điều này vượt xa khả năng của chatbot truyền thống chỉ cung cấp câu trả lời tĩnh.
    - **Lợi ích**: Giảm tải cho đội ngũ hỗ trợ, cải thiện thời gian phản hồi, cung cấp dịch vụ $^{24}/_{7}$ và cá nhân hóa trải nghiệm khách hàng.
    - **Ví dụ**: Một agent có thể nhận yêu cầu “Tôi muốn đổi trả sản phẩm X”. Agent sẽ truy xuất lịch sử mua hàng của khách, kiểm tra chính sách đổi trả, tạo nhãn vận chuyển trả hàng và thông báo cho khách hàng về các bước tiếp theo, tất cả mà không cần sự can thiệp của con người.
2. **Agent mã hóa và phát triển phần mềm (Coding Agents and Software Development)** [4]:
    - **Mô tả**: Agent AI có thể giải quyết các tác vụ phát triển phần mềm phức tạp, từ việc sửa lỗi trong mã nguồn (bug fixing) đến việc thực hiện các thay đổi

lớn trên nhiều tệp. Chúng có thể đọc mô tả tác vụ, lập kế hoạch, thực hiện các thay đổi mã, chạy thử nghiệm và lặp lại dựa trên kết quả thử nghiệm.

- **Lợi ích**: Tăng tốc độ phát triển, giảm lỗi, tự động hóa các tác vụ lặp đi lặp lại và cho phép các nhà phát triển tập trung vào các vấn đề phức tạp hơn.
- **Ví dụ**: Anthropic đã phát triển các agent có khả năng giải quyết các tác vụ SWE-bench, liên quan đến việc chỉnh sửa nhiều tệp dựa trên mô tả tác vụ. Một agent có thể nhận yêu cầu "Thêm tính năng đăng nhập bằng Google vào ứng dụng web". Agent sẽ xác định các tệp cần thay đổi, viết mã, cập nhật cấu hình, chạy các bài kiểm tra và đề xuất một pull request hoàn chỉnh.

9. Các lỗi thường gặp và cách tránh

Khi phát triển Agentic AI, có một số lỗi phổ biến mà các nhà phát triển thường mắc phải:

1. **Quá phức tạp hóa thiết kế ngay từ đầu**: Cố gắng xây dựng một agent quá phức tạp với nhiều lớp suy luận và công cụ mà không có bằng chứng rõ ràng về sự cần thiết. Điều này dẫn đến khó gỡ lỗi, chi phí cao và hiệu suất không tối ưu.
    - **Cách tránh**: Bắt đầu với các mẫu đơn giản (như Prompt Chaining hoặc Routing) và chỉ tăng độ phức tạp khi có yêu cầu rõ ràng từ tác vụ hoặc khi các giải pháp đơn giản hơn không đủ. Tuân thủ triết lý Simple & Composable của Anthropic.
2. **Thiết kế công cụ kém hiệu quả**: Các công cụ không được mô tả rõ ràng, có các tham số không trực quan hoặc yêu cầu định dạng đầu vào phức tạp. Điều này khiến LLM khó sử dụng công cụ một cách chính xác.
    - **Cách tránh**: Coi việc thiết kế công cụ như việc thiết kế giao diện người dùng. Cung cấp tài liệu chi tiết, ví dụ sử dụng, và kiểm tra cách LLM tương tác với công cụ. Sử dụng nguyên tắc Poka-yoke để giảm thiểu lỗi.
3. **Quản lý ngữ cảnh không tối ưu**: Cung cấp quá nhiều hoặc quá ít ngữ cảnh cho LLM, hoặc không cập nhật ngữ cảnh một cách hiệu quả. Ngữ cảnh quá lớn có thể làm tăng chi phí và độ trễ, trong khi ngữ cảnh quá nhỏ có thể khiến agent thiếu thông tin cần thiết.

    - **Cách tránh**: Áp dụng các kỹ thuật quản lý ngữ cảnh như tóm tắt, lọc thông tin liên quan và sử dụng bộ nhớ ngoài. Đảm bảo rằng agent chỉ nhận được thông tin cần thiết cho tác vụ hiện tại.

4. **Thiếu cơ chế giám sát và phản hồi**: Triển khai agent mà không có cách nào để theo dõi hoạt động của nó, gỡ lỗi khi có vấn đề hoặc cho phép con người can thiệp. Điều này đặc biệt rủi ro với các agent tự chủ.

    - **Cách tránh**: Xây dựng các lớp quan sát (observability layer) và các điểm kiểm tra để con người có thể giám sát và cung cấp phản hồi. Đảm bảo agent có khả năng báo cáo lỗi và yêu cầu trợ giúp khi gặp bế tắc.

10. Kỹ thuật nâng cao: Prompt Engineering Tools

Anthropic nhấn mạnh rằng việc tối ưu hóa công cụ (tool) quan trọng không kém, thậm chí hơn cả việc tối ưu hóa prompt chính cho agent [4]. Dưới đây là một số kỹ thuật nâng cao trong việc thiết kế prompt cho công cụ:

- **Cung cấp đủ "không gian suy nghĩ" (Thinking Space)**: Đảm bảo mô hình có đủ token để "suy nghĩ" và lập kế hoạch trước khi thực hiện hành động. Điều này giúp mô hình tránh mắc kẹt vào các quyết định sai lầm.

- **Giữ định dạng gần với ngôn ngữ tự nhiên**: Khi xác định định dạng đầu vào/đầu ra cho công cụ, hãy ưu tiên các định dạng mà mô hình đã quen thuộc từ dữ liệu huấn luyện trên internet (ví dụ: markdown, JSON đơn giản) thay vì các định dạng quá phức tạp hoặc yêu cầu xử lý đặc biệt (như tính toán số dòng trong diff).

- **Loại bỏ "chi phí định dạng" (Formatting Overhead)**: Tránh các yêu cầu định dạng gây gánh nặng cho mô hình, chẳng hạn như yêu cầu mô hình phải đếm chính xác hàng nghìn dòng mã hoặc thoát chuỗi phức tạp trong JSON. Mục tiêu là làm cho việc sử dụng công cụ trở nên dễ dàng nhất có thể cho LLM.

- **Thiết kế ACI (Agent-Computer Interface) như HCI (Human-Computer Interface)**: Đặt mình vào vị trí của mô hình. Mô tả công cụ một cách rõ ràng, bao gồm mục đích, các tham số, ví dụ sử dụng, các trường hợp biên và ranh giới với các công cụ khác. Coi đây như việc viết tài liệu API chất lượng cao cho một nhà phát triển.

- **Sử dụng Poka-yoke (Chống sai sót)**: Thay đổi thiết kế của công cụ hoặc các tham số của nó để giảm thiểu khả năng mô hình mắc lỗi. Ví dụ, thay vì cho phép đường dẫn tương đối, yêu cầu luôn sử dụng đường dẫn tuyệt đối để tránh lỗi khi agent thay đổi thư mục làm việc.

11. Kết luận

Sự chuyển dịch từ chatbot sang Agentic AI đánh dấu một kỷ nguyên mới trong khả năng của trí tuệ nhân tạo, nơi các hệ thống không chỉ hiểu mà còn hành động một cách tự chủ để hoàn thành các mục tiêu phức tạp. Anthropic, với triết lý Simple & Composable và các sản phẩm như Claude Cowork, đang dẫn đầu trong việc phát triển các agent mạnh mẽ, minh bạch và đáng tin cậy. Bằng cách tuân thủ các best practices và kỹ thuật nâng cao trong thiết kế kiến trúc và công cụ, các nhà phát triển có thể khai thác tối đa tiềm năng của Agentic AI để tạo ra các giải pháp đột phá, tự động hóa công việc và nâng cao hiệu suất trong nhiều lĩnh vực.

12. Tài liệu tham khảo

[1] IBM. (n.d.). *What is Agentic AI?*. Retrieved from https://www.ibm.com/think/topics/agentic-ai [2] AWS. (n.d.). *What is Agentic AI?*. Retrieved from https://aws.amazon.com/what-is/agentic-ai/ [3] Anthropic. (2026, January 30). *The Future of AI at Work: Introducing Cowork*. Retrieved from https://www.anthropic.com/webinars/future-of-ai-at-work-introducing-cowork [4] Anthropic. (2024, December 19). *Building Effective AI Agents*. Retrieved from https://www.anthropic.com/research/building-effective-agents [5] Akka. (2025, August 5). *Agentic AI architecture 101: An enterprise guide*. Retrieved from https://akka.io/blog/agentic-ai-architecture [6] Claude Help Center. (n.d.). *Getting started with Cowork*. Retrieved from https://support.claude.com/en/articles/13345190-getting-started-with-cowork

> ***Khái niệm then chốt:*** *Agentic AI, Claude Cowork, Simple & Composable, Workflows, Agents, Prompt Chaining, Tool Use, Human Oversight*

Chương 2: Chuỗi Prompt (Prompt Chaining)

I. Giới thiệu

Trong phát triển AI agents, xử lý nhiệm vụ phức tạp bằng một prompt duy nhất thường dẫn đến kết quả không nhất quán. **Chuỗi Prompt (Prompt Chaining)** nổi lên như một giải pháp hiệu quả, đặc biệt trong hệ sinh thái Claude của Anthropic. Kỹ thuật này chia nhỏ nhiệm vụ lớn thành các bước nhỏ hơn, dễ quản lý, từ đó nâng cao độ tin cậy và hiệu quả của AI agents.

II. Định nghĩa Chuỗi Prompt

Chuỗi Prompt là kỹ thuật phân rã một nhiệm vụ phức tạp thành một chuỗi các bước tuần tự, nơi đầu ra của một lời gọi Large Language Model (LLM) trở thành đầu vào cho lời gọi LLM tiếp theo [1]. Khác với các cuộc hội thoại đa lượt, prompt chaining sử dụng các yêu cầu API riêng biệt, mỗi yêu cầu tập trung vào một phần cụ thể của nhiệm vụ [2].

Kỹ thuật này khác với "Chain of Thought" (CoT) prompting, vốn khuyến khích LLM tự suy luận từng bước trong một prompt duy nhất. Prompt chaining chủ động chia nhỏ quá trình suy luận và thực thi thành các bước riêng biệt, có thể kiểm soát. Lợi ích cốt lõi bao gồm tăng cường **độ chính xác**, cải thiện **sự rõ ràng** của hướng dẫn và đầu ra, và nâng cao **khả năng truy vết**, giúp dễ dàng xác định và khắc phục sự cố [1].

III. Tại sao và Khi nào nên sử dụng Chuỗi Prompt

Tại sao?

Chuỗi prompt mang lại nhiều lợi ích trong việc xây dựng các hệ thống AI đáng tin cậy:

- **Độ chính xác cao hơn:** Mỗi subtask nhận được sự tập trung hoàn toàn của mô hình, giảm thiểu lỗi và cải thiện chất lượng đầu ra [1].

- **Rõ ràng và dễ hiểu:** Các subtask đơn giản hơn cho phép hướng dẫn rõ ràng, dẫn đến đầu ra chính xác và dễ dự đoán [1].
- **Khả năng truy vết và gỡ lỗi:** Dễ dàng xác định và sửa lỗi ở từng mắt xích trong chuỗi, cho phép tinh chỉnh các bước riêng lẻ mà không ảnh hưởng đến toàn bộ quy trình [1].

Khi nào?

Chuỗi prompt đặc biệt hữu ích trong các tình huống:

- **Các nhiệm vụ đa bước:** Ví dụ như tổng hợp nghiên cứu, phân tích tài liệu phức tạp, hoặc tạo nội dung lặp đi lặp lại [1].
- **Nhiệm vụ có nhiều phép biến đổi:** Khi yêu cầu nhiều chuyển đổi dữ liệu, trích dẫn hoặc tuân thủ nhiều hướng dẫn khác nhau, chaining giúp ngăn Claude xử lý sai các bước [1].
- **Ưu tiên độ chính xác hơn độ trễ:** Khi chất lượng đầu ra là tối quan trọng, việc chấp nhận một chút độ trễ do nhiều lời gọi LLM là hợp lý để đảm bảo độ chính xác [3].

IV. Hướng dẫn từng bước (Step-by-Step Guide)

Để triển khai chuỗi prompt hiệu quả, chúng ta tuân theo các bước sau:

1. **Phân rã nhiệm vụ (Identify Subtasks):** Chia nhỏ nhiệm vụ tổng thể thành các bước riêng biệt, tuần tự. Ví dụ: Nghiên cứu → Lập dàn ý → Viết bản nháp → Chỉnh sửa → Định dạng [1].
2. **Thiết kế Prompt cho từng bước:** Mỗi subtask có một mục tiêu duy nhất, rõ ràng. Sử dụng system prompt để xác định vai trò cụ thể của Claude trong từng bước, cung cấp hướng dẫn chi tiết và ràng buộc đầu ra [2].
3. **Sử dụng XML Tags để chuyển giao (Hand-offs):** Anthropic khuyến nghị sử dụng các thẻ XML để đóng gói đầu ra từ một bước và truyền nó như một phần của đầu vào cho bước tiếp theo. Điều này tạo sự phân tách rõ ràng giữa hướng dẫn và dữ liệu, giúp Claude hiểu ngữ cảnh và tránh nhầm lẫn [1].

4. **Triển khai "Gate Checks" (Kiểm tra cổng):** Thêm các bước kiểm tra logic (programmatic checks) giữa các prompt để xác thực đầu ra của một bước trước khi chuyển sang bước tiếp theo [3]. Nếu một gate check thất bại, hệ thống có thể dừng, báo lỗi, hoặc kích hoạt logic sửa lỗi để đảm bảo chất lượng và tính đúng đắn của luồng công việc [2].

5. **Lặp lại và tinh chỉnh (Iterate):** Tinh chỉnh các prompt, điều chỉnh các gate check và tối ưu hóa luồng công việc dựa trên hiệu suất của Claude. Quá trình lặp lại này giúp cải thiện độ chính xác và hiệu quả của chuỗi [1].

V. Các Best Practices từ Anthropic

Anthropic đã đưa ra một số best practices quan trọng cho prompt chaining [1] [3]:

1. **Mỗi Prompt một nhiệm vụ:** Mỗi prompt trong chuỗi chỉ tập trung vào một mục tiêu duy nhất và rõ ràng, tránh nhồi nhét quá nhiều yêu cầu vào một prompt. Điều này giúp Claude tập trung và tạo ra kết quả chính xác hơn.

2. **Sử dụng System Prompts hiệu quả:** Tận dụng system prompt để gán vai trò cụ thể và hướng dẫn hành vi của Claude ở mỗi bước. System prompt cung cấp ngữ cảnh và ràng buộc quan trọng cho mô hình.

3. **Đóng gói đầu vào/đầu ra bằng XML Tags:** Đặt dữ liệu đầu vào và đầu ra trong các thẻ XML giúp Claude phân biệt rõ ràng giữa hướng dẫn và nội dung, cải thiện độ tin cậy khi chuyển giao dữ liệu giữa các bước.

4. **Triển khai Gate Checks:** Luôn thêm các bước kiểm tra logic mạnh mẽ giữa các prompt để xác thực kết quả trung gian, ngăn chặn lỗi lan truyền và đảm bảo tính toàn vẹn của chuỗi.

5. **Tối ưu hóa song song (Parallelization):** Đối với các subtask độc lập, xem xét chạy chúng song song để giảm tổng thời gian thực thi. Điều này đặc biệt hữu ích khi các bước không phụ thuộc vào nhau.

VI. Ví dụ thực tế (Use Cases)

Use Case 1: Phân tích và tóm tắt tài liệu pháp lý

- **Bước 1 (Trích xuất):** Prompt Claude trích xuất các điều khoản liên quan đến "bồi thường", "chấm dứt hợp đồng" từ một hợp đồng pháp lý. Đầu ra là danh sách các điều khoản, được định dạng bằng XML.
- **Bước 2 (Gate Check):** Một script Python kiểm tra xem tất cả các điều khoản yêu cầu đã được trích xuất đầy đủ. Nếu thiếu, hệ thống yêu cầu Claude thực hiện lại Bước 1 hoặc báo lỗi để người dùng can thiệp.
- **Bước 3 (Tóm tắt rủi ro):** Prompt Claude với vai trò "chuyên gia phân tích rủi ro pháp lý" để tóm tắt các rủi ro và nghĩa vụ chính dựa trên các điều khoản đã trích xuất từ Bước 1.

Use Case 2: Chu trình tạo nội dung marketing đa ngôn ngữ

- **Bước 1 (Nghiên cứu & Thu thập thông tin):** Prompt Claude thu thập thông tin chính về một sản phẩm hoặc dịch vụ. Đầu ra là các gạch đầu dòng thông tin quan trọng.
- **Bước 2 (Tạo dàn ý):** Prompt Claude với vai trò "chuyên gia lập dàn ý blog" để tạo dàn ý chi tiết cho một bài viết blog dựa trên thông tin đã thu thập từ Bước 1.
- **Bước 3 (Gate Check):** Một gate check tự động kiểm tra dàn ý có bao gồm các phần bắt buộc và tuân thủ các nguyên tắc nội dung đã định trước.
- **Bước 4 (Viết bản nháp):** Prompt Claude với vai trò "người viết nội dung blog" để viết bản nháp đầy đủ dựa trên dàn ý đã xác thực.
- **Bước 5 (Dịch thuật):** Prompt Claude với vai trò "dịch giả chuyên nghiệp" để dịch bản nháp tiếng Việt sang tiếng Tây Ban Nha, đảm bảo giữ nguyên ngữ nghĩa và phong cách [2].

VII. Lỗi thường gặp và cách tránh

- **Prompt quá phức tạp:** Cố gắng thực hiện quá nhiều việc trong một prompt duy nhất. **Cách tránh:** Tuân thủ nguyên tắc "mỗi prompt một nhiệm vụ" và chia nhỏ các bước thành các prompt riêng biệt, rõ ràng [1].
- **Thiếu Gate Checks:** Bỏ qua các bước xác thực trung gian dẫn đến lỗi dây chuyền, khiến các lỗi nhỏ ở đầu chuỗi lan truyền và gây ra vấn đề lớn ở cuối. **Cách tránh:** Luôn thêm các bước kiểm tra logic mạnh mẽ giữa các prompt để xác thực kết quả trung gian [3].
- **Phụ thuộc vào định dạng đầu ra không nhất quán:** LLM có thể tạo ra đầu ra với định dạng không đồng nhất, gây khó khăn cho việc xử lý ở các bước tiếp theo. **Cách tránh:** Yêu cầu Claude trả về kết quả trong các định dạng có cấu trúc như XML hoặc JSON, và sử dụng các công cụ phân tích cú pháp để đảm bảo tính nhất quán [1].

VIII. Kỹ thuật nâng cao: Chuỗi tự sửa lỗi (Self-Correction Chains)

Chuỗi tự sửa lỗi (self-correction chains) là kỹ thuật nâng cao cho phép Claude tự đánh giá và tinh chỉnh công việc của chính nó, nâng cao đáng kể chất lượng và độ tin cậy của đầu ra [1].

Quy trình cơ bản:

1. **Tạo bản nháp:** Claude tạo ra một đầu ra ban đầu dựa trên prompt.
2. **Đánh giá:** Một prompt khác (với vai trò "nhà phê bình" hoặc "người kiểm tra chất lượng") phân tích đầu ra ban đầu, tìm kiếm lỗi hoặc điểm cần cải thiện. Đầu ra của bước này là các phản hồi hoặc đề xuất sửa đổi cụ thể.
3. **Sửa đổi:** Prompt cuối cùng yêu cầu Claude viết lại hoặc tinh chỉnh đầu ra dựa trên những phản hồi từ bước đánh giá. Quá trình này có thể lặp lại nhiều lần cho đến khi đạt được chất lượng mong muốn.

Ví dụ điển hình là "chuỗi tóm tắt nghiên cứu tự sửa lỗi" [1]. Kỹ thuật này đặc biệt hữu ích trong các lĩnh vực đòi hỏi độ chính xác cao như nghiên cứu khoa học, phân tích tài

chính hoặc tạo mã nguồn.

IX. Kết luận

Chuỗi Prompt (Prompt Chaining) là một kỹ thuật nền tảng và mạnh mẽ để xây dựng các AI agents đáng tin cậy và có khả năng xử lý các nhiệm vụ phức tạp. Bằng cách chia nhỏ vấn đề, áp dụng các gate checks và tận dụng khả năng của Claude, chúng ta có thể khai thác tối đa tiềm năng của các mô hình ngôn ngữ lớn. Việc nắm vững và áp dụng hiệu quả prompt chaining sẽ là chìa khóa để phát triển các ứng dụng AI tiên tiến và bền vững, mang lại giá trị cao trong nhiều lĩnh vực khác nhau.

Tài liệu tham khảo

[1] Chain complex prompts for stronger performance. Claude API Docs. https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/chain-prompts [2] Breaking Down Tasks with Prompt Chaining | CodeSignal Learn. https://codesignal.com/learn/courses/exploring-workflows-with-claude/lessons/breaking-down-tasks-with-prompt-chaining-1 [3] Building Effective AI Agents. Anthropic. https://www.anthropic.com/research/building-effective-agents

> ***Khái niệm then chốt:*** *Prompt Chaining, Gate Checks, Sequential Processing, Subtask Decomposition, XML Tags, System Prompts, Self-Correction Chains*

Chương 3: Điều hướng thông minh (Routing)

*Báo cáo này đi sâu vào **Điều hướng thông minh (Intelligent Routing)** trong việc xây dựng AI agent, một kỹ thuật thiết yếu để tạo ra các hệ thống agent hiệu quả, chính xác và có khả năng mở rộng, đặc biệt trong bối cảnh hệ sinh thái Claude/Anthropic.*

1. Định nghĩa và Tầm quan trọng

Điều hướng thông minh (AI Agent Routing) là quá trình phân tích yêu cầu đầu vào và chọn lựa agent chuyên biệt phù hợp nhất để xử lý, nhằm tối ưu hóa hiệu suất và tài nguyên. **Router Agent** đóng vai trò bộ điều phối trung tâm, xác định ý định (intent) của người dùng và chuyển hướng tác vụ đến agent hoặc quy trình con có năng lực tốt nhất.

Lợi ích cốt lõi của điều hướng chính xác:

- **Tăng hiệu quả và độ chính xác:** Giao nhiệm vụ cho agent chuyên sâu giảm thiểu ảo giác và tăng chất lượng phản hồi.
- **Tối ưu hóa chi phí:** Sử dụng LLM phù hợp với độ phức tạp yêu cầu (ví dụ: Claude Haiku cho tác vụ đơn giản, Claude Opus cho phân tích sâu).
- **Dễ dàng mở rộng và bảo trì:** Kiến trúc module hóa giúp thêm, bớt hoặc cập nhật agent đơn giản mà không ảnh hưởng toàn hệ thống.

2. Kỹ thuật phân loại ý định (Intent Classification)

Khả năng phân loại chính xác ý định người dùng là cốt lõi của Router Agent. Các phương pháp đã tiến hóa:

- **Dựa trên quy tắc (Rule-based):** Đơn giản, dễ triển khai nhưng thiếu linh hoạt.
- **Dựa trên học máy truyền thống:** Linh hoạt hơn, nhưng đòi hỏi lượng lớn dữ liệu huấn luyện chất lượng cao.
- **Dựa trên Mô hình ngôn ngữ lớn (LLM-based):** Hiệu quả nhất với LLM như Claude, có khả năng hiểu ngữ cảnh, xử lý truy vấn phức tạp, mơ hồ và đa ý định mà không cần dữ liệu huấn luyện lớn. Các kỹ thuật như **Few-shot Learning**, **Chain-of-Thought Prompting**, **Structured Output** và **Guardrails** được sử dụng để tối ưu hóa khả năng phân loại.

3. Các mẫu định tuyến AI Agent

Các quy trình làm việc đa agent có thể triển khai nhiều mẫu định tuyến:

- **Định tuyến đơn Agent:** Yêu cầu đầu vào được chuyển hướng đến một agent duy nhất.
- **Định tuyến đa Agent:** Router chuyển truy vấn đến nhiều agent cùng lúc, sau đó tổng hợp phản hồi.
- **Định tuyến phân cấp:** Sử dụng nhiều lớp Router Agent để đưa ra quyết định định tuyến phức tạp hơn.

4. Các phương pháp hay nhất (Best Practices)

Để xây dựng hệ thống điều hướng agent mạnh mẽ:

1. **Xác định rõ ràng vai trò của Agent:** Mỗi agent cần có vai trò, trách nhiệm và phạm vi chuyên môn rõ ràng, tránh chồng chéo. Chỉ định rõ trong prompt định tuyến và minh họa bằng ví dụ.
2. **Bắt đầu đơn giản và mở rộng dần dần:** Xây dựng logic cơ bản trước, tăng độ phức tạp dần khi ứng dụng phát triển.
3. **Duy trì ngữ cảnh trong các tương tác:** Đảm bảo Router Agent và các agent chuyên biệt có quyền truy cập lịch sử hội thoại và ngữ cảnh liên quan.
4. **Triển khai xử lý lỗi và cơ chế dự phòng (Fallback Mechanisms):** Luôn có "Default Agent" hoặc chuyển hướng đến hỗ trợ con người khi không thể phân loại hoặc gặp lỗi.
5. **Đánh giá hiệu suất định tuyến trong sản xuất:** Thường xuyên giám sát và đánh giá hiệu suất để phát hiện và khắc phục "concept drift" (thay đổi mẫu truy vấn của người dùng).

5. Hướng dẫn từng bước: Xây dựng Router Agent cơ bản với Claude

Mục tiêu: Xây dựng Router Agent phân loại truy vấn hỗ trợ khách hàng thành `Hỗ trợ kỹ thuật`, `Yêu cầu hoàn tiền` và `Câu hỏi chung`.

Bước 1: Xác định các Agent chuyên biệt: Định nghĩa rõ các agent con (ví dụ: Technical Support Agent, Refund Request Agent, General Inquiry Agent).

Bước 2: Thiết kế Prompt cho Router Agent: Tạo prompt chi tiết cho Claude, bao gồm hướng dẫn, định dạng đầu ra mong muốn (JSON) và các ví dụ (few-shot examples) để Claude học cách phân loại.

Bước 3: Triển khai Logic Router và Tích hợp: Sử dụng Claude API để gửi truy vấn và prompt. Xử lý phản hồi từ Claude để lấy danh mục và chuyển yêu cầu đến agent hoặc hàm xử lý tương ứng.

6. Ví dụ thực tế (Use Cases)

1. **Hệ thống hỗ trợ khách hàng đa kênh:** Router Agent phân loại yêu cầu hỗ trợ (kỹ thuật, thanh toán, thông tin sản phẩm) và chuyển đến đội ngũ/agent AI chuyên biệt, giảm thời gian phản hồi.
2. **Nền tảng phát triển phần mềm (IDE/Coding Assistant):** Router Agent phân tích yêu cầu của nhà phát triển (ví dụ: “viết hàm Python”, “tìm lỗi”, “tạo tài liệu”) và điều hướng đến các agent tương ứng như “Coding Agent”, “Debugging Agent”.

7. Các cạm bẫy thường gặp và cách tránh

1. **Phân loại sai ý định (Misclassification):** Thiết kế prompt rõ ràng, cung cấp ví dụ đa dạng. Sử dụng “Judge LLM” hoặc phản hồi người dùng để cải thiện.
2. **Xử lý truy vấn mơ hồ hoặc đa ý định:** Sử dụng định tuyến đa agent/phân cấp. Thiết kế prompt để yêu cầu làm rõ. Áp dụng Chain-of-Thought để LLM suy luận sâu hơn.
3. **Concept Drift trong sản xuất:** Giám sát, đánh giá hiệu suất định tuyến thường xuyên. Thu thập dữ liệu mới và huấn luyện lại mô hình định kỳ.
4. **Thiếu cơ chế dự phòng (Fallback):** Luôn triển khai “Default Agent” hoặc chuyển hướng đến hỗ trợ con người.

8. Kỹ thuật nâng cao

1. **Định tuyến thích ứng (Adaptive Routing):** Điều chỉnh quyết định định tuyến dựa trên phản hồi của agent hoặc tương tác người dùng tiếp theo.
2. **Định tuyến dựa trên độ tin cậy (Confidence-based Routing):** Router đánh giá mức độ tin cậy của quyết định. Nếu độ tin cậy thấp, chuyển đến agent tổng quát hơn, con người, hoặc kích hoạt làm rõ.
3. **Tối ưu hóa mô hình định tuyến với RAG:** Sử dụng RAG để truy xuất kiến thức chuyên sâu trước khi định tuyến, tăng độ chính xác.

9. Kết luận

Điều hướng thông minh là trụ cột quan trọng trong việc xây dựng hệ thống AI agent mạnh mẽ và có khả năng mở rộng. Bằng cách hiểu rõ khái niệm, áp dụng phương pháp hay nhất và tận dụng kỹ thuật nâng cao, đặc biệt với sự hỗ trợ của LLM như Claude, chúng ta có thể tạo ra các agent xử lý tác vụ phức tạp hiệu quả và chính xác, mang lại giá trị to lớn cho nhiều ứng dụng.

Tài liệu tham khảo: [1] AI Agent Routing: Tutorial & Best Practices. Patronus AI. [2] Building Effective AI Agents. Anthropic.

> ***Khái niệm then chốt:*** *Router Agent, Phân loại ý định (Intent Classification), Định tuyến dựa trên LLM (LLM-based Routing), Định tuyến đơn Agent (Single-Agent Routing), Định tuyến đa Agent (Multi-Agent Routing), Định tuyến phân cấp (Hierarchical Routing), Định tuyến thích ứng (Adaptive Routing), Định tuyến dựa trên độ tin cậy (Confidence-based Routing)*

Chương 4: Song song hóa tác vụ (Parallelization)

1. Giới thiệu

Trong bối cảnh phát triển nhanh chóng của Trí tuệ Nhân tạo (AI), đặc biệt là các mô hình ngôn ngữ lớn (LLM) và hệ thống tác nhân AI (AI Agents), khả năng xử lý song song các tác vụ đã trở thành một yếu tố then chốt để nâng cao hiệu suất, hiệu quả và khả năng giải quyết vấn đề phức tạp. Thay vì thực hiện các bước tuần tự, việc phân chia và xử lý đồng thời nhiều phần của một vấn đề cho phép các tác nhân AI vượt qua giới hạn của một tác nhân đơn lẻ, mở rộng phạm vi ứng dụng và tăng tốc độ hoàn thành công việc. Chủ điểm này sẽ đi sâu vào các khái niệm, mô hình, chiến lược thực thi và tổng hợp kết quả trong bối cảnh song song hóa tác vụ cho AI Agents, đặc biệt là trong hệ sinh thái Claude/Anthropic.

2. Định nghĩa các khái niệm chính

Song song hóa tác vụ (Parallelization) là kỹ thuật cho phép nhiều tác vụ hoặc các phần của một tác vụ lớn được thực hiện đồng thời, thay vì tuần tự. Trong AI Agents, điều này có nghĩa là nhiều tác nhân hoặc nhiều phiên bản của cùng một tác nhân có thể hoạt động cùng lúc để giải quyết một vấn đề chung.

Thực thi tác nhân song song (Parallel Agent Execution) đề cập đến việc chạy nhiều tác nhân AI độc lập hoặc phụ thuộc lẫn nhau cùng một lúc. Mỗi tác nhân có thể được giao một phần cụ thể của vấn đề hoặc cùng nhau giải quyết một vấn đề chung từ các góc độ khác nhau. Điều này giúp tăng tốc độ xử lý và khả năng giải quyết các vấn đề phức tạp đòi hỏi nhiều tài nguyên hoặc chuyên môn khác nhau [1].

Mô hình Sectioning (Phân đoạn) là một chiến lược song song hóa trong đó một tác vụ lớn được chia thành nhiều phần nhỏ hơn, độc lập. Mỗi phần được giao cho một tác nhân riêng biệt để xử lý. Các tác nhân này hoạt động trên các phần dữ liệu hoặc các khía cạnh khác nhau của vấn đề mà không cần tương tác trực tiếp với nhau trong quá trình thực thi. Kết quả từ các tác nhân sau đó được tổng hợp lại để tạo ra giải pháp cuối cùng. Ví dụ, trong dự án xây dựng trình biên dịch C của Anthropic, các tác nhân Claude

được giao các nhiệm vụ khác nhau như phân tích cú pháp, tạo mã, tối ưu hóa, và mỗi tác nhân làm việc trên các phần mã nguồn khác nhau [1].

Mô hình Voting (Bỏ phiếu) là một chiến lược tổng hợp kết quả nơi nhiều tác nhân cùng giải quyết một vấn đề hoặc đưa ra một quyết định, sau đó bỏ phiếu cho kết quả mà họ cho là tốt nhất. Một tác nhân điều phối (coordinator agent) sẽ thu thập các đề xuất hoặc phán đoán từ mỗi tác nhân và tổng hợp phiếu bầu để xác định kết quả cuối cùng. Việc bỏ phiếu có thể là đa số đơn giản hoặc có trọng số (ví dụ: phiếu bầu của tác nhân chuyên gia có trọng số cao hơn). Mô hình này giúp tăng cường độ chính xác và độ tin cậy thông qua sự đồng thuận, giảm thiểu sai sót hoặc thiên vị từ một tác nhân đơn lẻ [4].

Chiến lược tổng hợp (Aggregation Strategies) là các phương pháp được sử dụng để kết hợp hoặc tổng hợp kết quả từ nhiều tác nhân song song thành một đầu ra cuối cùng, có ý nghĩa. Các chiến lược này có thể bao gồm: bỏ phiếu (voting), hợp nhất (merging), tổng hợp có trọng số (weighted aggregation), hoặc sử dụng một tác nhân tổng hợp (aggregator agent) để phân tích và tổng hợp các kết quả riêng lẻ [3].

3. Lợi ích của Song song hóa tác vụ

Việc áp dụng song song hóa tác vụ mang lại nhiều lợi ích đáng kể cho hệ thống AI Agents:

- **Tăng tốc độ xử lý**: Bằng cách thực hiện nhiều tác vụ cùng lúc, thời gian hoàn thành tổng thể được rút ngắn đáng kể [3].
- **Nâng cao hiệu suất**: Các hệ thống đa tác nhân có thể giải quyết các vấn đề phức tạp hơn mà một tác nhân đơn lẻ không thể xử lý hiệu quả, đặc biệt là các truy vấn yêu cầu tìm kiếm thông tin rộng rãi [2].
- **Chuyên môn hóa**: Cho phép các tác nhân chuyên biệt tập trung vào các khía cạnh cụ thể của vấn đề (ví dụ: một tác nhân duy trì tài liệu, một tác nhân kiểm tra chất lượng mã, một tác nhân tối ưu hóa hiệu suất) [1].
- **Độ tin cậy và khả năng chịu lỗi**: Trong mô hình bỏ phiếu, nếu một hoặc hai tác nhân thất bại hoặc đưa ra kết quả sai lệch, hệ thống vẫn có thể hoạt động dựa trên sự đồng thuận của các tác nhân khác [4].

- **Giảm thiểu thiên vị**: Mô hình bỏ phiếu giúp giảm thiểu ảnh hưởng của thiên vị từ một tác nhân đơn lẻ bằng cách lấy ý kiến từ nhiều nguồn khác nhau [4].

4. Best Practices thực tế

1. **Thiết kế kiểm thử chất lượng cao**: Các tác nhân AI sẽ làm việc tự động để giải quyết vấn đề được giao. Do đó, điều quan trọng là hệ thống kiểm thử phải gần như hoàn hảo để đảm bảo tác nhân không giải quyết sai vấn đề. Cần xây dựng các bộ kiểm thử toàn diện và cơ chế kiểm tra liên tục (CI) để phát hiện lỗi sớm và đảm bảo các thay đổi mới không phá vỡ chức năng hiện có [1].

2. **Quản lý ngữ cảnh hiệu quả**: Các mô hình ngôn ngữ lớn có giới hạn về cửa sổ ngữ cảnh. Hệ thống cần được thiết kế để tránh "ô nhiễm cửa sổ ngữ cảnh" (context window pollution) bằng cách chỉ in ra một vài dòng đầu ra cần thiết và ghi tất cả thông tin quan trọng vào tệp nhật ký để tác nhân có thể tìm thấy khi cần. Các tệp nhật ký nên dễ dàng xử lý tự động và bao gồm các thống kê tóm tắt tổng hợp để tác nhân không phải tính toán lại [1].

3. **Làm cho việc song song hóa trở nên dễ dàng**: Khi có nhiều lỗi khác nhau, việc song song hóa là đơn giản: mỗi tác nhân chọn một lỗi khác nhau để xử lý. Tuy nhiên, khi các tác vụ trở nên phụ thuộc hơn, cần có cơ chế đồng bộ hóa để ngăn các tác nhân làm việc chồng chéo hoặc ghi đè lên nhau. Ví dụ, sử dụng cơ chế "khóa" tác vụ để mỗi tác nhân chỉ làm việc trên một phần cụ thể tại một thời điểm [1].

4. **Kỹ thuật Prompt để ủy quyền hiệu quả**: Trong hệ thống đa tác nhân, tác nhân điều phối cần được hướng dẫn cách phân chia truy vấn thành các tác vụ con và mô tả rõ ràng cho các tác nhân con. Mỗi tác nhân con cần có mục tiêu, định dạng đầu ra, hướng dẫn về công cụ và nguồn sử dụng, và ranh giới tác vụ rõ ràng. Điều này ngăn chặn việc trùng lặp công việc hoặc bỏ sót thông tin [2].

5. **Điều chỉnh nỗ lực theo độ phức tạp của truy vấn**: Các tác nhân thường gặp khó khăn trong việc đánh giá mức độ nỗ lực phù hợp cho các tác vụ khác nhau. Cần nhúng các quy tắc điều chỉnh quy mô vào prompt. Ví dụ, tìm kiếm thông tin đơn giản chỉ cần 1 tác nhân với 3-10 lượt gọi công cụ, so sánh trực tiếp có thể cần 2-4 tác nhân con với 10-15 lượt gọi, và nghiên cứu phức tạp có thể sử dụng hơn 10 tác nhân con với trách nhiệm được phân chia rõ ràng [2].

5. Step-by-step Guide: Triển khai thực thi tác nhân song song và tổng hợp kết quả

Để triển khai một hệ thống tác nhân song song, có thể theo các bước sau, dựa trên ví dụ từ CodeSignal và Anthropic [1] [3]:

1. **Phân tích và phân chia tác vụ**: Xác định tác vụ chính và phân chia nó thành các tác vụ con độc lập hoặc bán độc lập. Ví dụ, nếu là lập kế hoạch du lịch, các tác vụ con có thể là: tìm điểm tham quan, phương tiện di chuyển, văn hóa địa phương [3].

2. **Khởi tạo tác nhân**: Tạo nhiều phiên bản của tác nhân AI (hoặc các tác nhân chuyên biệt) cho mỗi tác vụ con. Trong hệ sinh thái Claude, điều này có thể liên quan đến việc gọi API Claude nhiều lần với các prompt khác nhau [3].

3. **Thực thi song song**: Sử dụng các cơ chế lập trình bất đồng bộ (ví dụ: `Promise.all` trong JavaScript/TypeScript) để gửi các yêu cầu đến các tác nhân Claude đồng thời. Điều này cho phép tất cả các tác nhân con bắt đầu xử lý tác vụ của họ mà không cần chờ đợi lẫn nhau [3].

4. **Đồng bộ hóa và quản lý ngữ cảnh**: Nếu các tác nhân cần chia sẻ tài nguyên hoặc trạng thái, hãy triển khai cơ chế đồng bộ hóa (ví dụ: hệ thống khóa tác vụ dựa trên Git như Anthropic đã làm) để tránh xung đột và đảm bảo tính nhất quán của dữ liệu [1].

5. **Thu thập kết quả**: Sau khi tất cả các tác nhân con hoàn thành tác vụ của mình, thu thập tất cả các kết quả riêng lẻ. Điều này có thể được thực hiện bằng cách chờ tất cả các promise hoàn thành trong lập trình bất đồng bộ [3].

6. **Tổng hợp kết quả**: Sử dụng một tác nhân tổng hợp (aggregator agent) hoặc một chiến lược tổng hợp đã định nghĩa (ví dụ: bỏ phiếu, hợp nhất thông tin) để kết hợp các kết quả riêng lẻ thành một đầu ra cuối cùng, mạch lạc và có ý nghĩa. Tác nhân tổng hợp có thể được prompt để phân tích, tóm tắt và trình bày thông tin một cách có tổ chức [3].

7. **Kiểm tra và lặp lại**: Đánh giá chất lượng của kết quả tổng hợp. Nếu cần, điều chỉnh các prompt, chiến lược phân chia tác vụ hoặc chiến lược tổng hợp và lặp lại quy trình.

6. Ví dụ thực tế (Use Cases)

6.1. Xây dựng trình biên dịch C với đội ngũ Claude song song [1]

Anthropic đã thử nghiệm việc xây dựng một trình biên dịch C bằng cách sử dụng 16 tác nhân Claude hoạt động song song. Mỗi tác nhân được đặt trong một container Docker riêng biệt và làm việc trên một kho lưu trữ Git chung. Để quản lý sự đồng bộ, một cơ chế "khóa" tác vụ đơn giản đã được triển khai: một tác nhân sẽ tạo một tệp văn bản để "khóa" một tác vụ (ví dụ: `parse_if_statement.txt`). Nếu hai tác nhân cố gắng khóa cùng một tác vụ, Git sẽ xử lý xung đột và tác nhân thứ hai sẽ chọn một tác vụ khác. Sau khi hoàn thành, tác nhân sẽ đẩy thay đổi lên kho lưu trữ chung và xóa khóa. Các xung đột hợp nhất (merge conflicts) thường xuyên xảy ra, nhưng Claude đủ thông minh để tự giải quyết. Dự án này đã tạo ra một trình biên dịch 100.000 dòng mã có khả năng biên dịch Linux kernel, chứng minh hiệu quả của việc thực thi tác nhân song song cho các tác vụ kỹ thuật phức tạp.

6.2. Hệ thống nghiên cứu đa tác nhân của Anthropic [2]

Hệ thống nghiên cứu của Anthropic sử dụng kiến trúc đa tác nhân với mô hình điều phối viên-người thực hiện (orchestrator-worker). Khi người dùng gửi một truy vấn, tác nhân chính (lead agent) sẽ phân tích, phát triển chiến lược và tạo ra các tác nhân con chuyên biệt để khám phá các khía cạnh khác nhau của truy vấn một cách đồng thời. Các tác nhân con này hoạt động như các bộ lọc thông minh, sử dụng công cụ tìm kiếm để thu thập thông tin và trả về kết quả cho tác nhân chính. Tác nhân chính sau đó tổng hợp các kết quả này và quyết định xem có cần nghiên cứu thêm hay không. Hệ thống này đã cho thấy hiệu suất vượt trội 90.2% so với hệ thống tác nhân đơn lẻ trong các tác vụ tìm kiếm thông tin rộng rãi, ví dụ như xác định tất cả thành viên hội đồng quản trị của các công ty trong S&P 500.

6.3. Đánh giá rủi ro tài chính bằng mô hình bỏ phiếu [4]

Một ngân hàng muốn hệ thống AI đánh giá các đơn xin vay vốn. Thay vì dựa vào một mô hình AI duy nhất, họ sử dụng một hội đồng các tác nhân AI, mỗi tác nhân phân tích rủi ro của người nộp đơn từ các góc độ khác nhau (lịch sử tín dụng, sự ổn định thu nhập, điều kiện thị trường, v.v.). Các tác nhân này sau đó bỏ phiếu về việc có nên chấp thuận hay từ chối khoản vay. Điều này đảm bảo rằng quyết định không bị ảnh hưởng bởi một mô hình thiên vị hoặc dễ mắc lỗi duy nhất. Nếu có 5 tác nhân, mỗi tác nhân có

độ chính xác 80%, thì quyết định dựa trên đa số phiếu có thể đạt độ chính xác cao hơn đáng kể.

7. Các lỗi thường gặp và cách tránh

1. **Ô nhiễm cửa sổ ngữ cảnh (Context Window Pollution)**: Các tác nhân có thể bị quá tải bởi lượng thông tin không cần thiết trong cửa sổ ngữ cảnh, dẫn đến giảm hiệu suất. **Cách tránh**: Thiết kế hệ thống để chỉ cung cấp thông tin cần thiết nhất cho tác nhân, ghi nhật ký chi tiết vào tệp và cung cấp các bản tóm tắt tổng hợp thay vì toàn bộ dữ liệu thô [1].

2. **Mù thời gian (Time Blindness)**: Các mô hình ngôn ngữ không thể nhận biết thời gian và có thể dành quá nhiều thời gian cho các tác vụ không quan trọng. **Cách tránh**: Xây dựng cơ chế để in tiến độ tăng dần (không quá thường xuyên để tránh ô nhiễm ngữ cảnh) và cung cấp các tùy chọn chạy nhanh (ví dụ: chỉ chạy một mẫu nhỏ các bài kiểm tra) để tác nhân có thể nhanh chóng xác định các hồi quy [1].

3. **Phân chia tác vụ không hiệu quả**: Nếu các tác vụ không được phân chia rõ ràng hoặc có sự phụ thuộc cao, các tác nhân có thể trùng lặp công việc, bỏ sót thông tin hoặc gây ra xung đột. **Cách tránh**: Sử dụng kỹ thuật prompt engineering để hướng dẫn tác nhân điều phối phân chia tác vụ một cách rõ ràng, xác định mục tiêu, định dạng đầu ra và ranh giới tác vụ cho mỗi tác nhân con [2].

4. **Chi phí token cao**: Hệ thống đa tác nhân thường tiêu tốn nhiều token hơn đáng kể so với tương tác trò chuyện đơn lẻ. **Cách tránh**: Đảm bảo rằng giá trị của tác vụ đủ cao để bù đắp chi phí token tăng lên. Tối ưu hóa prompt và thiết kế hệ thống để giảm thiểu các lượt gọi API không cần thiết [2].

5. **Độ trễ giao tiếp và tổng hợp**: Việc thu thập phản hồi từ nhiều tác nhân và tổng hợp chúng có thể gây ra độ trễ đáng kể. **Cách tránh**: Giới hạn số lượng tác nhân bỏ phiếu hoặc chạy chúng song song để quản lý chi phí này. Đảm bảo cơ chế tổng hợp hiệu quả và nhanh chóng [4].

6. **Không đảm bảo sự đồng thuận trong mô hình bỏ phiếu**: Các tác nhân có thể chia phiếu bầu, dẫn đến không có quyết định rõ ràng. **Cách tránh**: Sử dụng số lượng tác nhân lẻ để giảm thiểu trường hợp hòa. Thiết lập các quy tắc xử lý trường hợp không đạt được đa số rõ ràng, ví dụ: yêu cầu sự đồng thuận cao hơn hoặc chuyển giao cho con người [4].

8. Kỹ thuật nâng cao

- **Kiến trúc Orchestrator-Worker**: Một tác nhân chính (orchestrator) chịu trách nhiệm lập kế hoạch, phân chia tác vụ và điều phối các tác nhân con (worker) chuyên biệt. Các tác nhân con thực hiện các tác vụ cụ thể và trả về kết quả cho tác nhân chính để tổng hợp [2].
- **Tác nhân chuyên biệt (Specialized Agents)**: Thay vì chỉ có các tác nhân giống hệt nhau, tạo ra các tác nhân có chuyên môn hóa cao cho các nhiệm vụ cụ thể (ví dụ: tác nhân kiểm tra mã, tác nhân viết tài liệu, tác nhân tối ưu hóa). Điều này tận dụng tối đa khả năng của từng tác nhân [1].
- **Tìm kiếm động và thích ứng**: Thay vì sử dụng Retrieval Augmented Generation (RAG) tĩnh, hệ thống có thể sử dụng tìm kiếm đa bước, động để tìm thông tin liên quan, thích ứng với các phát hiện mới và phân tích kết quả để đưa ra câu trả lời chất lượng cao [2].
- **Tự cải thiện tác nhân (Agent Self-Improvement)**: Cho phép các tác nhân tự chẩn đoán lỗi và đề xuất cải tiến cho prompt hoặc công cụ của chúng. Anthropic đã sử dụng một tác nhân kiểm thử công cụ để tự động viết lại mô tả công cụ, giảm 40% thời gian hoàn thành tác vụ trong tương lai [2].

9. Kết luận

Song song hóa tác vụ là một kỹ thuật mạnh mẽ để mở rộng khả năng của AI Agents, đặc biệt là trong các hệ thống phức tạp như Claude/Anthropic. Bằng cách phân chia tác vụ, thực thi song song và áp dụng các chiến lược tổng hợp thông minh như sectioning và voting, các hệ thống AI có thể đạt được hiệu suất, độ chính xác và độ tin cậy cao hơn. Tuy nhiên, việc triển khai đòi hỏi sự chú ý đến các thách thức như quản lý ngữ cảnh, chi phí token và đồng bộ hóa. Với các best practices và kỹ thuật nâng cao, chúng ta có thể xây dựng các hệ thống tác nhân AI mạnh mẽ và hiệu quả hơn trong tương lai.

10. Tài liệu tham khảo

1. Building a C compiler with a team of parallel Claudes - Anthropic Engineering Blog

2. How we built our multi-agent research system - Anthropic Engineering Blog
3. Parallel Processing with Claude | CodeSignal Learn - CodeSignal Learn
4. Patterns for Democratic Multi-Agent AI: Voting-Based Council — Part 1 - Medium.com

> ***Khái niệm then chốt:*** *Song song hóa tác vụ, Thực thi tác nhân song song, Mô hình Sectioning, Mô hình Voting, Chiến lược tổng hợp, Ô nhiễm cửa sổ ngữ cảnh, Mù thời gian, Kiến trúc Orchestrator-Worker, Tác nhân chuyên biệt, Tự cải thiện tác nhân*

Chương 5: Kiến trúc Điều phối - Thực thi (Orchestrator-Workers)

Kiến trúc Điều phối - Thực thi (Orchestrator-Workers) trong AI Agents

Kiến trúc Điều phối - Thực thi (Orchestrator-Workers) là một mô hình mạnh mẽ trong thiết kế hệ thống AI Agent, đặc biệt phù hợp cho việc giải quyết các tác vụ phức tạp, không thể dự đoán trước, nơi mà cấu trúc vấn đề chỉ phát sinh trong quá trình thực thi. Mô hình này được Anthropic và nhiều tổ chức khác áp dụng để xây dựng các hệ thống tác nhân hiệu quả, có khả năng mở rộng và thích ứng cao.

1. Định nghĩa và Diễn giải Thuật ngữ Kỹ thuật

Trong kiến trúc Orchestrator-Workers, một **Orchestrator Agent (Tác nhân Điều phối)** trung tâm sẽ quản lý và điều phối hoạt động của nhiều **Worker Agents (Tác nhân Thực thi)** chuyên biệt. Quá trình này bao gồm phân tách tác vụ, ủy quyền, thực thi và tổng hợp kết quả.

- **Orchestrator Agent (Tác nhân Điều phối):** Thường là một Mô hình Ngôn ngữ Lớn (LLM) mạnh mẽ, đóng vai trò là bộ não của hệ thống. Nhiệm vụ chính của nó là:

    - **Phân tách tác vụ (Task Decomposition):** Nhận một tác vụ lớn, phức tạp từ người dùng và chia nhỏ nó thành một tập hợp các tác vụ con (subtasks) nhỏ hơn, dễ quản lý hơn. Điều này khác biệt so với các luồng công việc cố định vì Orchestrator tự động xác định các tác vụ con này dựa trên đầu vào cụ thể.
    - **Ủy quyền tác vụ (Task Delegation):** Gán các tác vụ con đã phân tách cho các Worker Agents phù hợp nhất dựa trên chuyên môn hoặc khả năng của chúng. Orchestrator cần có khả năng "hiểu" được năng lực của từng Worker.
    - **Tổng hợp kết quả (Result Synthesis):** Thu thập các kết quả từ các Worker Agents, đánh giá, tổng hợp và định dạng chúng thành một phản hồi mạch lạc, hoàn chỉnh cho tác vụ ban đầu.
    - **Lập kế hoạch động (Dynamic Planning):** Liên tục theo dõi tiến độ, đánh giá kết quả trung gian từ Worker Agents và điều chỉnh kế hoạch, tạo thêm tác vụ con hoặc thay đổi hướng tiếp cận nếu cần, cho đến khi mục tiêu cuối cùng được đạt được.

- **Worker Agents (Tác nhân Thực thi):** Là các LLM hoặc công cụ chuyên biệt, được thiết kế để thực hiện các tác vụ con cụ thể. Mỗi Worker Agent có thể được tối ưu hóa cho một loại công việc nhất định (ví dụ: tìm kiếm thông tin, viết mã, phân tích dữ liệu). Chúng nhận tác vụ từ Orchestrator, thực hiện công việc và trả về kết quả cho Orchestrator.

- **Phân tách tác vụ (Task Decomposition):** Quá trình chia một vấn đề lớn thành các phần nhỏ hơn, độc lập hoặc phụ thuộc lẫn nhau. Trong kiến trúc này, Orchestrator thực hiện việc này một cách linh hoạt, không theo một kịch bản cố định.

- **Tổng hợp kết quả (Result Synthesis):** Quá trình thu thập, xử lý và kết hợp các đầu ra riêng lẻ từ các Worker Agents thành một kết quả cuối cùng có ý nghĩa và đáp ứng yêu cầu của tác vụ ban đầu.

2. Best Practices Thực tế

1. **Dạy Orchestrator cách ủy quyền hiệu quả:** Orchestrator cần được hướng dẫn rõ ràng về cách phân tách tác vụ và mô tả chúng cho Worker Agents. Mỗi tác vụ con cần có mục tiêu rõ ràng, định dạng đầu ra mong muốn, hướng dẫn về công cụ và nguồn lực cần sử dụng, và ranh giới tác vụ cụ thể. Điều này giúp tránh trùng lặp công việc, bỏ sót thông tin hoặc hiểu sai tác vụ.

2. **Điều chỉnh nỗ lực theo độ phức tạp của truy vấn:** Các tác nhân thường gặp khó khăn trong việc đánh giá mức độ nỗ lực phù hợp cho các tác vụ khác nhau. Cần nhúng các quy tắc điều chỉnh quy mô vào các prompt của Orchestrator. Ví dụ, một tác vụ tìm kiếm thông tin đơn giản có thể chỉ cần 1 tác nhân với 3-10 lượt gọi công cụ, trong khi nghiên cứu phức tạp có thể cần hơn 10 tác nhân con với trách nhiệm được phân chia rõ ràng.

3. **Thiết kế và lựa chọn công cụ một cách cẩn thận:** Giao diện tác nhân-công cụ là rất quan trọng. Mỗi công cụ cần có mục đích riêng biệt và mô tả rõ ràng. Orchestrator nên được hướng dẫn để kiểm tra tất cả các công cụ có sẵn trước, khớp việc sử dụng công cụ với ý định của người dùng, ưu tiên tìm kiếm trên web cho các khám phá rộng rãi hoặc ưu tiên các công cụ chuyên biệt hơn các công cụ chung chung.

4. **Cho phép các tác nhân tự cải thiện:** Các mô hình LLM tiên tiến (như Claude 4) có thể là những kỹ sư prompt xuất sắc. Khi được cung cấp một prompt và một chế độ lỗi, chúng có khả năng chẩn đoán lý do tác nhân thất bại và đề xuất cải tiến. Điều này có thể được thực hiện thông qua một tác nhân kiểm tra công cụ, tự động thử nghiệm và viết lại mô tả công cụ để tránh lỗi.

5. **Bắt đầu rộng, sau đó thu hẹp:** Chiến lược tìm kiếm nên mô phỏng cách nghiên cứu của con người: khám phá tổng quan trước khi đi sâu vào chi tiết. Prompt các tác nhân bắt đầu với các truy vấn ngắn, rộng, đánh giá những gì có sẵn, sau đó dần dần thu hẹp trọng tâm.

3. Step-by-Step Guide: Xây dựng hệ thống Orchestrator-Workers

Đây là một hướng dẫn từng bước dựa trên cách Anthropic xây dựng hệ thống nghiên cứu đa tác nhân của họ:

1. **Phân tích và Lập kế hoạch (Analysis & Planning Phase):**
    - **Orchestrator nhận tác vụ:** Orchestrator LLM nhận tác vụ và ngữ cảnh từ người dùng.
    - **Phân tích và xác định cách tiếp cận:** Orchestrator phân tích tác vụ, xác định các cách tiếp cận có giá trị và tạo ra các mô tả tác vụ con có cấu trúc (thường ở định dạng XML để dễ phân tích).
    - **Lưu kế hoạch:** Orchestrator lưu kế hoạch vào bộ nhớ để duy trì ngữ cảnh, đặc biệt nếu cửa sổ ngữ cảnh vượt quá giới hạn.

2. **Thực thi (Execution Phase):**

    - **Tạo và ủy quyền Worker Agents:** Orchestrator tạo ra các Worker Agents chuyên biệt (ví dụ: Subagents) và ủy quyền các tác vụ con cho chúng. Các Worker Agents này có thể hoạt động song song.
    - **Mỗi Worker Agent thực hiện tác vụ:** Mỗi Worker Agent nhận tác vụ ban đầu để có ngữ cảnh, loại tác vụ con cụ thể, mô tả và bất kỳ ngữ cảnh bổ sung nào. Chúng độc lập thực hiện công việc được giao (ví dụ: tìm kiếm web, sử dụng công cụ, phân tích kết quả).
    - **Đánh giá và tinh chỉnh (Interleaved Thinking):** Các Worker Agents sử dụng "suy nghĩ xen kẽ" (interleaved thinking) sau khi nhận kết quả công cụ để đánh giá chất lượng, xác định các khoảng trống và tinh chỉnh truy vấn tiếp theo của chúng.
    - **Trả về kết quả:** Mỗi Worker Agent trả về kết quả của tác vụ con cho Orchestrator.

3. **Tổng hợp và Quyết định (Synthesis & Decision Phase):**

    - **Orchestrator tổng hợp kết quả:** Orchestrator thu thập và tổng hợp các kết quả từ tất cả các Worker Agents.
    - **Đánh giá và quyết định tiếp theo:** Orchestrator đánh giá các kết quả đã tổng hợp và quyết định xem có cần nghiên cứu thêm hay không. Nếu cần, nó có thể tạo thêm Worker Agents hoặc tinh chỉnh chiến lược của mình.
    - **Xử lý trích dẫn (nếu có):** Khi đủ thông tin, hệ thống có thể chuyển tất cả các phát hiện cho một CitationAgent để xử lý tài liệu và báo cáo nghiên cứu, xác định các vị trí cụ thể cho trích dẫn.
    - **Trả về kết quả cuối cùng:** Kết quả nghiên cứu cuối cùng, hoàn chỉnh với các trích dẫn (nếu có), được trả về cho người dùng.

4. Ví dụ Thực tế (Use Cases)

1. **Hệ thống Nghiên cứu Đa tác nhân của Anthropic:**

    - **Mô tả:** Hệ thống này sử dụng một Lead Agent (Orchestrator) để phân tích truy vấn nghiên cứu của người dùng, sau đó tạo ra các Subagents (Workers) chuyên biệt để tìm kiếm thông tin trên web, Google Workspace và các tích

hợp khác một cách song song. Các Subagents sau đó trả về thông tin đã lọc và tổng hợp cho Lead Agent để biên soạn câu trả lời cuối cùng.

- **Lợi ích:** Vượt trội hơn 90% so với hệ thống đơn tác nhân trong các tác vụ nghiên cứu mở rộng, đặc biệt là các truy vấn yêu cầu nhiều hướng độc lập. Ví dụ, khi được yêu cầu xác định tất cả các thành viên hội đồng quản trị của các công ty trong S&P 500 ngành Công nghệ thông tin, hệ thống đa tác nhân đã tìm thấy câu trả lời chính xác bằng cách phân tách thành các tác vụ cho các tác nhân con, trong khi hệ thống đơn tác nhân thất bại.

2. **Tạo nội dung Marketing đa dạng:**

- **Mô tả:** Một Orchestrator Agent nhận yêu cầu mô tả sản phẩm. Nó phân tích các loại nội dung marketing có giá trị (ví dụ: bản sao quảng cáo, bài đăng trên mạng xã hội, email marketing). Sau đó, nó tạo ra các mô tả tác vụ chuyên biệt cho các Worker Agents. Mỗi Worker Agent sẽ tạo ra các biến thể nội dung được tối ưu hóa cho các đối tượng khác nhau (ví dụ: đối tượng trẻ, đối tượng doanh nghiệp). Cuối cùng, Orchestrator tổng hợp và trả về các kết quả phối hợp từ tất cả các Worker.
- **Lợi ích:** Tạo ra nhiều biến thể nội dung chất lượng cao, phù hợp với nhiều đối tượng mục tiêu một cách nhanh chóng và hiệu quả, mà không cần phải dự đoán trước tất cả các biến thể cần thiết.

5. Các Lỗi Thường Gặp và Cách Tránh

1. **Phân tách tác vụ kém hiệu quả:**

- **Lỗi:** Orchestrator phân tách tác vụ quá thô hoặc quá mịn, dẫn đến các Worker Agents không thể thực hiện hiệu quả hoặc trùng lặp công việc.
- **Cách tránh:** Cung cấp cho Orchestrator các hướng dẫn rõ ràng và ví dụ về cách phân tách tác vụ. Sử dụng định dạng có cấu trúc (như XML) cho các mô tả tác vụ con để đảm bảo tính nhất quán. Thực hiện “suy nghĩ xen kẽ” (interleaved thinking) trong Orchestrator để nó có thể tự đánh giá và điều chỉnh kế hoạch phân tách.

2. **Quản lý ngữ cảnh kém:**

- **Lỗi:** Worker Agents thiếu ngữ cảnh cần thiết để hoàn thành tác vụ hoặc Orchestrator bị quá tải bởi ngữ cảnh không liên quan.

- **Cách tránh:** Đảm bảo mỗi Worker Agent nhận được cả tác vụ gốc và mô tả tác vụ con cụ thể. Sử dụng các kỹ thuật quản lý bộ nhớ (ví dụ: bộ nhớ ngắn hạn cho dữ liệu tạm thời, bộ nhớ dài hạn cho các quyết định quan trọng, kế hoạch tổng thể) và chỉ truyền những thông tin cần thiết giữa các tác nhân để tránh "phình to" ngữ cảnh.

3. **Xử lý lỗi không hiệu quả:**
    - **Lỗi:** Hệ thống không thể phục hồi gracefully khi một Worker Agent thất bại hoặc đi vào vòng lặp vô hạn.
    - **Cách tránh:** Triển khai các cơ chế xử lý lỗi mạnh mẽ, bao gồm xác thực đầu ra của Worker Agents (ví dụ: kiểm tra phản hồi trống). Thiết lập các "guardrails" và giới hạn số lần thử lại (retries) ở cấp độ tác vụ hoặc nút. Sử dụng các công cụ giám sát và ghi nhật ký để theo dõi luồng thực thi và phát hiện sớm các vấn đề.
4. **Chi phí token cao:**
    - **Lỗi:** Kiến trúc đa tác nhân có thể tiêu thụ lượng token lớn, dẫn đến chi phí cao, đặc biệt với các tác vụ đơn giản.
    - **Cách tránh:** Điều chỉnh nỗ lực của tác nhân theo độ phức tạp của truy vấn. Sử dụng các mô hình LLM nhỏ hơn, hiệu quả hơn (ví dụ: Claude Haiku) cho các tác vụ con đơn giản và chỉ sử dụng các mô hình mạnh mẽ hơn (ví dụ: Claude Opus) cho các tác vụ phức tạp hoặc Orchestrator. Tối ưu hóa prompt để giảm thiểu độ dài không cần thiết.

6. Kỹ thuật Nâng cao

1. **Sử dụng XML cho cấu trúc đầu ra:** Anthropic khuyến nghị sử dụng XML để tạo ra các mô tả tác vụ con có cấu trúc từ Orchestrator. Điều này giúp việc phân tích cú pháp trở nên đáng tin cậy và thân thiện với mô hình ngôn ngữ, đảm bảo các Worker Agents nhận được hướng dẫn rõ ràng.

2. **Parallel Tool Calling:** Để tăng tốc độ xử lý, đặc biệt trong các tác vụ nghiên cứu phức tạp, Orchestrator có thể tạo ra nhiều Subagents song song và mỗi Subagent có thể sử dụng nhiều công cụ cùng lúc. Điều này có thể giảm thời gian nghiên cứu tới 90% cho các truy vấn phức tạp.

3. **Interleaved Thinking (Suy nghĩ xen kẽ):** Kỹ thuật này cho phép các tác nhân (cả Orchestrator và Workers) xuất ra một quá trình suy nghĩ có thể kiểm soát được. Điều này giúp tác nhân lập kế hoạch, đánh giá kết quả công cụ, xác định khoảng trống và tinh chỉnh truy vấn tiếp theo. Nó cải thiện khả năng tuân thủ hướng dẫn, lý luận và hiệu quả.

4. **Tối ưu hóa Prompt để tự cải thiện:** Thiết kế prompt sao cho các tác nhân có thể tự chẩn đoán lỗi và đề xuất cải tiến cho prompt hoặc mô tả công cụ. Điều này có thể được thực hiện bằng cách tạo một tác nhân kiểm tra công cụ chuyên biệt.

5. **Quản lý Bộ nhớ Phân cấp:** Kết hợp bộ nhớ ngắn hạn (cho ngữ cảnh hiện tại của tác vụ con) và bộ nhớ dài hạn (cho các quyết định quan trọng, kế hoạch tổng thể) để duy trì tính liên tục và tránh mất ngữ cảnh trong các luồng công việc dài. Các framework như CrewAI cung cấp các loại bộ nhớ có thể cấu hình để kiểm soát chi tiết hơn.

Kiến trúc Orchestrator-Workers mang lại sự linh hoạt và khả năng mở rộng đáng kể cho các hệ thống AI Agent, cho phép chúng giải quyết các vấn đề phức tạp một cách hiệu quả hơn bằng cách tận dụng sức mạnh của nhiều tác nhân chuyên biệt được điều phối thông minh. Tuy nhiên, việc triển khai thành công đòi hỏi sự chú ý đến các best practices về thiết kế prompt, quản lý ngữ cảnh và xử lý lỗi.

> ***Khái niệm then chốt:*** *Orchestrator Agent, Worker Agents, Task Decomposition, Result Synthesis, Dynamic Planning, Interleaved Thinking, Parallel Tool Calling, Context Management*

Chương 6: Vòng lặp Đánh giá - Tối ưu (Evaluator-Optimizer)

Vòng lặp Đánh giá - Tối ưu (Evaluator-Optimizer) trong AI Agents

1. Định nghĩa

Vòng lặp Đánh giá - Tối ưu (Evaluator-Optimizer) là một mô hình kiến trúc quan trọng trong việc xây dựng các tác nhân AI (AI agents) hiệu quả, đặc biệt trong hệ sinh thái của Claude/Anthropic. Mô hình này hoạt động dựa trên nguyên tắc phản hồi lặp đi lặp lại, nơi một mô hình ngôn ngữ lớn (LLM) đóng vai trò là "người tạo" (optimizer) để tạo ra phản hồi hoặc giải pháp, và một LLM khác đóng vai trò là "người đánh giá" (evaluator) để cung cấp đánh giá và phản hồi về chất lượng của giải pháp đó. Quá trình này lặp lại cho đến khi giải pháp đạt được các tiêu chí chất lượng mong muốn hoặc đáp ứng điều kiện dừng đã định.

Cụ thể, trong vòng lặp này:

- **Người tạo (Optimizer/Generator):** Là một LLM chịu trách nhiệm tạo ra các phản hồi, giải pháp, hoặc thực hiện các hành động dựa trên yêu cầu ban đầu và bất kỳ phản hồi nào từ người đánh giá. Nó học hỏi và điều chỉnh đầu ra của mình qua mỗi lần lặp.
- **Người đánh giá (Evaluator):** Là một LLM khác (hoặc một hệ thống đánh giá dựa trên quy tắc/mã) có nhiệm vụ kiểm tra đầu ra của người tạo dựa trên các tiêu chí đánh giá rõ ràng. Người đánh giá cung cấp một "điểm số" hoặc "phản hồi" chi tiết, chỉ ra những điểm cần cải thiện.

Anthropic nhấn mạnh rằng vòng lặp Evaluator-Optimizer đặc biệt hiệu quả khi có các tiêu chí đánh giá rõ ràng và khi việc tinh chỉnh lặp đi lặp lại mang lại giá trị đáng kể. Hai dấu hiệu cho thấy sự phù hợp tốt là: (1) phản hồi của LLM có thể được cải thiện rõ rệt khi có phản hồi từ con người, và (2) LLM có khả năng tự cung cấp phản hồi có ý nghĩa. Điều này tương tự như quá trình viết lặp đi lặp lại mà một nhà văn con người thực hiện để tạo ra một tài liệu hoàn chỉnh.

2. Thực tiễn tốt nhất (Best Practices)

1. **Tiêu chí đánh giá rõ ràng:** Các tiêu chí cụ thể, đo lường được và không mơ hồ là then chốt. Sử dụng **rubric-based evaluation** (đánh giá dựa trên tiêu chí) hiệu quả, có thể là thang điểm hoặc kiểm tra đạt/không đạt. Anthropic khuyến nghị rubric rõ ràng, giám khảo độc lập và hiệu chuẩn thường xuyên với đánh giá của con người.

2. **LLM chuyên biệt:** Sử dụng các LLM khác nhau cho vai trò Evaluator và Optimizer mang lại kết quả tốt hơn. Evaluator tập trung phát hiện lỗi và phản hồi xây dựng, Optimizer tập trung tạo giải pháp và tích hợp phản hồi.

3. **Giới hạn số lần lặp & Lưu trữ lịch sử:** Thiết lập điều kiện dừng rõ ràng (đạt điểm mục tiêu, số lần lặp tối đa) để kiểm soát chi phí và hiệu quả. Người tạo nên ghi nhớ lịch sử tạo và phản hồi trước đó để tránh lặp lại lỗi và cải thiện hiệu suất.

4. **Hiệu chuẩn con người:** Đặc biệt với đánh giá dựa trên mô hình, hiệu chuẩn định kỳ với đánh giá của con người rất quan trọng để đảm bảo tính chính xác và phù hợp của tiêu chí.

3. Hướng dẫn từng bước (Step-by-step Guide)

Để triển khai vòng lặp Evaluator-Optimizer:

1. **Xác định Nhiệm vụ & Mục tiêu:** Rõ ràng hóa nhiệm vụ và mục tiêu chất lượng của tác nhân AI.

2. **Thiết kế Prompt Generator:** Xây dựng prompt hướng dẫn LLM tạo giải pháp ban đầu, bao gồm mô tả nhiệm vụ và định dạng đầu ra.

3. **Thiết kế Prompt Evaluator:** Xây dựng prompt hướng dẫn LLM đánh giá giải pháp dựa trên tiêu chí (rubric) và định dạng phản hồi.

4. **Khởi tạo & Lặp lại:** Thực hiện lần tạo đầu tiên bằng cách gọi LLM Generator, lưu trữ kết quả. Sau đó, gọi LLM Evaluator để đánh giá. Nếu “PASS”, vòng lặp kết thúc. Nếu “NEEDS_IMPROVEMENT” hoặc “FAIL”, sử dụng phản hồi làm ngữ cảnh cho Generator và lặp lại.

5. **Thiết lập Điều kiện Dừng:** Đảm bảo vòng lặp có các điều kiện dừng để ngăn chặn việc chạy vô hạn.

4. Ví dụ thực tế (Use Cases)

1. **Dịch thuật văn học:** Optimizer (LLM dịch thuật) tạo bản dịch. Evaluator (LLM khác) đánh giá sắc thái văn hóa, phong cách, độ chính xác, cung cấp phản hồi để tinh chỉnh. Vòng lặp tiếp diễn cho đến khi đạt chất lượng cao.
2. **Tối ưu hóa mã nguồn:** Optimizer (LLM như Claude Code) tạo mã. Evaluator (LLM hoặc hệ thống kiểm tra tự động) đánh giá mã về lỗi, phong cách, hiệu suất, cung cấp phản hồi chi tiết. Vòng lặp tiếp diễn cho đến khi mã đáp ứng tiêu chí.

5. Các lỗi thường gặp và cách tránh

1. **Tiêu chí đánh giá không rõ ràng/mâu thuẫn:** Dẫn đến phản hồi mơ hồ.
    - **Cách tránh:** Định nghĩa rubric chi tiết, cụ thể. Hiệu chuẩn Evaluator với đánh giá của con người.
2. **Vòng lặp vô tận/không hội tụ & Chi phí/độ trễ cao:** Optimizer không học được từ phản hồi, hoặc tốn tài nguyên.
    - **Cách tránh:** Thiết lập điều kiện dừng rõ ràng. Đảm bảo phản hồi đủ chi tiết, mang tính xây dựng. Sử dụng kỹ thuật "ghi nhớ". Tối ưu hóa prompt, sử dụng LLM nhỏ hơn.
3. **"Over-optimization"/"Gaming the Eval":** Optimizer tối ưu hóa quá mức cho tiêu chí đánh giá.
    - **Cách tránh:** Đảm bảo tiêu chí phản ánh đúng mục tiêu. Kết hợp các loại đánh giá khác. Thường xuyên xem xét, cập nhật rubric.
4. **Phản hồi không đủ chi tiết:** Optimizer khó cải thiện.
    - **Cách tránh:** Thiết kế prompt của Evaluator yêu cầu phản hồi chi tiết, có thể hành động được.

6. Kỹ thuật nâng cao

1. **Adaptive Rubrics:** Tạo bộ tiêu chí đánh giá độc đáo cho mỗi prompt/đầu vào, hữu ích cho tác vụ tác nhân không đồng nhất (ví dụ: Vertex AI).

2. **Multi-judge Consensus:** Sử dụng nhiều Evaluator để đánh giá, tổng hợp kết quả để tăng độ tin cậy.
3. **Human-in-the-Loop:** Tích hợp con người vào vòng lặp để cung cấp phản hồi chất lượng cao hoặc tinh chỉnh trực tiếp.
4. **Tối ưu hóa dựa trên Học tăng cường (RLHF/RLAIF):** Sử dụng phản hồi từ Evaluator/con người để huấn luyện/tinh chỉnh Optimizer, giúp nó học cách tạo ra giải pháp tốt hơn.
5. **Giám sát và Phân tích Trace:** Ghi lại toàn bộ "dấu vết" của mỗi lần lặp để hiểu quá trình ra quyết định và tối ưu hóa vòng lặp.

Kết luận

Vòng lặp Đánh giá - Tối ưu là một mô hình mạnh mẽ để xây dựng các tác nhân AI có khả năng tự cải thiện và đạt được hiệu suất cao hơn thông qua phản hồi lặp đi lặp lại. Bằng cách kết hợp các LLM với vai trò chuyên biệt và áp dụng các thực tiễn tốt nhất, các nhà phát triển có thể tạo ra các hệ thống AI mạnh mẽ, linh hoạt và đáng tin cậy hơn.

> ***Khái niệm then chốt:*** *Vòng lặp Đánh giá - Tối ưu, Người tạo (Optimizer/Generator), Người đánh giá (Evaluator), Rubric-based evaluation, Điều kiện dừng (Stopping Criteria), Hiệu chuẩn với đánh giá của con người (Human Calibration), Adaptive Rubrics, Human-in-the-Loop*

Chương 7: Vòng lặp Agentic & Kỹ thuật Lập kế hoạch

1. Giới thiệu

Trong bối cảnh phát triển nhanh chóng của Trí tuệ Nhân tạo (AI), đặc biệt là các mô hình ngôn ngữ lớn (LLM), khái niệm về **Agentic Loop** (Vòng lặp Agentic) và các **Kỹ thuật Lập kế hoạch** (Planning Techniques) đã trở nên cực kỳ quan trọng. Chúng cho phép các AI agent không chỉ hiểu và tạo ra văn bản mà còn thực hiện các hành động

phức tạp, học hỏi và thích nghi trong môi trường động. Anthropic, một trong những nhà phát triển LLM hàng đầu với các mô hình Claude, đã chia sẻ nhiều hiểu biết sâu sắc về cách xây dựng các agent hiệu quả, nhấn mạnh vào sự đơn giản, khả năng kết hợp và tương tác liên tục giữa agent và môi trường [1] [2].

2. Định nghĩa Vòng lặp Agentic

Vòng lặp Agentic là cơ chế cốt lõi cho phép các AI agent thực hiện các nhiệm vụ phức tạp thông qua quá trình suy luận và hành động lặp đi lặp lại. Về bản chất, một agentic system (hệ thống agentic) là một hệ thống trong đó LLM và các công cụ được điều phối để hoàn thành một mục tiêu. Anthropic phân biệt rõ ràng giữa **Workflows** (Quy trình làm việc) và **Agents** (Tác nhân) [1]:

- **Workflows**: Là các hệ thống nơi LLM và các công cụ được điều phối thông qua các đường dẫn mã được xác định trước. Chúng mang lại tính dự đoán và nhất quán cho các tác vụ được xác định rõ ràng.
- **Agents**: Là các hệ thống nơi LLM tự động định hướng các quy trình và việc sử dụng công cụ của chúng, duy trì quyền kiểm soát cách chúng hoàn thành nhiệm vụ. Agents là lựa chọn tốt hơn khi cần sự linh hoạt và khả năng ra quyết định dựa trên mô hình ở quy mô lớn.

Trong Claude Code, vòng lặp agentic hoạt động qua ba giai đoạn chính [2]:

1. **Gather Context (Thu thập ngữ cảnh)**: Agent thu thập thông tin cần thiết từ môi trường, bao gồm các tệp, đầu ra lệnh, lịch sử trò chuyện, và các tài liệu liên quan. Đây là bước đầu tiên để hiểu rõ vấn đề và các ràng buộc.
2. **Take Action (Thực hiện hành động)**: Dựa trên ngữ cảnh đã thu thập, agent chọn và thực thi các công cụ phù hợp (ví dụ: đọc tệp, chỉnh sửa mã, chạy lệnh shell, tìm kiếm web) để tiến gần hơn đến mục tiêu.
3. **Verify Results (Xác minh kết quả)**: Sau khi thực hiện hành động, agent đánh giá kết quả để xác định xem hành động đó có thành công hay không và liệu có cần điều chỉnh gì không. Quá trình này có thể bao gồm chạy thử nghiệm, kiểm tra đầu ra hoặc tìm kiếm phản hồi.

Ba giai đoạn này hòa quyện vào nhau, tạo thành một chu trình lặp đi lặp lại. Claude sẽ quyết định những gì mỗi bước yêu cầu dựa trên những gì nó học được từ bước trước,

xâu chuỗi hàng chục hành động lại với nhau và tự điều chỉnh trong suốt quá trình [2].

3. Best Practices (Thực tiễn tốt nhất)

Anthropic khuyến nghị một số thực tiễn tốt nhất để xây dựng các AI agent hiệu quả [1] [2]:

- **Bắt đầu với sự đơn giản**: Luôn tìm kiếm giải pháp đơn giản nhất có thể và chỉ tăng độ phức tạp khi thực sự cần thiết. Đối với nhiều ứng dụng, việc tối ưu hóa các lệnh gọi LLM đơn lẻ với truy xuất và ví dụ trong ngữ cảnh là đủ.
- **Thiết kế công cụ rõ ràng và chu đáo**: Agentic Loop được cung cấp bởi các mô hình suy luận và các công cụ hành động. Việc thiết kế các bộ công cụ và tài liệu của chúng một cách rõ ràng và chu đáo là rất quan trọng. Các công cụ nên có giao diện dễ sử dụng và được ghi lại tốt cho LLM.
- **Hiểu rõ các Framework**: Mặc dù các framework có thể giúp triển khai hệ thống agentic dễ dàng hơn, nhưng chúng thường tạo ra các lớp trừu tượng bổ sung có thể che khuất các prompt và phản hồi cơ bản, khiến việc gỡ lỗi trở nên khó khăn hơn. Nên bắt đầu bằng cách sử dụng trực tiếp các API của LLM và chỉ sử dụng framework khi hiểu rõ mã cơ bản của chúng.
- **Tương tác và điều hướng**: Người dùng là một phần của vòng lặp agentic. Có thể ngắt lời agent bất cứ lúc nào để điều hướng nó theo một hướng khác, cung cấp ngữ cảnh bổ sung hoặc yêu cầu nó thử một cách tiếp cận khác. Agent hoạt động tự chủ nhưng vẫn phản hồi đầu vào của người dùng.
- **Quản lý ngữ cảnh hiệu quả**: Ngữ cảnh của Claude chứa lịch sử trò chuyện, nội dung tệp, đầu ra lệnh, và các hướng dẫn hệ thống. Khi ngữ cảnh đầy lên, Claude sẽ tự động nén lại. Để duy trì các quy tắc và thông tin quan trọng, nên đặt chúng vào tệp `CLAUDE.md` thay vì chỉ dựa vào lịch sử trò chuyện.
- **Sử dụng Checkpoints và Quyền hạn**: Claude cung cấp các cơ chế an toàn như checkpoints để hoàn tác các thay đổi tệp và quyền hạn để kiểm soát những gì Claude có thể làm mà không cần hỏi. Điều này đặc biệt quan trọng khi agent thực hiện các hành động có tác động bên ngoài (ví dụ: thay đổi hệ thống từ xa).

4. Step-by-step Guide: Vòng lặp Agentic trong Claude Code

Để minh họa vòng lặp agentic, hãy xem xét ví dụ về việc Claude Code sửa lỗi kiểm thử (failing tests) [2]:

1. **Chạy bộ kiểm thử**: Claude sẽ sử dụng một công cụ để chạy bộ kiểm thử và xác định những gì đang bị lỗi.
2. **Đọc đầu ra lỗi**: Sau khi chạy kiểm thử, Claude sẽ đọc và phân tích đầu ra lỗi để hiểu nguyên nhân gốc rễ của vấn đề.
3. **Tìm kiếm tệp nguồn liên quan**: Dựa trên thông tin lỗi, Claude sẽ tìm kiếm các tệp nguồn có khả năng liên quan đến lỗi.
4. **Đọc các tệp đó để hiểu mã**: Claude sẽ đọc nội dung của các tệp được xác định để hiểu cấu trúc mã và logic kinh doanh.
5. **Chỉnh sửa các tệp để khắc phục sự cố**: Sau khi hiểu vấn đề, Claude sẽ thực hiện các chỉnh sửa cần thiết vào mã nguồn.
6. **Chạy lại kiểm thử để xác minh**: Cuối cùng, Claude sẽ chạy lại bộ kiểm thử để xác minh rằng lỗi đã được khắc phục và không có lỗi mới nào được đưa vào.

Mỗi lần sử dụng công cụ sẽ cung cấp cho Claude thông tin mới, từ đó định hình quyết định tiếp theo của nó. Đây chính là vòng lặp agentic đang hoạt động.

5. Ví dụ thực tế (Use Cases)

Anthropic đã xác định một số quy trình làm việc (workflows) và trường hợp sử dụng (use cases) cho các hệ thống agentic [1]:

- **Prompt Chaining (Chuỗi Prompt)**: Phân tách một nhiệm vụ thành một chuỗi các bước, trong đó mỗi lệnh gọi LLM xử lý đầu ra của lệnh gọi trước đó. Hữu ích cho các tác vụ có thể phân tách rõ ràng, ví dụ: tạo bản sao tiếp thị và sau đó dịch sang ngôn ngữ khác, hoặc viết dàn ý tài liệu và sau đó viết tài liệu dựa trên dàn ý đó.
- **Routing (Định tuyến)**: Phân loại đầu vào và chuyển hướng nó đến một tác vụ tiếp theo chuyên biệt. Điều này cho phép tách biệt các mối quan tâm và xây dựng

các prompt chuyên biệt hơn. Ví dụ: định tuyến các truy vấn dịch vụ khách hàng khác nhau (câu hỏi chung, yêu cầu hoàn tiền, hỗ trợ kỹ thuật) đến các quy trình hạ nguồn khác nhau.

- **Parallelization (Song song hóa)**: LLM làm việc đồng thời trên một tác vụ và tổng hợp đầu ra của chúng. Có hai biến thể chính: **Sectioning** (Phân đoạn) – chia tác vụ thành các tác vụ con độc lập chạy song song (ví dụ: triển khai guardrails, tự động đánh giá hiệu suất LLM); và **Voting** (Bỏ phiếu) – chạy cùng một tác vụ nhiều lần để có được các đầu ra đa dạng (ví dụ: xem xét mã để tìm lỗ hổng, đánh giá nội dung không phù hợp).

- **Orchestrator-Workers (Điều phối viên-Người làm việc)**: Một LLM trung tâm tự động phân tách các tác vụ, ủy quyền chúng cho các LLM người làm việc và tổng hợp kết quả của chúng. Phù hợp cho các tác vụ phức tạp mà không thể dự đoán số lượng bước cần thiết, ví dụ: sản phẩm mã hóa thực hiện các thay đổi phức tạp cho nhiều tệp hoặc các tác vụ tìm kiếm liên quan đến việc thu thập và phân tích thông tin từ nhiều nguồn.

- **Evaluator-Optimizer (Người đánh giá-Người tối ưu hóa)**: Một lệnh gọi LLM tạo ra phản hồi trong khi một lệnh gọi khác cung cấp đánh giá và phản hồi trong một vòng lặp. Hiệu quả khi có tiêu chí đánh giá rõ ràng và khi việc tinh chỉnh lặp đi lặp lại mang lại giá trị có thể đo lường được. Ví dụ: dịch văn học hoặc các tác vụ tìm kiếm phức tạp yêu cầu nhiều vòng tìm kiếm và phân tích.

- **Autonomous Agents (Tác nhân tự trị)**: Các agent tự trị có thể xử lý các tác vụ phức tạp, mở, nơi khó hoặc không thể dự đoán số lượng bước cần thiết. Ví dụ: một agent mã hóa để giải quyết các tác vụ SWE-bench (liên quan đến chỉnh sửa nhiều tệp dựa trên mô tả tác vụ) hoặc triển khai tham chiếu “computer use” của Anthropic, nơi Claude sử dụng máy tính để hoàn thành các tác vụ.

6. Lỗi thường gặp và cách tránh

Khi xây dựng và triển khai các AI agent, có một số lỗi thường gặp cần lưu ý [1]:

- **Phức tạp hóa quá mức**: Một trong những lỗi phổ biến nhất là cố gắng xây dựng các hệ thống agentic quá phức tạp ngay từ đầu. Điều này có thể dẫn đến khó khăn trong việc gỡ lỗi, chi phí cao và hiệu suất không như mong đợi. **Cách tránh**: Bắt đầu với các giải pháp đơn giản nhất (ví dụ: tối ưu hóa các lệnh gọi LLM đơn lẻ

với truy xuất và ví dụ trong ngữ cảnh) và chỉ tăng độ phức tạp khi có bằng chứng rõ ràng về việc cải thiện hiệu suất.

- **Phụ thuộc quá nhiều vào Framework**: Các framework có thể hữu ích, nhưng chúng có thể tạo ra các lớp trừu tượng che khuất hoạt động bên trong của agent. **Cách tránh**: Luôn hiểu rõ mã cơ bản của framework và các prompt/phản hồi mà nó tạo ra. Nếu có thể, hãy bắt đầu bằng cách sử dụng trực tiếp các API của LLM.

- **Quản lý ngữ cảnh kém**: Ngữ cảnh của LLM có giới hạn và thông tin quan trọng có thể bị mất nếu không được quản lý đúng cách. **Cách tránh**: Sử dụng `CLAUDE.md` để lưu trữ các hướng dẫn và quy tắc bền vững. Tận dụng các tính năng như skills và subagents để quản lý ngữ cảnh hiệu quả hơn, đặc biệt là trong các phiên làm việc dài.

- **Thiếu cơ chế xác minh và kiểm soát**: Việc thiếu các bước xác minh kết quả hoặc khả năng kiểm soát hành động của agent có thể dẫn đến các lỗi nghiêm trọng hoặc hành vi không mong muốn. **Cách tránh**: Tích hợp chặt chẽ các bước xác minh vào vòng lặp agentic (ví dụ: chạy kiểm thử, kiểm tra đầu ra). Sử dụng các checkpoints để hoàn tác thay đổi và thiết lập quyền hạn rõ ràng cho các hành động của agent.

- **Bỏ qua phản hồi của con người**: Mặc dù agent hoạt động tự chủ, nhưng phản hồi của con người vẫn rất quan trọng để điều hướng và cải thiện hiệu suất. **Cách tránh**: Thiết kế hệ thống để cho phép người dùng dễ dàng can thiệp, cung cấp ngữ cảnh bổ sung hoặc điều chỉnh hướng đi của agent bất cứ lúc nào.

7. Kỹ thuật nâng cao

Ngoài các quy trình làm việc cơ bản, có một số kỹ thuật nâng cao có thể được áp dụng để tối ưu hóa các AI agent [1] [2]:

- **Context Engineering hiệu quả**: Tối ưu hóa cách agent thu thập và sử dụng ngữ cảnh. Điều này bao gồm việc tinh chỉnh các truy vấn tìm kiếm, lựa chọn công cụ phù hợp và xác định thông tin cần giữ lại trong bộ nhớ. Anthropic đã phát triển các giải pháp mã hóa agentic như Claude Code để thực hiện phân tích dữ liệu phức tạp trên các cơ sở dữ liệu lớn bằng cách sử dụng kỹ thuật này.

- **Sử dụng Subagents**: Đối với các nhiệm vụ phức tạp, việc phân chia công việc cho các subagents (tác nhân phụ) có thể giúp quản lý ngữ cảnh hiệu quả hơn và tăng

cường khả năng giải quyết vấn đề. Mỗi subagent có ngữ cảnh riêng, giúp tránh làm phình to ngữ cảnh của agent chính và cải thiện hiệu suất tổng thể.

- **Tích hợp Model Context Protocol (MCP)**: MCP cho phép các nhà phát triển tích hợp với một hệ sinh thái công cụ của bên thứ ba đang phát triển. Điều này mở rộng đáng kể khả năng của agent bằng cách cho phép nó tương tác với nhiều dịch vụ và nguồn dữ liệu bên ngoài.
- **Tối ưu hóa chi phí và độ trễ**: Các hệ thống agentic thường đánh đổi độ trễ và chi phí để có hiệu suất nhiệm vụ tốt hơn. Kỹ thuật nâng cao bao gồm việc sử dụng các mô hình khác nhau cho các loại tác vụ khác nhau (ví dụ: Claude Haiku cho các câu hỏi dễ, Claude Sonnet/Opus cho các tác vụ phức tạp) để tối ưu hóa chi phí và hiệu suất.
- **Kiểm thử và đánh giá liên tục**: Triển khai các quy trình kiểm thử và đánh giá tự động để liên tục đo lường hiệu suất của agent và lặp lại các cải tiến. Điều này đặc biệt quan trọng đối với các agent tự trị, nơi lỗi có thể tích lũy.

8. Kết luận

Vòng lặp Agentic và các kỹ thuật lập kế hoạch là những yếu tố then chốt trong việc xây dựng các AI agent mạnh mẽ và hiệu quả. Bằng cách hiểu rõ định nghĩa, áp dụng các best practices, tránh các lỗi thường gặp và tận dụng các kỹ thuật nâng cao, các nhà phát triển có thể tạo ra các agent có khả năng suy luận, hành động và học hỏi một cách tự chủ, mở ra nhiều ứng dụng tiềm năng trong nhiều lĩnh vực khác nhau. Anthropic, với những đóng góp và hướng dẫn của mình, đang định hình cách chúng ta tiếp cận và phát triển thế hệ AI agent tiếp theo.

9. Tài liệu tham khảo

[1] Anthropic. (2024, December 19). *Building Effective AI Agents*. Retrieved from https://www.anthropic.com/research/building-effective-agents [2] Claude Code Docs. (n.d.). *How Claude Code works*. Retrieved from https://code.claude.com/docs/en/how-claude-code-works

> ***Khái niệm then chốt:*** *Vòng lặp Agentic, Kỹ thuật Lập kế hoạch, Gather Context, Take Action, Verify Results, Workflows, Agents, Context Engineering*

Chương 8: Kỹ thuật Kỹ nghệ Ngữ cảnh (Context Engineering)

Mục tiêu của bài viết này là cung cấp một cái nhìn tổng quan toàn diện về **Kỹ thuật Kỹ nghệ Ngữ cảnh (Context Engineering)**, một lĩnh vực đang ngày càng trở nên quan trọng trong việc xây dựng các tác nhân trí tuệ nhân tạo (AI agents) hiệu quả, đặc biệt là trên hệ sinh thái của Anthropic. Trong bối cảnh các mô hình ngôn ngữ lớn (LLMs) ngày càng có khả năng tự chủ cao hơn, việc quản lý "ngữ cảnh" - tức là toàn bộ thông tin mà mô hình có thể truy cập tại một thời điểm - đã trở thành một thách thức và cơ hội lớn. Bài viết sẽ đi sâu vào các khái niệm cốt lõi, các phương pháp hay nhất, hướng dẫn từng bước, ví dụ thực tế và các cạm bẫy thường gặp, nhằm trang bị cho các nhà phát triển kiến thức cần thiết để xây dựng các tác nhân AI mạnh mẽ và đáng tin cậy hơn.

Định nghĩa Kỹ thuật Kỹ nghệ Ngữ cảnh

Kỹ thuật Kỹ nghệ Ngữ cảnh (Context Engineering) được định nghĩa là một tập hợp các chiến lược và kỹ thuật nhằm mục đích quản lý, tối ưu hóa và duy trì một bộ thông tin (tokens) hiệu quả nhất trong cửa sổ ngữ cảnh của một mô hình ngôn ngữ lớn (LLM) trong suốt quá trình suy luận. Đây được xem là sự tiến hóa tự nhiên của **Kỹ nghệ Prompt (Prompt Engineering)**. Trong khi Prompt Engineering tập trung chủ yếu vào việc thiết kế các câu lệnh (prompts) để đạt được kết quả mong muốn trong một lượt tương tác, Context Engineering mở rộng phạm vi ra toàn bộ vòng đời của một tác nhân AI, bao gồm việc quản lý trạng thái, bộ nhớ, và các nguồn thông tin bên ngoài qua nhiều lượt tương tác và trong một khoảng thời gian dài.

Ngữ cảnh không chỉ bao gồm các câu lệnh hệ thống (system prompts) mà còn cả lịch sử cuộc trò chuyện, các công cụ (tools) có sẵn, dữ liệu được truy xuất từ các nguồn bên ngoài (ví dụ như thông qua Retrieval-Augmented Generation - RAG), và trạng thái nội tại của tác nhân. Thách thức kỹ thuật ở đây là làm thế nào để tối đa hóa giá trị của từng token trong một không gian ngữ cảnh hữu hạn, nhằm đảm bảo tác nhân hoạt động một cách nhất quán và hiệu quả để đạt được mục tiêu đã định.

Các Phương pháp Hay nhất (Best Practices) trong Kỹ nghệ Ngữ cảnh

Để xây dựng các tác nhân AI hiệu quả, việc áp dụng các phương pháp hay nhất trong kỹ nghệ ngữ cảnh là vô cùng quan trọng. Dưới đây là những nguyên tắc cốt lõi được đúc

kết từ các tài liệu của Anthropic và kinh nghiệm thực tiễn của cộng đồng.

1. Coi Ngữ cảnh là một Tài nguyên Hữu hạn và Quý giá: Nguyên tắc cơ bản nhất là phải nhận thức rằng cửa sổ ngữ cảnh của LLM, dù lớn đến đâu, vẫn có giới hạn. Hiện tượng "context rot" (sự suy giảm khả năng truy xuất thông tin khi ngữ cảnh tăng lên) cho thấy việc nhồi nhét quá nhiều thông tin sẽ làm giảm hiệu suất của mô hình. Do đó, các nhà phát triển cần liên tục tìm cách tối ưu hóa, chỉ giữ lại những thông tin thực sự cần thiết và có giá trị cao nhất cho nhiệm vụ hiện tại.

2. Thiết kế System Prompts Rõ ràng và Có Cấu trúc: System prompt đóng vai trò là "kim chỉ nam" cho tác nhân. Một prompt hiệu quả cần phải rõ ràng, sử dụng ngôn ngữ trực tiếp và có cấu trúc tốt. Anthropic khuyến nghị sử dụng các thẻ XML (ví dụ: `<instructions>`, `<tools>`) hoặc tiêu đề Markdown để phân chia các phần khác nhau trong prompt. Điều này giúp mô hình dễ dàng phân tích và hiểu rõ các yêu cầu, vai trò và các ràng buộc của nó. Cần tìm sự cân bằng giữa việc cung cấp đủ chi tiết để định hướng hành vi và việc giữ cho prompt đủ linh hoạt để tác nhân có thể tự suy luận trong các tình huống không lường trước.

3. Xây dựng Bộ Công cụ (Tools) Tối giản và Hiệu quả: Công cụ là phương tiện để tác nhân tương tác với thế giới bên ngoài. Một bộ công cụ được thiết kế tốt phải đảm bảo mỗi công cụ có một chức năng rõ ràng, riêng biệt và không chồng chéo. Các bộ công cụ cồng kềnh với nhiều chức năng mơ hồ sẽ khiến tác nhân khó đưa ra quyết định đúng đắn. Tên công cụ và mô tả các tham số đầu vào phải rõ ràng, dễ hiểu đối với cả người và máy, giúp LLM lựa chọn và sử dụng công cụ một cách chính xác.

4. Áp dụng Dynamic Context Loading (Tải Ngữ cảnh Động): Thay vì tải tất cả thông tin có thể liên quan vào ngữ cảnh ngay từ đầu (eager loading), một chiến lược hiệu quả hơn là tải thông tin "just-in-time" (đúng lúc). Tác nhân chỉ nên truy xuất và đưa vào ngữ cảnh những thông tin cần thiết tại thời điểm ra quyết định. Kỹ thuật này, thường được kết hợp với Retrieval-Augmented Generation (RAG), cho phép tác nhân hoạt động trên một lượng lớn kiến thức mà không làm quá tải cửa sổ ngữ cảnh. Tác nhân sẽ duy trì các tham chiếu nhẹ (như đường dẫn tệp, ID tài liệu) và sử dụng các công cụ để tải nội dung chi tiết khi cần.

5. Triển khai các Chiến lược Quản lý Bộ nhớ Dài hạn: Đối với các tác vụ kéo dài (long-horizon tasks), việc duy trì bộ nhớ vượt ra ngoài một cửa sổ ngữ cảnh là bắt buộc. Có ba kỹ thuật chính được Anthropic đề xuất:

- **Compaction (Nén):** Khi cuộc trò chuyện gần đầy cửa sổ ngữ cảnh, tác nhân sẽ tự tóm tắt lại các thông tin quan trọng nhất và bắt đầu một phiên mới với bản tóm

tắt đó. Điều này giúp chắt lọc những chi tiết cốt lõi trong khi loại bỏ những thông tin không còn phù hợp.

- **Structured Note-taking (Ghi chú có cấu trúc):** Tác nhân được cung cấp một công cụ để ghi lại các thông tin quan trọng, kế hoạch, hoặc trạng thái công việc vào một bộ nhớ ngoài (ví dụ: một tệp `NOTES.md`). Khi cần, nó có thể đọc lại các ghi chú này để khôi phục ngữ cảnh và tiếp tục công việc một cách liền mạch.
- **Sub-agent Architectures (Kiến trúc Tác nhân phụ):** Đối với các nhiệm vụ phức tạp, có thể chia nhỏ vấn đề và giao cho các tác nhân phụ chuyên biệt. Mỗi tác nhân phụ hoạt động trong một ngữ cảnh riêng, tập trung giải quyết một phần của vấn đề và sau đó báo cáo kết quả cô đọng lại cho tác nhân chính. Điều này giúp quản lý ngữ cảnh hiệu quả và cho phép xử lý song song.

Hướng dẫn từng bước: Triển khai Dynamic Context Loading với Tool Search trên Claude

Dynamic Context Loading là một kỹ thuật mạnh mẽ cho phép tác nhân AI truy xuất thông tin một cách linh hoạt và hiệu quả, tránh việc tải quá nhiều dữ liệu vào cửa sổ ngữ cảnh. Dưới đây là hướng dẫn từng bước để triển khai kỹ thuật này, đặc biệt nhấn mạnh việc sử dụng Tool Search Tool của Anthropic:

Bước 1: Xác định các Nguồn Dữ liệu và Công cụ Tiềm năng

Trước tiên, hãy liệt kê tất cả các loại thông tin mà tác nhân của bạn có thể cần truy cập và các hành động mà nó có thể thực hiện. Ví dụ:

- **Nguồn dữ liệu:** Cơ sở dữ liệu khách hàng, tài liệu kỹ thuật, tệp nhật ký, trang web, API bên ngoài.
- **Công cụ:** Công cụ truy vấn SQL, công cụ đọc tệp (ví dụ: `read_file`), công cụ tìm kiếm web (ví dụ: `web_search`), công cụ gọi API (ví dụ: `call_api`).

Bước 2: Thiết kế Mô tả Công cụ Rõ ràng và Ngắn gọn

Đối với mỗi công cụ, hãy tạo một mô tả rõ ràng, súc tích về chức năng của nó, các tham số đầu vào cần thiết và loại đầu ra mà nó trả về. Mô tả này sẽ được LLM sử dụng để quyết định khi nào và làm thế nào để sử dụng công cụ. Ví dụ:

```
{
  "name": "read_file",
  "description": "Đọc nội dung của một tệp từ hệ thống tệp. Hữu ích khi cần
xem xét chi tiết nội dung của một tài liệu hoặc mã nguồn.",
  "input_schema": {
    "type": "object",
    "properties": {
      "file_path": {
        "type": "string",
        "description": "Đường dẫn tuyệt đối đến tệp cần đọc."
      }
    },
    "required": ["file_path"]
  }
}
```

Bước 3: Cấu hình Tool Search Tool (Nếu có)

Trong hệ sinh thái Claude, bạn có thể tận dụng **Tool Search Tool** để cho phép tác nhân động khám phá và tải các công cụ khi cần. Thay vì cung cấp tất cả các định nghĩa công cụ vào ngữ cảnh ngay từ đầu, bạn cung cấp cho Tool Search Tool một danh sách các công cụ có sẵn và mô tả của chúng. Tác nhân sẽ sử dụng Tool Search Tool để tìm công cụ phù hợp dựa trên nhiệm vụ hiện tại.

Bước 4: Xây dựng System Prompt để Hướng dẫn Tác nhân

System prompt của bạn cần hướng dẫn tác nhân cách sử dụng các công cụ và chiến lược tải ngữ cảnh động. Các điểm chính cần bao gồm:

- **Mục tiêu của tác nhân:** Xác định rõ ràng nhiệm vụ chính.
- **Hướng dẫn sử dụng công cụ:** Chỉ dẫn tác nhân sử dụng Tool Search Tool (hoặc các công cụ đã biết) để truy xuất thông tin khi cần, thay vì yêu cầu tất cả thông tin ngay lập tức.
- **Tối ưu hóa token:** Nhấn mạnh tầm quan trọng của việc chỉ đưa thông tin cần thiết vào ngữ cảnh để tiết kiệm token và duy trì sự tập trung của mô hình.
- **Xử lý kết quả:** Hướng dẫn cách xử lý kết quả từ công cụ, ví dụ: chỉ đọc các phần liên quan của tệp lớn (sử dụng `head` / `tail` hoặc tóm tắt).

Bước 5: Thực hiện Quá trình Truy xuất và Xử lý Just-in-Time

Khi tác nhân hoạt động, quy trình sẽ diễn ra như sau:

1. **Phân tích nhiệm vụ:** Tác nhân phân tích nhiệm vụ hiện tại và xác định loại thông tin hoặc hành động cần thiết.
2. **Tìm kiếm công cụ:** Nếu cần một công cụ mới, tác nhân sẽ sử dụng Tool Search Tool để tìm công cụ phù hợp nhất dựa trên mô tả công cụ.
3. **Gọi công cụ:** Tác nhân gọi công cụ đã chọn với các tham số thích hợp.
4. **Đưa kết quả vào ngữ cảnh:** Kết quả từ công cụ (ví dụ: nội dung tệp, kết quả truy vấn API) được đưa vào cửa sổ ngữ cảnh của LLM. Điều quan trọng là chỉ đưa phần thông tin liên quan nhất để tránh làm đầy ngữ cảnh.
5. **Tiếp tục suy luận:** Với thông tin mới trong ngữ cảnh, tác nhân tiếp tục quá trình suy luận để hoàn thành nhiệm vụ.

Bước 6: Lặp lại và Tinh chỉnh

Theo dõi chặt chẽ hành vi của tác nhân. Nếu tác nhân gặp khó khăn trong việc sử dụng công cụ, truy xuất thông tin không chính xác, hoặc lãng phí token, hãy tinh chỉnh các mô tả công cụ, system prompt và chiến lược xử lý kết quả. Việc này là một quá trình lặp đi lặp lại để đạt được hiệu suất tối ưu.

Ví dụ Thực tế (Use Cases)

1. Claude Code cho Phân tích Dữ liệu Lớn [1]:

Claude Code, giải pháp mã hóa tác nhân của Anthropic, là một ví dụ điển hình về việc áp dụng kỹ nghệ ngữ cảnh động. Khi được giao nhiệm vụ phân tích một tập dữ liệu lớn hoặc một cơ sở dữ liệu phức tạp, Claude Code không tải toàn bộ dữ liệu vào cửa sổ ngữ cảnh của nó. Thay vào đó, nó sử dụng các công cụ để:

- **Viết truy vấn có mục tiêu:** Tác nhân có thể tạo và thực thi các truy vấn SQL hoặc các lệnh tương tự để trích xuất chỉ những phần dữ liệu cụ thể, liên quan đến câu hỏi hiện tại.
- **Sử dụng các lệnh xử lý tệp:** Đối với các tệp dữ liệu lớn, Claude Code có thể sử dụng các lệnh Bash như `head` (đọc vài dòng đầu) hoặc `tail` (đọc vài dòng cuối) để xem xét cấu trúc hoặc các mẫu dữ liệu mà không cần tải toàn bộ tệp. Điều này giúp nó hiểu được dữ liệu mà không làm cạn kiệt token.

- **Lưu trữ kết quả tạm thời:** Các kết quả truy vấn hoặc phân tích trung gian có thể được lưu trữ vào các tệp tạm thời hoặc bộ nhớ ngoài, và chỉ những tóm tắt hoặc phần quan trọng nhất mới được đưa vào ngữ cảnh để mô hình suy luận tiếp.

Cách tiếp cận này cho phép Claude Code xử lý các tác vụ phân tích dữ liệu quy mô lớn một cách hiệu quả, vượt qua giới hạn về kích thước ngữ cảnh và mô phỏng cách một nhà khoa học dữ liệu con người sẽ tương tác với dữ liệu.

2. Tác nhân chơi Pokémon với Bộ nhớ Tác nhân [1]:

Một ví dụ minh họa khác về kỹ nghệ ngữ cảnh cho các tác vụ dài hạn là một tác nhân AI được thiết kế để chơi trò chơi Pokémon. Trò chơi này đòi hỏi sự duy trì trạng thái và chiến lược qua hàng nghìn bước. Tác nhân này sử dụng kỹ thuật **ghi chú có cấu trúc (structured note-taking)** để quản lý bộ nhớ:

- **Ghi lại tiến trình:** Tác nhân liên tục ghi lại các mục tiêu hiện tại (ví dụ: "đang huấn luyện Pokémon ở Tuyến đường 1, Pikachu đã tăng 8 cấp độ"), bản đồ các khu vực đã khám phá, các thành tích đã mở khóa và các chiến lược chiến đấu hiệu quả chống lại các loại đối thủ khác nhau.

- **Lưu trữ bên ngoài ngữ cảnh:** Những ghi chú này được lưu trữ trong một tệp `NOTES.md` hoặc một hệ thống bộ nhớ ngoài, không nằm trực tiếp trong cửa sổ ngữ cảnh của LLM.

- **Khôi phục ngữ cảnh:** Khi ngữ cảnh của LLM bị đặt lại (ví dụ: sau một phiên làm việc dài hoặc khi chuyển sang một nhiệm vụ phụ mới), tác nhân sẽ đọc lại các ghi chú này. Điều này cho phép nó khôi phục trạng thái, tiếp tục các chuỗi huấn luyện kéo dài hàng giờ hoặc khám phá ngục tối một cách liền mạch, duy trì sự mạch lạc và chiến lược dài hạn mà không bị "quên" thông tin quan trọng.

Các Lỗi Thường gặp và Cách Tránh

Việc triển khai kỹ nghệ ngữ cảnh không phải lúc nào cũng suôn sẻ. Dưới đây là một số lỗi phổ biến và cách để tránh chúng:

1. Nhồi nhét Ngữ cảnh (Context Stuffing):

- **Lỗi:** Cố gắng đưa càng nhiều thông tin càng tốt vào cửa sổ ngữ cảnh, với niềm tin rằng nhiều thông tin hơn sẽ luôn tốt hơn. Điều này dẫn đến "context rot", làm giảm khả năng tập trung và suy luận của mô hình, đồng thời tăng chi phí token.

- **Cách tránh:** Luôn coi ngữ cảnh là tài nguyên hữu hạn. Áp dụng nguyên tắc "tối thiểu cần thiết" (minimal viable context). Sử dụng các kỹ thuật tải ngữ cảnh động, nén và ghi chú có cấu trúc để chỉ đưa thông tin liên quan nhất vào ngữ cảnh tại thời điểm cần thiết.

2. Prompt Quá Cứng nhắc hoặc Quá Mơ hồ:

- **Lỗi:**
    - **Quá cứng nhắc:** Hardcoding logic phức tạp, chi tiết quá mức vào system prompt, khiến tác nhân kém linh hoạt và khó thích nghi với các tình huống mới.
    - **Quá mơ hồ:** Cung cấp hướng dẫn chung chung, không đủ cụ thể, khiến tác nhân không hiểu rõ mục tiêu hoặc cách thực hiện nhiệm vụ.
- **Cách tránh:** Tìm kiếm sự cân bằng. System prompt nên cung cấp đủ cấu trúc và hướng dẫn để định hướng hành vi, nhưng vẫn cho phép mô hình có không gian để suy luận và thích nghi. Sử dụng các thẻ XML hoặc Markdown để cấu trúc prompt một cách rõ ràng, giúp mô hình dễ dàng phân biệt các phần hướng dẫn khác nhau.

3. Bộ Công cụ Cồng kềnh và Chồng chéo:

- **Lỗi:** Cung cấp cho tác nhân một số lượng lớn công cụ với chức năng chồng chéo hoặc mô tả không rõ ràng. Điều này khiến tác nhân khó chọn đúng công cụ hoặc sử dụng chúng một cách hiệu quả, dẫn đến lãng phí token và hành vi không mong muốn.
- **Cách tránh:** Thiết kế công cụ theo nguyên tắc đơn chức năng (single responsibility principle). Mỗi công cụ nên có một mục đích rõ ràng và mô tả súc tích. Sử dụng Tool Search Tool để tác nhân chỉ tải các công cụ cần thiết vào ngữ cảnh khi chúng thực sự được yêu cầu.

4. Nén Ngữ cảnh Quá mức (Over-compaction):

- **Lỗi:** Trong nỗ lực tiết kiệm token, tóm tắt hoặc nén ngữ cảnh quá mạnh, dẫn đến mất đi các chi tiết tinh tế nhưng quan trọng mà tác nhân có thể cần sau này.
- **Cách tránh:** Cẩn thận trong việc điều chỉnh các chiến lược nén. Bắt đầu bằng cách ưu tiên khả năng thu hồi (recall) thông tin, sau đó dần dần tinh chỉnh để cải

thiện độ chính xác (precision) bằng cách loại bỏ nội dung thừa. Luôn kiểm tra kỹ lưỡng để đảm bảo không có thông tin quan trọng nào bị bỏ qua.

5. Thiếu Cơ chế Bộ nhớ Dài hạn:

- **Lỗi:** Chỉ dựa vào cửa sổ ngữ cảnh hiện tại của LLM mà không có cơ chế lưu trữ và truy xuất thông tin dài hạn. Điều này khiến tác nhân "quên" các thông tin quan trọng sau một vài lượt tương tác hoặc khi ngữ cảnh bị đặt lại, đặc biệt trong các tác vụ dài hạn.
- **Cách tránh:** Triển khai các kỹ thuật bộ nhớ tác nhân như ghi chú có cấu trúc (structured note-taking) hoặc sử dụng các hệ thống bộ nhớ ngoài. Điều này cho phép tác nhân duy trì trạng thái và kiến thức qua các phiên làm việc dài.

Kỹ thuật Nâng cao trong Kỹ nghệ Ngữ cảnh

Khi các mô hình ngôn ngữ lớn và các tác nhân AI ngày càng phát triển, các kỹ thuật kỹ nghệ ngữ cảnh cũng trở nên tinh vi hơn. Dưới đây là một số kỹ thuật nâng cao:

1. Model Context Protocol (MCP) [1]:

MCP là một giao thức được Anthropic phát triển để cho phép các tác nhân LLM tương tác với một số lượng lớn công cụ một cách hiệu quả. Thay vì tải tất cả các định nghĩa công cụ vào ngữ cảnh, MCP cho phép tác nhân:

- **Tải công cụ theo yêu cầu:** Chỉ tải các công cụ cần thiết vào ngữ cảnh khi tác nhân quyết định rằng chúng là cần thiết cho nhiệm vụ hiện tại.
- **Lọc dữ liệu trước khi đến mô hình:** MCP có thể giúp lọc và tiền xử lý dữ liệu từ các công cụ trước khi nó được đưa vào cửa sổ ngữ cảnh của LLM, giảm thiểu lượng token và tăng cường sự liên quan.

2. Dynamic Filtering trong Tìm kiếm Web của Claude [1]:

Khi thực hiện tìm kiếm web, Claude sử dụng một kỹ thuật lọc động. Thay vì tải toàn bộ nội dung HTML của một trang web vào ngữ cảnh và yêu cầu mô hình suy luận trên đó, Claude thực hiện lọc "trước khi tải kết quả vào ngữ cảnh". Điều này có nghĩa là các thông tin không liên quan hoặc dư thừa sẽ được loại bỏ trước khi chúng chiếm không gian quý giá trong cửa sổ ngữ cảnh, giúp quá trình tìm kiếm nhanh hơn và tiết kiệm chi phí hơn.

3. Position Encoding Interpolation [1]:

Đây là một kỹ thuật kiến trúc cho phép các mô hình transformer xử lý các chuỗi dài hơn so với độ dài ngữ cảnh mà chúng được huấn luyện ban đầu. Mặc dù có thể có một số suy giảm nhỏ trong việc hiểu vị trí token, kỹ thuật này giúp mở rộng khả năng của LLM trong việc xử lý các ngữ cảnh rất dài mà không cần huấn luyện lại toàn bộ mô hình từ đầu.

4. Chiến lược Lai (Hybrid Strategy) [1]:

Các tác nhân hiệu quả nhất thường không chỉ dựa vào một kỹ thuật duy nhất mà kết hợp nhiều phương pháp. Một chiến lược lai có thể bao gồm:

- **Truy xuất dữ liệu trước (Pre-computed data retrieval):** Tải một số thông tin cơ bản, thường xuyên được sử dụng vào ngữ cảnh ngay từ đầu để tăng tốc độ.
- **Khám phá tự động trong thời gian chạy (Runtime exploration):** Cho phép tác nhân tự động khám phá và truy xuất thông tin chi tiết hơn khi cần thiết thông qua các công cụ và truy vấn động.

Ví dụ, Claude Code sử dụng chiến lược này bằng cách đưa các tệp `CLAUDE.md` (chứa hướng dẫn hoặc thông tin quan trọng) vào ngữ cảnh trước, trong khi vẫn sử dụng các nguyên thủy như `glob` và `grep` để điều hướng môi trường và truy xuất các tệp khác "just-in-time".

Kết luận

Kỹ thuật Kỹ nghệ Ngữ cảnh đại diện cho một sự thay đổi cơ bản trong cách chúng ta xây dựng và tương tác với các mô hình ngôn ngữ lớn, đặc biệt là trong bối cảnh phát triển các tác nhân AI. Nó vượt ra ngoài phạm vi của kỹ nghệ prompt truyền thống, tập trung vào việc quản lý toàn diện và tối ưu hóa tài nguyên ngữ cảnh hữu hạn của LLM. Bằng cách áp dụng các chiến lược như tải ngữ cảnh động, nén thông tin, ghi chú có cấu trúc và kiến trúc tác nhân phụ, các nhà phát triển có thể xây dựng các tác nhân AI không chỉ hiệu quả hơn mà còn có khả năng duy trì sự mạch lạc và thực hiện các tác vụ phức tạp, dài hạn. Các phương pháp hay nhất và kỹ thuật nâng cao được Anthropic tiên phong đang mở đường cho một thế hệ tác nhân AI mạnh mẽ và đáng tin cậy hơn, có khả năng hoạt động tự chủ trong nhiều môi trường khác nhau.

Tài liệu tham khảo

[1] Anthropic. (2025, September 29). *Effective context engineering for AI agents*. Retrieved from https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents

> ***Khái niệm then chốt:*** *Kỹ thuật Kỹ nghệ Ngữ cảnh, Ngữ cảnh, Context Rot, Dynamic Context Loading, Compaction, Structured Note-taking, Sub-agent Architectures, Model Context Protocol*

Chương 9: Cấu trúc & Quy chuẩn Skills (Agent Skills Standard)

1. Giới thiệu

Trong bối cảnh phát triển nhanh chóng của Trí tuệ Nhân tạo (AI), đặc biệt là các mô hình ngôn ngữ lớn (LLMs), việc xây dựng các tác nhân (agents) có khả năng thực hiện các tác vụ phức tạp một cách tự chủ đang trở thành một lĩnh vực trọng tâm. Để các tác nhân này có thể hoạt động hiệu quả, chúng cần được trang bị các "kỹ năng" chuyên biệt. Agent Skills Standard (agentskills.io) ra đời như một khuôn khổ mở, nhẹ nhàng nhằm định nghĩa và chuẩn hóa cách thức các kỹ năng này được tạo, đóng gói và sử dụng bởi các tác nhân AI. Tiêu chuẩn này đặc biệt được Anthropic, nhà phát triển mô hình Claude, áp dụng và thúc đẩy, giúp các tác nhân Claude mở rộng khả năng và thực hiện các quy trình làm việc phức tạp một cách nhất quán.

2. Định nghĩa Agent Skills

Agent Skill là một thư mục chứa các hướng dẫn, tập lệnh (scripts) và tài nguyên mà các tác nhân AI có thể khám phá và sử dụng để thực hiện các tác vụ một cách chính xác và hiệu quả hơn [1]. Về cơ bản, một kỹ năng là một tập hợp các chỉ dẫn được đóng gói dưới dạng một thư mục đơn giản, dạy cho Claude cách xử lý các tác vụ hoặc quy trình làm việc cụ thể [2].

Các kỹ năng này giải quyết vấn đề thiếu ngữ cảnh mà các tác nhân thường gặp phải khi thực hiện công việc thực tế. Bằng cách cung cấp kiến thức thủ tục và ngữ cảnh cụ thể của công ty, nhóm hoặc người dùng, kỹ năng cho phép các tác nhân mở rộng khả năng của chúng dựa trên tác vụ đang thực hiện [1].

2.1. Cấu trúc thư mục của một Skill

Một kỹ năng tối thiểu là một thư mục chứa tệp `SKILL.md`. Ngoài ra, nó có thể bao gồm các thư mục tùy chọn khác để hỗ trợ chức năng của kỹ năng [3]:

```
my-skill/
├── SKILL.md          # Bắt buộc: hướng dẫn + metadata
├── scripts/          # Tùy chọn: mã thực thi (Python, Bash, v.v.)
├── references/       # Tùy chọn: tài liệu được tải khi cần
└── assets/           # Tùy chọn: mẫu, phông chữ, biểu tượng được sử dụng
trong đầu ra
```

- **`SKILL.md`**: Đây là tệp cốt lõi, chứa các hướng dẫn chính cho tác nhân và siêu dữ liệu (metadata) của kỹ năng. Nó phải tuân thủ định dạng YAML frontmatter theo sau là nội dung Markdown [3].
- **`scripts/`**: Chứa mã thực thi mà tác nhân có thể chạy. Các tập lệnh này nên tự chứa hoặc tài liệu rõ ràng về các phụ thuộc, bao gồm các thông báo lỗi hữu ích và xử lý các trường hợp ngoại lệ một cách khéo léo [3].
- **`references/`**: Chứa tài liệu bổ sung mà tác nhân có thể đọc khi cần. Các tệp tham chiếu nên được giữ tập trung và nhỏ gọn vì tác nhân sẽ tải chúng theo yêu cầu [3].
- **`assets/`**: Chứa các tài nguyên tĩnh như mẫu tài liệu, mẫu cấu hình, hình ảnh (sơ đồ, ví dụ) hoặc tệp dữ liệu (bảng tra cứu, lược đồ) [3].

2.2. Định dạng tệp SKILL.md

Tệp `SKILL.md` phải chứa YAML frontmatter theo sau là nội dung Markdown. Phần frontmatter là bắt buộc và chứa các trường siêu dữ liệu quan trọng [3]:

```
---
name: pdf-processing
description: Extract text and tables from PDF files, fill forms, merge
documents.
license: Apache-2.0
metadata:
  author: example-org
  version: "1.0"
allowed-tools: Bash(git:*) Bash(jq:*) Read
---


# PDF Processing


## When to use this skill

Use this skill when the user needs to work with PDF files...


## How to extract text

1. Use pdfplumber for text extraction...


## How to fill forms

...
```

Các trường bắt buộc trong Frontmatter:

- **`name`**: Một định danh ngắn gọn cho kỹ năng. Phải là 1-64 ký tự, chỉ chứa các ký tự chữ và số unicode viết thường và dấu gạch nối (`a-z` và `-`), không được bắt đầu hoặc kết thúc bằng dấu gạch nối, và không chứa dấu gạch nối liên tiếp (`--`). Tên này phải khớp với tên thư mục cha [3].

- `description`: Mô tả về những gì kỹ năng thực hiện và khi nào nên sử dụng nó. Phải là 1-1024 ký tự, nên bao gồm các từ khóa cụ thể giúp tác nhân xác định các tác vụ liên quan [3].

Các trường tùy chọn trong Frontmatter:

- `license`: Chỉ định giấy phép áp dụng cho kỹ năng [3].
- `compatibility`: Chỉ ra các yêu cầu môi trường cụ thể của kỹ năng (ví dụ: sản phẩm dự định, gói hệ thống cần thiết, nhu cầu truy cập mạng) [3].
- `metadata`: Một bản đồ từ khóa chuỗi đến giá trị chuỗi để lưu trữ các thuộc tính bổ sung không được định nghĩa bởi đặc tả Agent Skills [3].
- `allowed-tools`: Một danh sách các công cụ được phê duyệt trước để chạy. Đây là một trường thử nghiệm và sự hỗ trợ có thể khác nhau giữa các triển khai tác nhân [3].

Phần nội dung Markdown sau frontmatter chứa các hướng dẫn của kỹ năng. Không có hạn chế về định dạng, nhưng nên bao gồm các hướng dẫn từng bước, ví dụ về đầu vào và đầu ra, và các trường hợp ngoại lệ phổ biến [3].

3. Cách thức hoạt động của Skills (Progressive Disclosure)

Agent Skills sử dụng cơ chế **tiết lộ dần dần (progressive disclosure)** để quản lý ngữ cảnh một cách hiệu quả, giảm thiểu việc sử dụng token trong khi vẫn duy trì chuyên môn hóa [2] [3]:

1. **Khám phá (Discovery)**: Khi khởi động, các tác nhân chỉ tải tên và mô tả của mỗi kỹ năng có sẵn. Điều này cung cấp đủ thông tin để tác nhân biết khi nào kỹ năng đó có thể liên quan [2] [3].
2. **Kích hoạt (Activation)**: Khi một tác vụ khớp với mô tả của kỹ năng, tác nhân sẽ đọc toàn bộ hướng dẫn `SKILL.md` vào ngữ cảnh [2] [3].
3. **Thực thi (Execution)**: Tác nhân tuân theo các hướng dẫn, tùy chọn tải các tệp tham chiếu hoặc thực thi mã được đóng gói khi cần [2] [3].

Cách tiếp cận này giúp các tác nhân nhanh chóng trong khi vẫn cho phép chúng truy cập vào nhiều ngữ cảnh hơn theo yêu cầu [2].

4. Best Practices (Thực tiễn tốt nhất)

Anthropic và cộng đồng đã đưa ra nhiều thực tiễn tốt nhất để xây dựng các kỹ năng hiệu quả cho Claude [2]:

1. **Mô tả rõ ràng và cụ thể**: Trường `description` trong `SKILL.md` là cực kỳ quan trọng. Nó phải mô tả rõ ràng kỹ năng làm gì và khi nào nên sử dụng, bao gồm các từ khóa kích hoạt cụ thể mà người dùng có thể nói. Tránh các mô tả quá chung chung hoặc quá kỹ thuật [2] [3].
    - **Ví dụ tốt**: `description: Extracts text and tables from PDF files, fills PDF forms, and merges multiple PDFs. Use when working with PDF documents or when the user mentions PDFs, forms, or document extraction.`
    - **Ví dụ kém**: `description: Helps with PDFs.`
2. **Cấu trúc hướng dẫn rõ ràng và dễ thực hiện**: Nội dung Markdown của `SKILL.md` nên được tổ chức tốt với các hướng dẫn từng bước, ví dụ và xử lý lỗi. Sử dụng các tiêu đề, danh sách gạch đầu dòng và danh sách được đánh số để cải thiện khả năng đọc [2] [3].
3. **Sử dụng tiết lộ dần dần hiệu quả**: Giữ tệp `SKILL.md` tập trung vào các hướng dẫn cốt lõi (dưới 500 dòng hoặc 5000 token được khuyến nghị). Di chuyển tài liệu tham khảo chi tiết, tập lệnh và tài nguyên tĩnh sang các thư mục `references/`, `scripts/` và `assets/` tương ứng. Tác nhân sẽ chỉ tải chúng khi cần, giúp giảm thiểu việc sử dụng token [2] [3].
4. **Xử lý lỗi và trường hợp ngoại lệ**: Bao gồm các hướng dẫn rõ ràng về cách xử lý lỗi và các trường hợp ngoại lệ phổ biến. Đối với các xác thực quan trọng, hãy xem xét việc đóng gói một tập lệnh thực hiện kiểm tra theo chương trình thay vì chỉ dựa vào hướng dẫn ngôn ngữ [2].
5. **Kiểm thử và lặp lại**: Kiểm thử kỹ năng của bạn ở các mức độ khác nhau, từ kiểm thử thủ công trong Claude.ai đến kiểm thử tự động. Tập trung vào việc kích hoạt

kỹ năng đúng lúc, chức năng chính xác và cải thiện hiệu suất so với không có kỹ năng [2].

5. Step-by-step Guide: Tạo một Skill đơn giản

Đây là hướng dẫn từng bước để tạo một kỹ năng đơn giản cho Claude, ví dụ: một kỹ năng để tóm tắt các bài báo khoa học.

Bước 1: Xác định trường hợp sử dụng và tên kỹ năng

- **Trường hợp sử dụng**: Tóm tắt các bài báo khoa học dài thành các điểm chính ngắn gọn.
- **Tên kỹ năng**: `summarize-scientific-paper` (tuân thủ kebab-case).

Bước 2: Tạo cấu trúc thư mục

Tạo một thư mục với tên kỹ năng đã chọn:

```
mkdir summarize-scientific-paper
cd summarize-scientific-paper
```

Bước 3: Tạo tệp SKILL.md

Tạo tệp `SKILL.md` bên trong thư mục `summarize-scientific-paper` và thêm nội dung sau:

```
---
name: summarize-scientific-paper
description: Tóm tắt các bài báo khoa học, nghiên cứu, hoặc tài liệu kỹ thuật dài thành các điểm chính ngắn gọn và dễ hiểu. Sử dụng khi người dùng yêu cầu "tóm tắt bài báo", "trích xuất ý chính", hoặc "hiểu nhanh tài liệu khoa học".
metadata:
  author: Manus AI
  version: "1.0"
---


# Tóm tắt Bài báo Khoa học


## Khi nào sử dụng kỹ năng này

Sử dụng kỹ năng này khi bạn cần chắt lọc thông tin từ các bài báo khoa học, nghiên cứu, hoặc tài liệu kỹ thuật có độ dài đáng kể. Kỹ năng này sẽ giúp bạn nắm bắt các luận điểm chính, phương pháp, kết quả và kết luận mà không cần đọc toàn bộ tài liệu.


## Cách tóm tắt


1. **Xác định mục tiêu**: Hỏi người dùng về mục tiêu tóm tắt (ví dụ: cần tóm tắt cho ai, độ dài mong muốn, các khía cạnh cần tập trung).

2. **Đọc và phân tích**: Đọc lướt qua bài báo để hiểu cấu trúc tổng thể, sau đó tập trung vào phần Giới thiệu, Tóm tắt (Abstract), Kết luận (Conclusion), và các tiêu đề chính.

3. **Trích xuất thông tin chính**: Xác định các luận điểm chính, phương pháp nghiên cứu, kết quả quan trọng và kết luận. Ghi chú các thuật ngữ kỹ thuật quan trọng.

4. **Tổng hợp**: Viết một bản tóm tắt mạch lạc, súc tích, tập trung vào các thông tin đã trích xuất. Đảm bảo bản tóm tắt phản ánh đúng nội dung và giọng điệu của bài báo gốc.

5. **Rà soát**: Kiểm tra lại bản tóm tắt để đảm bảo tính chính xác, đầy đủ và dễ hiểu. Loại bỏ các thông tin thừa hoặc lặp lại.
```

```
## Ví dụ

**Người dùng nói**: "Hãy tóm tắt bài báo về 'Deep Learning for Natural Language Processing' này cho tôi."

**Hành động của Claude**:

1. Kích hoạt kỹ năng `summarize-scientific-paper`.

2. Yêu cầu người dùng cung cấp bài báo hoặc liên kết đến bài báo.

3. Thực hiện các bước tóm tắt đã định nghĩa.

4. Trả về bản tóm tắt cho người dùng.


## Các lỗi thường gặp và cách tránh


*   **Tóm tắt quá dài hoặc quá ngắn**: Điều chỉnh bước 1 để hỏi rõ ràng về độ dài mong muốn.

*   **Bỏ sót thông tin quan trọng**: Đảm bảo bước 3 bao gồm việc xác định các phần cốt lõi của bài báo.

*   **Tóm tắt không khách quan**: Tập trung vào việc trích xuất thông tin trực tiếp từ bài báo thay vì diễn giải quá mức.
```

Bước 4: (Tùy chọn) Thêm Script hoặc Reference

Nếu kỹ năng này cần một tập lệnh Python để xử lý văn bản hoặc một tệp tham chiếu với các hướng dẫn chi tiết hơn về các kỹ thuật tóm tắt khác nhau, bạn có thể tạo các thư mục `scripts/` và `references/`.

```
mkdir scripts
mkdir references
```

Ví dụ, tạo `scripts/text_processor.py`:

```
# scripts/text_processor.py

def process_text(text):
    # Logic xử lý văn bản, ví dụ: làm sạch, phân tích cú pháp

    return text.upper()
```

Và `references/summarization_techniques.md`:

```
# Kỹ thuật Tóm tắt Nâng cao

## Tóm tắt trích xuất (Extractive Summarization)

...

## Tóm tắt trừu tượng (Abstractive Summarization)

...
```

Bước 5: Kiểm thử kỹ năng

Sử dụng Claude.ai hoặc Claude Code để tải lên và kiểm thử kỹ năng của bạn. Đảm bảo rằng Claude kích hoạt kỹ năng khi bạn đưa ra các yêu cầu liên quan và thực hiện các bước như mong đợi.

6. Ví dụ thực tế (Use Cases)

1. **Tạo tài liệu và tài sản (Document & Asset Creation)**: Các kỹ năng có thể được sử dụng để tạo ra các tài liệu, bản trình bày, ứng dụng hoặc mã nguồn nhất quán, chất lượng cao. Ví dụ, một kỹ năng `frontend-design` có thể tạo ra các giao diện người dùng dựa trên các nguyên tắc thiết kế và tiêu chuẩn thương hiệu đã được nhúng [2].

2. **Tự động hóa quy trình làm việc (Workflow Automation)**: Đối với các quy trình nhiều bước yêu cầu phương pháp luận nhất quán, kỹ năng có thể tự động hóa chúng. Ví dụ, một kỹ năng `skill-creator` có thể hướng dẫn người dùng qua quy trình định nghĩa trường hợp sử dụng, tạo frontmatter và viết hướng dẫn cho một kỹ năng mới [2].

3. **Nâng cao tích hợp MCP (MCP Enhancement)**: Kỹ năng cung cấp hướng dẫn quy trình làm việc để nâng cao khả năng truy cập công cụ mà một máy chủ MCP cung cấp. Ví dụ, một kỹ năng `sentry-code-review` có thể tự động phân tích và sửa lỗi được phát hiện trong GitHub Pull Requests bằng cách sử dụng dữ liệu giám sát lỗi của Sentry thông qua máy chủ MCP của họ [2].

7. Các lỗi thường gặp và cách tránh

- **Kỹ năng không tải lên được**: Thường do tên tệp `SKILL.md` không chính xác (phải là `SKILL.md` chính xác, phân biệt chữ hoa chữ thường) hoặc lỗi định dạng YAML frontmatter. Đảm bảo tuân thủ nghiêm ngặt các quy tắc đặt tên và định dạng [2].
- **Kỹ năng không kích hoạt**: Mô tả (description) quá chung chung hoặc không chứa các từ khóa kích hoạt mà người dùng có thể sử dụng. Cần làm cho mô tả cụ thể hơn, bao gồm cả những gì kỹ năng làm và khi nào nên sử dụng, với các cụm từ kích hoạt rõ ràng [2].
- **Kỹ năng kích hoạt quá thường xuyên**: Mô tả quá rộng, khiến kỹ năng được kích hoạt cho các truy vấn không liên quan. Thêm các từ khóa phủ định hoặc làm cho mô tả cụ thể hơn về phạm vi của kỹ năng [2].
- **Claude không tuân theo hướng dẫn**: Hướng dẫn quá dài dòng, bị chôn vùi trong văn bản, hoặc ngôn ngữ mơ hồ. Giữ hướng dẫn ngắn gọn, sử dụng danh sách, đặt các hướng dẫn quan trọng lên đầu và sử dụng ngôn ngữ rõ ràng, không mơ hồ [2].
- **Vấn đề về ngữ cảnh lớn**: Nội dung kỹ năng quá lớn (tệp `SKILL.md` quá dài) hoặc quá nhiều kỹ năng được bật cùng lúc có thể làm chậm Claude hoặc làm giảm chất lượng phản hồi. Tối ưu hóa kích thước `SKILL.md` bằng cách di chuyển tài liệu chi tiết sang `references/` và giảm số lượng kỹ năng được bật đồng thời [2].

8. Kỹ thuật nâng cao

- **Phối hợp quy trình làm việc tuần tự (Sequential workflow orchestration)**: Thiết kế kỹ năng để thực hiện các quy trình nhiều bước theo một trình tự cụ thể, với các phụ thuộc giữa các bước, xác thực ở mỗi giai đoạn và hướng dẫn hoàn tác cho các lỗi [2].
- **Phối hợp đa MCP (Multi-MCP coordination)**: Xây dựng kỹ năng để điều phối các quy trình làm việc trải rộng trên nhiều dịch vụ (ví dụ: xuất thiết kế từ Figma, lưu trữ tài sản trong Drive, tạo tác vụ trong Linear và thông báo qua Slack) [2].
- **Lựa chọn công cụ nhận biết ngữ cảnh (Context-aware tool selection)**: Cho phép kỹ năng chọn công cụ phù hợp dựa trên ngữ cảnh (ví dụ: lưu trữ tệp lớn vào bộ nhớ đám mây, tài liệu cộng tác vào Notion/Docs, tệp mã vào GitHub) [2].
- **Tích hợp trí tuệ chuyên biệt theo lĩnh vực (Domain-specific intelligence)**: Nhúng kiến thức chuyên môn theo lĩnh vực vào logic của kỹ năng, ví dụ: kiểm tra tuân thủ tài chính trước khi xử lý thanh toán [2].

9. Kết luận

Agent Skills Standard cung cấp một khuôn khổ mạnh mẽ và linh hoạt để mở rộng khả năng của các tác nhân AI như Claude. Bằng cách tuân thủ các quy chuẩn về cấu trúc thư mục, định dạng `SKILL.md` và áp dụng các thực tiễn tốt nhất, nhà phát triển có thể tạo ra các kỹ năng hiệu quả, đáng tin cậy, giúp Claude thực hiện các tác vụ phức tạp một cách tự chủ và nhất quán. Cơ chế tiết lộ dần dần đảm bảo hiệu quả về token, trong khi khả năng tích hợp với các công cụ và dịch vụ bên ngoài thông qua MCP mở ra vô số trường hợp sử dụng trong thế giới thực.

10. Tài liệu tham khảo

[1] Agent Skills: Overview. (n.d.). Retrieved from https://agentskills.io/ [2] Anthropic. (n.d.). *The Complete Guide to Building Skills for Claude*. Retrieved from https://resources.anthropic.com/hubfs/The-Complete-Guide-to-Building-Skill-for-Claude.pdf?hsLang=en [3] What are skills? - Agent Skills. (n.d.). Retrieved from https://agentskills.io/what-are-skills [4] Specification - Agent Skills. (n.d.). Retrieved from https://agentskills.io/specification

> ***Khái niệm then chốt:*** *Agent Skill, SKILL.md, Progressive Disclosure, YAML Frontmatter, Kebab-case*

Chương 10: Thiết kế & Viết SKILL.md hiệu quả

1. Giới thiệu

Trong bối cảnh phát triển nhanh chóng của Trí tuệ Nhân tạo (AI), các AI Agent đóng vai trò ngày càng quan trọng trong việc tự động hóa và thực hiện các tác vụ phức tạp. Để các agent này có thể hoạt động hiệu quả, việc trang bị cho chúng những kỹ năng chuyên biệt là điều cần thiết. SKILL.md là một thành phần cốt lõi trong hệ sinh thái Claude/Anthropic, cho phép định nghĩa và tổ chức các khả năng này một cách rõ ràng và hiệu quả. Báo cáo này sẽ đi sâu vào các nguyên tắc, thực tiễn tốt nhất, hướng dẫn từng bước, ví dụ thực tế, các lỗi thường gặp và kỹ thuật nâng cao để thiết kế và viết SKILL.md, nhằm tối ưu hóa hiệu suất của AI Agents.

2. Định nghĩa SKILL.md và các thuật ngữ liên quan

SKILL.md là một tệp Markdown chứa các hướng dẫn, quy trình, ví dụ và tài nguyên mà một AI Agent (cụ thể là Claude) có thể đọc và sử dụng để thực hiện các tác vụ chuyên biệt. Nó hoạt động như một bản mô tả chi tiết về một "kỹ năng" mà agent sở hữu.

Agent Skills là các khả năng hoặc chuyên môn được định nghĩa thông qua SKILL.md và các tệp liên quan, cho phép Claude mở rộng phạm vi hoạt động của mình ngoài các khả năng vốn có của mô hình ngôn ngữ lớn (LLM). Các kỹ năng này có thể bao gồm từ việc xử lý tài liệu, phân tích dữ liệu đến tương tác với các công cụ bên ngoài.

YAML Frontmatter là một khối dữ liệu YAML tùy chọn nằm ở đầu tệp Markdown, được sử dụng để lưu trữ siêu dữ liệu (metadata) về kỹ năng. Trong ngữ cảnh SKILL.md, nó chứa các trường bắt buộc như `name` (tên kỹ năng) và `description` (mô tả kỹ năng), giúp Claude nhanh chóng nhận diện và hiểu mục đích của kỹ năng.

Context Window (Cửa sổ ngữ cảnh) là giới hạn về lượng thông tin mà một mô hình AI có thể xử lý tại một thời điểm. Khi Claude sử dụng một kỹ năng, nội dung của SKILL.md và các tệp liên quan sẽ chiếm một phần trong cửa sổ ngữ cảnh này. Việc tối ưu hóa SKILL.md giúp giảm thiểu chi phí token và đảm bảo Claude có đủ không gian để xử lý các thông tin quan trọng khác như lịch sử hội thoại và yêu cầu của người dùng.

Progressive Disclosure (Tiết lộ lũy tiến) là một nguyên tắc thiết kế hướng dẫn, trong đó thông tin được trình bày theo từng lớp, từ tổng quan đến chi tiết. Trong SKILL.md, điều này có nghĩa là tệp chính SKILL.md cung cấp một cái nhìn tổng thể và chỉ dẫn Claude đến các tệp tài nguyên chi tiết hơn khi cần, giúp quản lý hiệu quả cửa sổ ngữ cảnh.

3. Các Nguyên tắc Cốt lõi để Viết SKILL.md Hiệu quả (Best Practices)

3.1. Súc tích là chìa khóa (Concise is Key)

Một SKILL.md hiệu quả cần phải súc tích. Claude đã là một mô hình rất thông minh, do đó, chỉ nên cung cấp những thông tin mà Claude chưa có. Mỗi token trong SKILL.md đều có giá trị, vì nó cạnh tranh với các thông tin khác trong cửa sổ ngữ cảnh. Khi Claude tải một kỹ năng, mọi token trong đó đều chiếm không gian. Do đó, cần loại bỏ các giải thích không cần thiết và tập trung vào những hướng dẫn cốt lõi [1].

- **Ví dụ tốt (súc tích)**:

  ```
  ## Extract PDF text


  Use pdfplumber for text extraction:
  ```python
  import pdfplumber
  with pdfplumber.open("file.pdf") as pdf:
      text = pdf.pages[0].extract_text()
  ```
  ```

  „`

- **Ví dụ tệ (quá dài dòng)**:

```
## Extract PDF text


PDF (Portable Document Format) files are a common file format that
contains
text, images, and other content. To extract text from a PDF, you\'ll
need to
use a library. There are many libraries available for PDF processing,
but
pdfplumber is recommended because it\'s easy to use and handles most
cases well.
First, you\'ll need to install it using pip. Then you can use the code
below...
```

3.2. Đặt mức độ tự do phù hợp (Set Appropriate Degrees of Freedom)

Độ chi tiết của hướng dẫn trong SKILL.md cần phù hợp với tính chất của tác vụ. Một số tác vụ yêu cầu hướng dẫn rất cụ thể, trong khi những tác vụ khác có thể linh hoạt hơn [1].

- **High freedom (tự do cao)**: Sử dụng khi có nhiều cách tiếp cận hợp lệ, quyết định phụ thuộc vào ngữ cảnh, hoặc khi cần các heuristic để hướng dẫn. Ví dụ: quy trình đánh giá mã (code review).
- **Medium freedom (tự do trung bình)**: Áp dụng khi có một mẫu ưu tiên nhưng vẫn chấp nhận một số biến thể, hoặc khi cấu hình ảnh hưởng đến hành vi. Ví dụ: tạo báo cáo với các tham số tùy chỉnh.
- **Low freedom (tự do thấp)**: Dành cho các hoạt động dễ xảy ra lỗi, yêu cầu tính nhất quán cao, hoặc phải tuân theo một trình tự cụ thể. Ví dụ: di chuyển cơ sở dữ liệu (database migration) cần chạy chính xác một script.

3.3. Kiểm thử với tất cả các mô hình dự định sử dụng (Test with All Models You Plan to Use)

Hiệu quả của một kỹ năng phụ thuộc vào mô hình AI cơ bản. Do đó, cần kiểm thử kỹ năng với tất cả các phiên bản Claude (Haiku, Sonnet, Opus) mà bạn dự định sử dụng. Một hướng dẫn hoàn hảo cho Claude Opus có thể cần chi tiết hơn cho Claude Haiku để đảm bảo hiệu quả [1].

3.4. Sử dụng tên và mô tả hiệu quả (Effective Naming and Descriptions)

Quy ước đặt tên: Tên kỹ năng (trong YAML frontmatter) nên nhất quán và dễ hiểu. Nên sử dụng dạng gerund (động từ + -ing) để mô tả rõ ràng hoạt động hoặc khả năng của kỹ năng. Tên phải viết thường, chỉ chứa chữ cái, số và dấu gạch ngang, tối đa 64 ký tự, và không chứa các từ dành riêng như “anthropic” hoặc “claude” [1].

- **Ví dụ tốt**: `processing-pdfs`, `analyzing-spreadsheets`.
- **Tránh**: `helper`, `utils`, `documents`.

Viết mô tả hiệu quả: Trường `description` trong YAML frontmatter rất quan trọng cho việc khám phá kỹ năng. Nó phải mô tả rõ ràng kỹ năng làm gì và khi nào nên sử dụng, được viết ở ngôi thứ ba, cụ thể và bao gồm các từ khóa kích hoạt. Độ dài tối đa là 1024 ký tự và không chứa các thẻ XML [1].

- **Ví dụ tốt**: `description: Extract text and tables from PDF files, fill forms, merge documents. Use when working with PDF files or when the user mentions PDFs, forms, or document extraction.`
- **Tránh**: `description: Helps with documents`.

3.5. Áp dụng mô hình tiết lộ lũy tiến (Progressive Disclosure Patterns)

SKILL.md nên hoạt động như một bản tóm tắt, chỉ dẫn Claude đến các tài liệu chi tiết hơn khi cần. Điều này giúp giữ cho tệp SKILL.md chính gọn gàng và quản lý cửa sổ ngữ cảnh hiệu quả. Nên giữ nội dung chính của SKILL.md dưới 500 dòng. Nếu nội dung vượt quá giới hạn này, hãy chia nhỏ thành các tệp riêng biệt và tham chiếu chúng từ SKILL.md [1].

4. Cấu trúc SKILL.md

Một tệp SKILL.md bao gồm YAML Frontmatter ở đầu và phần nội dung Markdown. Cấu trúc thư mục cũng đóng vai trò quan trọng trong cách Claude truy cập và sử dụng các tệp kỹ năng [1].

YAML Frontmatter: Như đã đề cập, chứa `name` và `description` với các quy tắc định dạng và giới hạn ký tự nghiêm ngặt.

Cấu trúc thư mục: Claude điều hướng thư mục kỹ năng như một hệ thống tệp. Các tệp tham chiếu, dữ liệu hoặc tài liệu lớn không tiêu tốn token ngữ cảnh cho đến khi chúng thực sự được đọc. Điều này cho phép tổ chức kỹ năng theo cấu trúc thư mục logic, ví dụ:

```
bigquery-skill/
├── SKILL.md (overview, points to reference files)
└── reference/
    ├── finance.md (revenue metrics)
    ├── sales.md (pipeline data)
    └── product.md (usage analytics)
```

Khi người dùng hỏi về doanh thu, Claude sẽ đọc SKILL.md, thấy tham chiếu đến `reference/finance.md`, và chỉ đọc tệp đó. Các tệp khác vẫn nằm trên hệ thống tệp mà không tiêu tốn token ngữ cảnh cho đến khi cần [1].

5. Hướng dẫn từng bước: Quy trình Phát triển Skill lặp lại với Claude

Quy trình phát triển kỹ năng hiệu quả nhất liên quan đến việc sử dụng chính Claude để thiết kế và tinh chỉnh kỹ năng. Điều này bao gồm việc luân phiên giữa một phiên bản Claude đóng vai trò chuyên gia (Claude A) và một phiên bản khác đóng vai trò agent thực hiện tác vụ (Claude B) [1].

1. **Hoàn thành một tác vụ mà không cần Skill**: Làm việc với Claude A để giải quyết một vấn đề bằng cách sử dụng prompt thông thường. Trong quá trình này, bạn sẽ cung cấp ngữ cảnh, giải thích các ưu tiên và chia sẻ kiến thức thủ tục. Ghi nhận những thông tin bạn cung cấp lặp đi lặp lại.

2. **Xác định mẫu có thể tái sử dụng**: Sau khi hoàn thành tác vụ, xác định những ngữ cảnh hoặc quy trình mà bạn đã cung cấp có thể hữu ích cho các tác vụ tương tự trong tương lai.

3. **Yêu cầu Claude A tạo Skill**: Yêu cầu Claude A tạo một Skill dựa trên mẫu đã xác định. Claude hiểu định dạng và cấu trúc Skill một cách tự nhiên, do đó nó có thể tạo ra nội dung SKILL.md có cấu trúc phù hợp với frontmatter và nội dung chính.

4. **Kiểm tra tính súc tích và cải thiện kiến trúc thông tin**: Xem xét lại Skill do Claude A tạo để đảm bảo không có giải thích không cần thiết. Yêu cầu Claude A tổ chức lại nội dung hiệu quả hơn, ví dụ, di chuyển các schema bảng vào một tệp tham chiếu riêng biệt.

5. **Kiểm thử trên các tác vụ tương tự với Claude B**: Sử dụng Skill đã tạo với Claude B (một phiên bản mới của Claude đã tải Skill) trên các trường hợp sử dụng liên quan. Quan sát xem Claude B có tìm thấy thông tin chính xác, áp dụng các quy tắc đúng cách và hoàn thành tác vụ thành công hay không.

6. **Lặp lại dựa trên quan sát**: Nếu Claude B gặp khó khăn hoặc bỏ sót điều gì đó, quay lại Claude A với các chi tiết cụ thể. Ví dụ: "Khi Claude sử dụng Skill này, nó đã quên lọc theo ngày cho Q4. Chúng ta có nên thêm một phần về các mẫu lọc ngày không?" [1].

Quá trình này là một chu trình quan sát-tinh chỉnh-kiểm thử liên tục, giúp cải thiện Skill dựa trên hành vi thực tế của agent, chứ không phải dựa trên các giả định.

6. Ví dụ Thực tế (Use Cases)

6.1. Quy trình phân tích tài liệu (không code)

Một Skill có thể hướng dẫn Claude thực hiện phân tích tài liệu phức tạp mà không cần thực thi mã. Ví dụ, một quy trình phân tích báo cáo nghiên cứu có thể được định nghĩa bằng một danh sách kiểm tra (checklist) trong SKILL.md [1]:

```
## Research Report Analysis Workflow

Copy this checklist and check off items as you complete them:

- [ ] Step 1: Read all source documents
- [ ] Step 2: Identify key themes
- [ ] Step 3: Cross-reference claims
- [ ] Step 4: Create structured summary
- [ ] Step 5: Verify citations
```

Sau đó, mỗi bước sẽ được giải thích chi tiết hơn trong SKILL.md hoặc các tệp tham chiếu khác. Claude sẽ tự động theo dõi tiến độ bằng cách đánh dấu các mục đã hoàn thành trong danh sách kiểm tra.

6.2. Quy trình điền form PDF (có code)

Đối với các tác vụ yêu cầu tương tác với hệ thống bên ngoài hoặc xử lý dữ liệu phức tạp, Skill có thể kết hợp hướng dẫn với các script thực thi. Ví dụ, một quy trình điền form PDF có thể bao gồm các bước và script sau [1]:

```
## PDF Form Filling Workflow

Copy this checklist and check off items as you complete them:

- [ ] Step 1: Analyze the form (run analyze_form.py)
- [ ] Step 2: Create field mapping (edit fields.json)
- [ ] Step 3: Validate mapping (run validate_fields.py)
- [ ] Step 4: Fill the form (run fill_form.py)
- [ ] Step 5: Verify output (run verify_output.py)
```

Trong đó, mỗi script (`analyze_form.py`, `validate_fields.py`, `fill_form.py`, `verify_output.py`) sẽ thực hiện một phần cụ thể của quy trình, và Claude sẽ được hướng dẫn để chạy chúng theo đúng trình tự. Điều này giúp đảm bảo tính chính xác và nhất quán của quá trình.

7. Các Lỗi Thường Gặp và Cách Tránh (Common Pitfalls and How to Avoid Them)

Để xây dựng SKILL.md hiệu quả, cần tránh một số lỗi phổ biến [1]:

- **Tránh đường dẫn kiểu Windows**: Luôn sử dụng dấu gạch chéo tiến (forward slashes) trong các đường dẫn tệp (ví dụ: `scripts/helper.py`), ngay cả trên hệ thống Windows. Đường dẫn kiểu Unix tương thích trên mọi nền tảng, trong khi đường dẫn kiểu Windows (`scripts\\helper.py`) có thể gây lỗi trên các hệ thống Unix.

- **Tránh đưa ra quá nhiều lựa chọn**: Không nên trình bày nhiều cách tiếp cận trừ khi thực sự cần thiết. Thay vào đó, hãy cung cấp một phương pháp mặc định rõ ràng và chỉ đề xuất các lựa chọn thay thế khi có trường hợp ngoại lệ cụ thể. Ví dụ: "Sử dụng `pdfplumber` để trích xuất văn bản. Đối với các PDF được quét cần OCR, hãy sử dụng `pdf2image` với `pytesseract` thay thế."
- **Xử lý lỗi rõ ràng trong script thay vì để Claude tự xử lý**: Khi viết script cho Skill, hãy xử lý các điều kiện lỗi một cách rõ ràng trong script thay vì để Claude tự tìm cách giải quyết. Ví dụ, nếu một tệp không tìm thấy, script nên tạo tệp mặc định thay vì chỉ báo lỗi và dừng lại.
- **Tránh các "hằng số ma thuật" (magic numbers) trong code**: Mọi tham số cấu hình trong script nên được giải thích và ghi lại rõ ràng. Tránh sử dụng các giá trị số không có ý nghĩa rõ ràng (ví dụ: `TIMEOUT = 47`). Thay vào đó, hãy giải thích lý do chọn giá trị đó (ví dụ: `REQUEST_TIMEOUT = 30 # HTTP requests typically complete within 30 seconds`).
- **Tránh thông tin nhạy cảm về thời gian**: Không bao gồm thông tin sẽ trở nên lỗi thời. Nếu cần đề cập đến các phương pháp cũ, hãy đặt chúng trong một phần riêng biệt (ví dụ: `<details><summary>Legacy v1 API (deprecated 2025-08)</summary>...</details>`) để cung cấp ngữ cảnh lịch sử mà không làm lộn xộn nội dung chính.
- **Sử dụng thuật ngữ nhất quán**: Chọn một thuật ngữ duy nhất cho mỗi khái niệm và sử dụng nó xuyên suốt Skill. Sự nhất quán giúp Claude hiểu và tuân theo hướng dẫn tốt hơn (ví dụ: luôn dùng "API endpoint" thay vì xen kẽ "URL", "API route").

8. Kỹ thuật Nâng cao

8.1. Cung cấp các script tiện ích (utility scripts)

Ngay cả khi Claude có thể tự viết script, việc cung cấp các script tiện ích được tạo sẵn mang lại nhiều lợi ích: đáng tin cậy hơn, tiết kiệm token (không cần đưa mã vào ngữ cảnh), tiết kiệm thời gian (không cần tạo mã) và đảm bảo tính nhất quán. Cần làm rõ liệu Claude nên thực thi script hay chỉ đọc nó làm tài liệu tham khảo [1].

8.2. Sử dụng phân tích hình ảnh (visual analysis)

Khi các đầu vào có thể được hiển thị dưới dạng hình ảnh, hãy tận dụng khả năng thị giác của Claude để phân tích chúng. Ví dụ, chuyển đổi PDF sang hình ảnh và yêu cầu Claude phân tích bố cục để xác định các trường biểu mẫu. Điều này giúp Claude hiểu cấu trúc và bố cục một cách trực quan [1].

8.3. Tạo đầu ra trung gian có thể kiểm chứng (verifiable intermediate outputs)

Đối với các tác vụ phức tạp, Claude có thể mắc lỗi. Mô hình "lập kế hoạch-kiểm chứng-thực thi" (plan-validate-execute) giúp phát hiện lỗi sớm bằng cách yêu cầu Claude tạo một kế hoạch có cấu trúc, sau đó kiểm chứng kế hoạch đó bằng script trước khi thực thi. Điều này đặc biệt hữu ích cho các hoạt động hàng loạt, thay đổi mang tính phá hủy hoặc các quy tắc kiểm chứng phức tạp [1].

8.4. Quản lý phụ thuộc gói (package dependencies)

Cần liệt kê các gói cần thiết trong SKILL.md và xác minh chúng có sẵn trong tài liệu công cụ thực thi mã. Môi trường `claude.ai` có thể cài đặt gói từ npm và PyPI, trong khi Claude API không có quyền truy cập mạng hoặc cài đặt gói thời gian chạy [1].

8.5. Tham chiếu công cụ MCP (MCP tool references)

Nếu Skill sử dụng các công cụ Model Context Protocol (MCP), luôn sử dụng tên công cụ đủ điều kiện (ví dụ: `ServerName:tool_name`) để tránh lỗi "tool not found". Điều này đảm bảo Claude có thể định vị công cụ chính xác, đặc biệt khi có nhiều máy chủ MCP [1].

9. Checklist cho SKILL.md hiệu quả

Trước khi chia sẻ một Skill, hãy xác minh các điểm sau [1]:

Chất lượng cốt lõi:

- Mô tả cụ thể và bao gồm các từ khóa chính.
- Mô tả bao gồm cả những gì Skill làm và khi nào nên sử dụng.

- Nội dung SKILL.md dưới 500 dòng.
- Các chi tiết bổ sung nằm trong các tệp riêng biệt (nếu cần).
- Không có thông tin nhạy cảm về thời gian (hoặc nằm trong phần “old patterns”).
- Sử dụng thuật ngữ nhất quán xuyên suốt.
- Các ví dụ cụ thể, không trừu tượng.
- Các tham chiếu tệp chỉ sâu một cấp.
- Sử dụng tiết lộ lũy tiến một cách thích hợp.
- Các quy trình làm việc có các bước rõ ràng.

Code và script:

- Script giải quyết vấn đề thay vì để Claude tự xử lý.
- Xử lý lỗi rõ ràng và hữu ích.
- Không có “hằng số ma thuật” (tất cả các giá trị đều được giải thích).
- Các gói cần thiết được liệt kê trong hướng dẫn và được xác minh là có sẵn.
- Script có tài liệu rõ ràng.
- Không có đường dẫn kiểu Windows (tất cả đều là dấu gạch chéo tiến).
- Các bước kiểm chứng/xác minh cho các hoạt động quan trọng.
- Các vòng lặp phản hồi được bao gồm cho các tác vụ quan trọng về chất lượng.

Kiểm thử:

- Ít nhất ba đánh giá đã được tạo.
- Đã kiểm thử với Haiku, Sonnet và Opus.
- Đã kiểm thử với các kịch bản sử dụng thực tế.
- Phản hồi của nhóm đã được tích hợp (nếu có).

10. Kết luận

Việc thiết kế và viết SKILL.md hiệu quả là yếu tố then chốt để tối ưu hóa khả năng của AI Agents trên nền tảng Claude/Anthropic. Bằng cách tuân thủ các nguyên tắc về tính súc tích, mức độ tự do phù hợp, kiểm thử đa mô hình, đặt tên và mô tả rõ ràng, cùng với việc áp dụng mô hình tiết lộ lũy tiến, chúng ta có thể tạo ra các kỹ năng mạnh mẽ và đáng tin cậy. Việc hiểu rõ cấu trúc, quy trình phát triển lặp lại với Claude, và tránh các lỗi phổ biến sẽ giúp các nhà phát triển xây dựng các agent có khả năng thích ứng và thực hiện tác vụ một cách xuất sắc. Liên tục đánh giá và tinh chỉnh kỹ năng dựa trên hành vi thực tế của agent là chìa khóa để duy trì và nâng cao hiệu suất lâu dài.

References

[1] Skill authoring best practices - Claude API Docs. (n.d.). Retrieved from https://platform.claude.com/docs/en/agents-and-tools/agent-skills/best-practices

> ***Khái niệm then chốt:*** *SKILL.md, Agent Skills, YAML Frontmatter, Context Window, Progressive Disclosure, Degrees of Freedom, Iterative Skill Development, Common Pitfalls*

Chương 11: Xây dựng hệ thống Multi-Agent thực tế

1. Giới thiệu

Hệ thống Multi-Agent (MAS) là tập hợp các tác nhân trí tuệ nhân tạo (AI agents) hoạt động cùng nhau để đạt mục tiêu chung hoặc giải quyết vấn đề phức tạp. Với sự phát triển của các mô hình ngôn ngữ lớn (LLM) như Claude của Anthropic, MAS đang trở thành hướng đi hứa hẹn để giải quyết các tác vụ phức tạp mà một agent đơn lẻ khó thực hiện hiệu quả. Bài nghiên cứu này tập trung vào phối hợp multi-agent, giao thức giao tiếp, quản lý trạng thái chia sẻ và các mẫu triển khai thực tế trong hệ sinh thái Claude/Anthropic.

2. Định nghĩa các khái niệm chính

- **Hệ thống Multi-Agent (MAS):** Tập hợp các tác nhân AI tự chủ, tương tác trong môi trường chia sẻ để đạt mục tiêu chung [1].
- **Phối hợp Multi-Agent (Multi-Agent Coordination):** Cơ chế cho phép các agent cộng tác, chia sẻ thông tin và điều phối hành động hiệu quả [1].
- **Giao thức giao tiếp (Communication Protocols):** Quy tắc chuẩn hóa để các agent trao đổi thông tin, yêu cầu dịch vụ và phản hồi [2].
- **Quản lý trạng thái chia sẻ (Shared State Management):** Quá trình quản lý thông tin chung mà nhiều agent cần truy cập và cập nhật, đảm bảo tính nhất quán và đồng bộ [2].
- **Mẫu triển khai thực tế (Real-world Deployment Patterns):** Kiến trúc và phương pháp đã được chứng minh để triển khai MAS trong ứng dụng thực tế.

3. Best Practices trong xây dựng hệ thống Multi-Agent

3.1. Kiến trúc Orchestrator-Worker

Anthropic áp dụng kiến trúc này, sử dụng một **Agent chính (Orchestrator)** phân tích truy vấn, phát triển chiến lược và ủy quyền tác vụ cho các **Subagent (Worker Agents)** chuyên biệt. Các Subagent hoạt động song song, thực hiện tác vụ và trả về kết quả cho Agent chính để tổng hợp, giúp phân chia công việc và quản lý độ phức tạp [1].

3.2. Tối ưu hóa Prompt Engineering

Thiết kế prompt hiệu quả là rất quan trọng cho từng vai trò agent [1]:

- **Dạy Orchestrator ủy quyền:** Hướng dẫn chi tiết cho agent chính về cách phân tách truy vấn, mô tả mục tiêu, định dạng đầu ra, công cụ và nguồn cho từng subagent, tránh trùng lặp công việc.
- **Điều chỉnh nỗ lực theo độ phức tạp:** Nhúng quy tắc mở rộng quy mô vào prompt để agent đánh giá mức độ nỗ lực phù hợp.

- **Hướng dẫn quy trình tư duy:** Sử dụng "chế độ tư duy mở rộng" (extended thinking mode) để Claude xuất thêm token trong quy trình tư duy hiển thị, giúp agent chính lập kế hoạch, đánh giá công cụ và định nghĩa vai trò subagent, cải thiện lý luận và hiệu quả [1].

3.3. Thiết kế và lựa chọn công cụ hiệu quả

Cung cấp cho agent các heuristic rõ ràng để sử dụng đúng công cụ, ví dụ: kiểm tra tất cả công cụ có sẵn, khớp việc sử dụng công cụ với ý định người dùng, ưu tiên công cụ chuyên biệt. Mô tả công cụ phải rõ ràng và có mục đích riêng biệt [1].

3.4. Song song hóa các lượt gọi công cụ và Subagent

Để tăng tốc độ nghiên cứu, áp dụng hai loại song song hóa [1]:

- **Song song hóa Subagent:** Agent chính tạo 3-5 subagent hoạt động song song.
- **Song song hóa công cụ:** Các subagent sử dụng 3+ công cụ song song.

3.5. Quản lý trạng thái chia sẻ

Sử dụng Memory để duy trì ngữ cảnh cho agent chính [1]. Các giao thức giao tiếp như **ACP (Agent Communication Protocol)** của IBM chuẩn hóa cách agent giao tiếp, ủy quyền tác vụ và điều phối quy trình làm việc, duy trì ngữ cảnh và cung cấp công cụ giám sát [2].

4. Step-by-step Guide: Xây dựng hệ thống nghiên cứu Multi-Agent (theo Anthropic)

Anthropic đã chia sẻ quy trình làm việc của hệ thống nghiên cứu multi-agent của họ [1]:

1. **Tiếp nhận truy vấn:** Người dùng gửi truy vấn.
2. **Phân tích và lập kế hoạch:** Agent chính (LeadResearcher) phân tích, lập chiến lược và lưu vào Memory.
3. **Tạo Subagent chuyên biệt:** Agent chính tạo các Subagent với nhiệm vụ cụ thể.

4. **Thực hiện nghiên cứu song song:** Mỗi Subagent độc lập tìm kiếm, đánh giá kết quả và trả về cho Agent chính.
5. **Tổng hợp và quyết định:** Agent chính tổng hợp kết quả, quyết định nghiên cứu thêm hoặc tinh chỉnh chiến lược.
6. **Xử lý trích dẫn:** CitationAgent xử lý tài liệu và báo cáo để xác định vị trí trích dẫn.
7. **Trả về kết quả:** Kết quả nghiên cứu cuối cùng, hoàn chỉnh với trích dẫn, được trả về cho người dùng [1].

5. Ví dụ thực tế (Use Cases)

5.1. Hệ thống quản lý năng lượng nhà thông minh

MAS tối ưu hóa sử dụng năng lượng dựa trên giá điện, dự báo thời tiết và sở thích người dùng [2]. Bao gồm các agent như User Assistant, Grid Connection (MCP), Weather Forecasting (MCP), Device Control (MCP), Energy Optimization (ACP, A2A), và Smart Device Agents.

5.2. Hệ thống xử lý tài liệu doanh nghiệp

MAS tự động hóa quy trình xử lý tài liệu từ tiếp nhận đến trích xuất và lưu trữ dữ liệu [2]. Bao gồm các agent như Ingestion, OCR (MCP), Extraction (A2A), Validation, Storage (MCP), và Orchestration (ACP).

6. Các lỗi thường gặp và cách tránh

- **Phát sinh quá nhiều Subagent:** Lãng phí tài nguyên. **Cách tránh:** Nhúng quy tắc mở rộng quy mô vào prompt để giới hạn số lượng subagent dựa trên độ phức tạp của truy vấn [1].
- **Truy vấn quá rộng/hẹp:** Tìm kiếm không hiệu quả. **Cách tránh:** Hướng dẫn agent bắt đầu với truy vấn ngắn, rộng, sau đó thu hẹp trọng tâm [1].
- **Thiếu phối hợp:** Trùng lặp công việc. **Cách tránh:** Dạy agent chính ủy quyền hiệu quả, cung cấp mô tả nhiệm vụ chi tiết và sử dụng giao thức giao tiếp rõ ràng (A2A, ACP) [1, 2].

- **Tiêu tốn token nhanh:** Chi phí cao. **Cách tránh:** Chỉ sử dụng MAS cho tác vụ giá trị cao; tối ưu hóa prompt và thiết kế công cụ để giảm thiểu token không cần thiết [1].

7. Kỹ thuật nâng cao

7.1. Agent tự cải thiện

Các mô hình Claude 4 có thể chẩn đoán lỗi và đề xuất cải tiến prompt. Anthropic đã tạo agent kiểm tra công cụ, giúp giảm 40% thời gian hoàn thành nhiệm vụ bằng cách viết lại mô tả công cụ để tránh lỗi [1].

7.2. Chế độ tư duy mở rộng

Chế độ này cho phép Claude xuất thêm token trong quy trình tư duy hiển thị, hoạt động như bản nháp kiểm soát. Agent chính sử dụng để lập kế hoạch, đánh giá công cụ, xác định độ phức tạp truy vấn và số lượng subagent, cải thiện lý luận và hiệu quả [1].

8. Kết luận

Xây dựng hệ thống multi-agent thực tế là nỗ lực phức tạp nhưng mang lại nhiều lợi ích, đặc biệt cho các tác vụ đòi hỏi sự song song hóa cao và xử lý thông tin vượt quá giới hạn cửa sổ ngữ cảnh đơn lẻ. Bằng cách áp dụng các best practices về phối hợp, giao thức giao tiếp (MCP, A2A, ACP), quản lý trạng thái chia sẻ và các kỹ thuật nâng cao, các nhà phát triển có thể tạo ra các AI agents mạnh mẽ và hiệu quả hơn trên hệ sinh thái Claude/Anthropic.

9. Tài liệu tham khảo

[1] Anthropic. (2025, June 13). *How we built our multi-agent research system*. https://www.anthropic.com/engineering/multi-agent-research-system [2] Gupta, M. (2025, May 5). *Agentic AI Protocols: MCP, A2A, and ACP*. Medium. https://medium.com/@manavg/agentic-ai-protocols-mcp-a2a-and-acp-ea0200eac18b

> ***Khái niệm then chốt:*** *Hệ thống Multi-Agent (MAS), Phối hợp Multi-Agent, Giao thức giao tiếp, Quản lý trạng thái chia sẻ, Mẫu triển khai thực tế, Kiến trúc Orchestrator-Worker, Prompt Engineering, Chế độ tư duy mở rộng*

Chương 12: An toàn, Giám sát & Tối ưu hóa Agents trong môi trường Production

Giới thiệu

Triển khai AI Agents trên Claude/Anthropic trong môi trường sản xuất đòi hỏi an toàn, giám sát, tối ưu hóa. Báo cáo này khám phá các nguyên tắc, kỹ thuật đảm bảo agent hiệu quả, đáng tin cậy, bảo mật.

Định nghĩa các thuật ngữ chính

Khái niệm cốt lõi quản lý AI Agents trong sản xuất:

- **Human-in-the-loop (HITL):** Hệ thống AI tích hợp con người giám sát, điều chỉnh, phê duyệt hành động agent trong tình huống rủi ro cao [1].
- **Minimal Footprint Principle (Nguyên tắc Dấu chân Tối thiểu):** Vận hành AI Agents hiệu quả tài nguyên (token, chi phí, bộ nhớ), giảm chi phí, tăng tốc độ, giảm tác động môi trường. Tối ưu hóa ngữ cảnh, lời gọi API trong LLM [3].
- **Error Handling (Xử lý lỗi):** Xác định, phản ứng, phục hồi lỗi/ngoại lệ, giúp agent ổn định, đáng tin cậy, tự phục hồi.
- **Monitoring (Giám sát):** Theo dõi hiệu suất, hành vi, trạng thái agent theo thời gian thực; thu thập chỉ số, nhật ký, cảnh báo để phát hiện vấn đề, đánh giá hiệu quả, đảm bảo an toàn.
- **Performance Optimization (Tối ưu hóa hiệu suất):** Kỹ thuật cải thiện tốc độ, hiệu quả, chất lượng đầu ra của AI Agents: tối ưu hóa prompt, quản lý ngữ cảnh, lựa chọn mô hình, kiến trúc agent phức tạp (chuỗi prompt, song song hóa) [2].

Best Practices trong phát triển và vận hành AI Agents

Nguyên tắc phát triển agent an toàn, đáng tin cậy của Anthropic [1]:

1. **Giữ con người kiểm soát trong khi vẫn cho phép agent tự chủ (Keeping humans in control while enabling agent autonomy):** Cân bằng tự chủ agent với giám sát con người, phê duyệt quyết định rủi ro cao (ví dụ: Claude Code yêu cầu chấp thuận sửa đổi mã) [1].

2. **Minh bạch trong hành vi của agent (Transparency in agent behavior):** Agent giải thích logic để con người hiểu, đánh giá. Claude Code hiển thị danh sách kiểm tra hành động theo thời gian thực, cho phép can thiệp [1].

3. **Căn chỉnh agent với giá trị và kỳ vọng của con người (Aligning agents with human values and expectations):** Đảm bảo agent phù hợp giá trị con người, tránh đi ngược lợi ích người dùng. Minh bạch, kiểm soát then chốt [1].

4. **Bảo vệ quyền riêng tư trong các tương tác mở rộng (Protecting privacy across extended interactions):** Bảo vệ quyền riêng tư khi agent lưu giữ thông tin. Thiết kế công cụ/quy trình bảo vệ phù hợp; MCP của Anthropic kiểm soát truy cập [1].

5. **Bảo mật tương tác của agent (Securing agents' interactions):** Thiết kế hệ thống agent bảo vệ dữ liệu nhạy cảm, ngăn chặn lạm dụng. Claude dùng phân loại, bảo mật chống prompt injection; Anthropic cung cấp hướng dẫn [1].

6. **Kỹ thuật ngữ cảnh hiệu quả (Effective Context Engineering):** Tối ưu hóa tiện ích token trong ngữ cảnh hữu hạn LLM: tinh chỉnh system prompts, công cụ hiệu quả, quản lý lịch sử tin nhắn để giữ ngữ cảnh nhỏ gọn, tín hiệu cao [3].

7. **Đơn giản hóa thiết kế agent (Simplicity in agent design):** Ưu tiên thiết kế agent đơn giản, dùng mẫu có thể kết hợp. Bắt đầu giải pháp đơn giản, tăng phức tạp khi cần; nhiều ứng dụng chỉ cần một lời gọi LLM [2].

Step-by-step Guide: Xây dựng Agent hiệu quả với Claude

Để xây dựng một AI Agent hiệu quả và đáng tin cậy trên hệ sinh thái Claude, có thể tuân theo các bước sau, kết hợp các nguyên tắc đã nêu:

1. **Xác định mục tiêu và phạm vi:** Định nghĩa rõ mục tiêu, tác vụ, giới hạn quyền tự chủ [2].
2. **Ưu tiên thiết kế đơn giản:** Bắt đầu với một lời gọi LLM duy nhất, dùng mẫu có thể kết hợp để dễ debug, mở rộng [2].
3. **Kỹ thuật Ngữ cảnh hiệu quả:**
    - **System Prompts rõ ràng:** Viết system prompts rõ ràng, trực tiếp, cấu trúc bằng thẻ XML/Markdown, cung cấp thông tin tối thiểu, đầy đủ [3].
    - **Công cụ hiệu quả:** Thiết kế công cụ hiệu quả token, khuyến khích hành vi agent hiệu quả, giao diện rõ ràng cho LLM [3].
    - **Quản lý lịch sử tin nhắn:** Tinh chỉnh, nén lịch sử tin nhắn để tránh Context Rot [3].
4. **Tích hợp Human-in-the-loop (HITL):** Xác định điểm HITL, thiết kế cơ chế giám sát, dừng, điều chỉnh, phê duyệt hành động (ví dụ: quyền chỉ đọc mặc định, yêu cầu phê duyệt sửa đổi) [1].
5. **Giám sát và Xử lý lỗi:**
    - **Giám sát:** Thu thập chỉ số hiệu suất, độ trễ, chi phí token, hành vi bất thường. Dùng công cụ giám sát, nhật ký, cảnh báo (ví dụ: theo dõi danh sách kiểm tra Claude Code) [1].
    - **Xử lý lỗi:** Xây dựng cơ chế xử lý lỗi mạnh mẽ: thử lại tác vụ, thông báo lỗi không tự khắc phục, chuyển giao quyền kiểm soát.
6. **Tối ưu hóa liên tục:** Dựa trên giám sát và phản hồi, tinh chỉnh prompt, công cụ, logic agent để cải thiện hiệu suất, giảm chi phí, tăng an toàn. Dùng evals đo lường hiệu quả [1, 3].

Ví dụ thực tế (Use Cases)

1. **Claude Code - Agent lập trình tự động [1]:**
    - **Mô tả:** Claude Code (Anthropic) tự động viết, debug, chỉnh sửa mã.
    - **An toàn & Giám sát:** Mặc định chỉ đọc, yêu cầu chấp thuận sửa đổi mã/hệ thống. Cấp quyền lâu dài, can thiệp qua danh sách kiểm tra hành động [1].

- **Tối ưu hóa:** Tối ưu chi phí qua bộ nhớ đệm prompt, nén ngữ cảnh [4].

2. **Agent phân tích dữ liệu tài chính tại Block [1]:**
    - **Mô tả:** Agent Block (dịch vụ tài chính) giúp nhân viên phi kỹ thuật truy cập dữ liệu, tiết kiệm thời gian kỹ sư.
    - **An toàn & Giám sát:** HITL quan trọng: agent truy vấn, tạo báo cáo; quyết định tài chính/thay đổi hệ thống cần con người phê duyệt. Giám sát chặt chẽ để đảm bảo chính xác, tuân thủ.
    - **Tối ưu hóa:** Tối ưu hóa bằng routing: chuyển truy vấn phức tạp/nhạy cảm đến mô hình mạnh hơn/HITL; truy vấn đơn giản xử lý bởi mô hình nhỏ, tiết kiệm chi phí [2].

Các lỗi thường gặp và cách tránh

1. **Prompt Injection:** Kẻ tấn công chèn hướng dẫn độc hại vào prompt, thao túng agent bỏ qua chỉ dẫn, tiết lộ thông tin trái phép, hoặc thực hiện hành động không mong muốn.
    - **Cách tránh:** Claude dùng phân loại chống prompt injection. Thiết kế công cụ, quyền hạn agent cẩn thận, áp dụng đặc quyền tối thiểu, giám sát hành vi bất thường [1].
2. **Context Rot (Suy giảm ngữ cảnh):** Độ dài ngữ cảnh tăng làm giảm khả năng truy xuất thông tin, khiến agent bỏ sót hoặc quyết định kém chính xác.
    - **Cách tránh:** Áp dụng kỹ thuật ngữ cảnh hiệu quả: tinh chỉnh system prompts, dùng công cụ truy xuất chọn lọc, nén/tóm tắt lịch sử tin nhắn để giữ ngữ cảnh nhỏ gọn [3].
3. **Agent Hallucinations (Ảo giác của Agent):** Agent tạo thông tin sai lệch/không có thật khi thiếu thông tin hoặc suy luận quá xa dữ liệu.
    - **Cách tránh:** Tăng cường minh bạch agent để con người kiểm tra logic/nguồn. Dùng RAG cung cấp thông tin đáng tin cậy. Tích hợp HITL xác minh đầu ra.
4. **Over-automation (Tự động hóa quá mức):** Triển khai agent tự chủ quá mức, thiếu giám sát/kiểm soát con người, dẫn đến hành động không mong muốn/gây

hại.

    - **Cách tránh:** Bắt đầu quyền tự chủ hạn chế, tăng dần khi agent đáng tin cậy. Áp dụng HITL cho quyết định rủi ro cao. Đảm bảo có kill switch/cơ chế can thiệp khẩn cấp [1].

Kỹ thuật nâng cao

1. **Multi-agent Orchestration (Điều phối đa Agent):** Chia tác vụ phức tạp cho nhiều agent chuyên biệt, phối hợp cải thiện hiệu suất, mở rộng, chịu lỗi.
    - **Ví dụ:** Ví dụ: agent chính điều phối agent con tìm kiếm, phân tích, tạo báo cáo [2].
2. **Adaptive Context Management (Quản lý ngữ cảnh thích ứng):** Điều chỉnh ngữ cảnh động cho LLM dựa trên tác vụ, mức độ quan trọng, ràng buộc token (ưu tiên thông tin gần đây, tóm tắt, nén).
3. **Automated Evals (Đánh giá tự động):** Phát triển hệ thống đánh giá tự động (evals) để kiểm tra liên tục hiệu suất, an toàn, căn chỉnh agent trong sandbox/dữ liệu ẩn danh, phát hiện sớm vấn đề và cải tiến [1].
4. **Model Context Protocol (MCP):** Giao thức mã nguồn mở của Anthropic cho phép Claude kết nối dịch vụ khác, kiểm soát truy cập công cụ/quy trình, cấp quyền một lần/vĩnh viễn, bảo vệ quyền riêng tư và bảo mật [1].

Kết luận

Xây dựng và vận hành AI Agents trên Claude/Anthropic trong production đòi hỏi tiếp cận toàn diện: tích hợp HITL, minimal footprint, xử lý lỗi, giám sát, tối ưu hóa hiệu suất và kỹ thuật ngữ cảnh. Điều này tạo ra AI Agents mạnh mẽ, hiệu quả, đáng tin cậy và an toàn. Các công cụ như MCP và Automated Evals hỗ trợ xây dựng hệ sinh thái agent bền vững, có trách nhiệm.

Tài liệu tham khảo

[1] Anthropic. (2025, August 4). *Our framework for developing safe and trustworthy agents*. https://www.anthropic.com/news/our-framework-for-developing-safe-and-trustworthy-agents [2] Anthropic. (2024, December 19). *Building Effective AI Agents*. https://www.anthropic.com/research/building-effective-agents [3] Anthropic. (2025, September 29). *Effective context engineering for AI agents*. https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents [4] Medium. (n.d.). *10 Claude AI Agents That Reduced My API Costs*. https://medium.com/@bhagyarana80/10-claude-ai-agents-that-reduced-my-api-costs-0b4e125da29e

> ***Khái niệm then chốt:*** *Human-in-the-loop (HITL), Minimal Footprint Principle, Error Handling, Monitoring, Performance Optimization, Prompt Injection, Context Rot, Automated Evals*
