# Chứng chỉ Claude Certified Architect — Foundations

## Hướng dẫn ôn tập (Dựa trên Hướng dẫn thi chính thức)

## Giới thiệu

Chứng chỉ **Claude Certified Architect — Foundations** xác nhận rằng một chuyên gia có thể đưa ra các quyết định trade-off hợp lý khi triển khai các giải pháp dựa trên Claude trong thực tế. Bài thi đánh giá kiến thức nền tảng về Claude Code, Claude Agent SDK, Claude API, và Model Context Protocol (MCP)—những công nghệ cốt lõi để xây dựng các ứng dụng production với Claude.

Các câu hỏi trong bài thi dựa trên những kịch bản thực tế của ngành: xây dựng các hệ thống agentic cho hỗ trợ khách hàng, thiết kế các pipeline nghiên cứu multi-agent, tích hợp Claude Code vào CI/CD, tạo các công cụ nâng cao năng suất cho lập trình viên, và trích xuất dữ liệu có cấu trúc từ các tài liệu phi cấu trúc.

## Đối tượng ứng viên

Ứng viên lý tưởng là một **solution architect** thiết kế và phát hành các ứng dụng production với Claude. Bạn nên có ít nhất 6 tháng kinh nghiệm thực hành với:

- **Claude Agent SDK** — điều phối multi-agent, ủy thác cho subagent, tích hợp tool, lifecycle hook
- **Claude Code** — CLAUDE.md, MCP server, Agent Skills, planning mode
- **Model Context Protocol (MCP)** — tool và resource để tích hợp backend
- **Prompt engineering** — JSON schema, ví dụ few-shot, template trích xuất dữ liệu
- **Context window** — làm việc với tài liệu dài, truyền context giữa nhiều agent
- **CI/CD pipeline** — review code tự động, tạo test
- **Escalation và độ tin cậy** — xử lý lỗi, human-in-the-loop

## Định dạng bài thi

| Tham số | Giá trị |
|---|---|
| Loại câu hỏi | Trắc nghiệm (1 đáp án đúng trong 4) |
| Cách chấm điểm | Thang 100–1000, điểm đạt **720** |
| Phạt khi đoán | Không (hãy trả lời mọi câu hỏi!) |
| Số kịch bản | 4 trong số 8 kịch bản có thể (chọn ngẫu nhiên) |

# Nội dung bài thi: 5 Lĩnh vực

| Lĩnh vực | Trọng số |
|---|---|
| 1. Kiến trúc agent và điều phối | **27%** |
| 2. Thiết kế tool và tích hợp MCP | **18%** |
| 3. Cấu hình và quy trình làm việc Claude Code | **20%** |
| 4. Prompt engineering và structured output | **20%** |
| 5. Quản lý context và độ tin cậy | **15%** |

---

# Các kịch bản bài thi

## Kịch bản 1: Agent Hỗ trợ Khách hàng

Bạn xây dựng một agent để xử lý trả hàng, tranh chấp thanh toán, và các vấn đề tài khoản bằng Claude Agent SDK. Agent sử dụng các MCP tool (`get_customer`, `lookup_order`, `process_refund`, `escalate_to_human`). Mục tiêu là đạt 80%+ tỷ lệ giải quyết ngay lần liên hệ đầu tiên với escalation phù hợp.

## Kịch bản 2: Sinh mã với Claude Code

Bạn dùng Claude Code để tăng tốc phát triển: sinh mã, refactor, debug, viết tài liệu. Bạn cần tích hợp nó với các custom slash command và cấu hình CLAUDE.md, đồng thời hiểu khi nào nên dùng planning mode.

## Kịch bản 3: Hệ thống Nghiên cứu Multi-Agent

Một coordinator agent ủy thác các tác vụ cho những subagent chuyên biệt: nghiên cứu web, phân tích tài liệu, tổng hợp, và tạo báo cáo. Hệ thống phải tạo ra các báo cáo hoàn chỉnh kèm trích dẫn.

## Kịch bản 4: Công cụ Nâng cao Năng suất Lập trình viên

Agent giúp các kỹ sư khám phá những codebase chưa quen thuộc, sinh boilerplate code, và tự động hóa các tác vụ thường nhật. Các tool dựng sẵn (Read, Write, Bash, Grep, Glob) và MCP server được sử dụng.

## Kịch bản 5: Claude Code cho Continuous Integration

Tích hợp Claude Code vào một CI/CD pipeline để review code tự động, tạo test, và phản hồi trên pull request. Prompt phải được thiết kế để giảm thiểu false positive.

## Kịch bản 6: Trích xuất Dữ liệu Có cấu trúc

Hệ thống trích xuất thông tin từ các tài liệu phi cấu trúc, validate output bằng JSON schema, và duy trì độ chính xác cao. Nó phải xử lý đúng các edge case.

### Kịch bản 7: Các Mẫu Kiến trúc AI Hội thoại

Bạn thiết kế các hệ thống hội thoại multi-turn bao gồm quản lý context window, duy trì chỉ dẫn xuyên suốt các lượt, các chiến lược memory, thiết kế tool để thực thi an toàn, và xử lý các đầu vào người dùng mơ hồ hoặc mâu thuẫn.

### Kịch bản 8: Công cụ Agentic AI *(thiếu nội dung — hãy giúp chúng tôi bổ sung!)*

Kịch bản này đã được các ứng viên thi báo cáo nhưng chưa được đề cập trong hướng dẫn này. Nếu bạn từng gặp các câu hỏi từ kịch bản này trong bài thi thật, hãy chia sẻ chúng tại GitHub Issues để chúng tôi có thể bổ sung đầy đủ. Đóng góp của bạn sẽ giúp ích cho mọi người đang ôn tập cho bài thi.

---

# Tài liệu chính thức

| Tài nguyên | URL |
|---|---|
| **Claude API — Messages** | https://platform.claude.com/docs/en/api/messages |
| **Claude API — Tool Use** | https://platform.claude.com/docs/en/build-with-claude/tool-use |
| **Claude API — Message Batches** | https://platform.claude.com/docs/en/build-with-claude/message-batches |
| **Claude Agent SDK — Overview** | https://platform.claude.com/docs/en/agent-sdk/overview |
| **Claude Agent SDK — Hooks** | https://platform.claude.com/docs/en/agent-sdk/hooks |
| **Claude Agent SDK — Subagents** | https://platform.claude.com/docs/en/agent-sdk/subagents |
| **Claude Agent SDK — Sessions** | https://platform.claude.com/docs/en/agent-sdk/sessions |
| **Model Context Protocol (MCP)** | https://modelcontextprotocol.io/ |
| **MCP — Tools** | https://modelcontextprotocol.io/docs/concepts/tools |
| **MCP — Resources** | https://modelcontextprotocol.io/docs/concepts/resources |
| **MCP — Servers** | https://modelcontextprotocol.io/docs/concepts/servers |
| **Claude Code — Documentation** | https://code.claude.com/docs/en/overview |
| **Claude Code — CLAUDE.md and Memory** | https://code.claude.com/docs/en/memory |
| **Claude Code — Skills (incl. slash commands)** | https://code.claude.com/docs/en/skills |
| **Claude Code — Hooks** | https://code.claude.com/docs/en/hooks |
| **Claude Code — Sub-agents** | https://code.claude.com/docs/en/sub-agents |
| **Claude Code — MCP Integration** | https://code.claude.com/docs/en/mcp |

| | |
|---|---|
| **Claude Code — GitHub Actions CI/CD** | https://code.claude.com/docs/en/github-actions |
| **Claude Code — GitLab CI/CD** | https://code.claude.com/docs/en/gitlab-ci-cd |
| **Claude Code — Headless (non-interactive mode)** | https://code.claude.com/docs/en/headless |
| **Prompt Engineering Guide** | https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/overview |
| **Extended Thinking** | https://platform.claude.com/docs/en/build-with-claude/extended-thinking |
| **Anthropic Cookbook (code examples)** | https://github.com/anthropics/anthropic-cookbook |

---

# PHẦN I: NỀN TẢNG LÝ THUYẾT

Phần này bao gồm toàn bộ lý thuyết bạn cần để vượt qua bài thi thành công. Tài liệu được tổ chức theo công nghệ và khái niệm thay vì theo các lĩnh vực thi—điều này giúp bạn xây dựng hiểu biết sâu hơn về từng chủ đề.

---

# Chương 1: Claude API — Nền tảng Tương tác với Mô hình

> Tài liệu: Messages API | Prompt Engineering

## 1.1 Cấu trúc Yêu cầu API

Claude API tuân theo mô hình request–response. Mỗi request đến Claude Messages API bao gồm:

```
{
  "model": "claude-sonnet-4-6",
  "max_tokens": 1024,
  "system": "You are a helpful assistant.",
  "messages": [
    {"role": "user", "content": "Hi!"},
    {"role": "assistant", "content": "Hello!"},
    {"role": "user", "content": "How are you?"}
  ],
  "tools": [...],
  "tool_choice": {"type": "auto"}
}
```

**Các trường chính:**

- `model` — lựa chọn mô hình (`claude-opus-4-6`, `claude-sonnet-4-6`, `claude-haiku-4-5`)
- `max_tokens` — số lượng token tối đa trong phản hồi
- `system` — system prompt (định nghĩa hành vi của mô hình)
- `messages` — lịch sử hội thoại (**bạn phải gửi toàn bộ lịch sử** để duy trì tính nhất quán)
- `tools` — định nghĩa của các tool khả dụng
- `tool_choice` — chiến lược lựa chọn tool

## 1.2 Vai trò của Message

Mảng `messages` sử dụng hai vai trò hội thoại cộng với một vai trò chỉ dẫn:

- `user` — tin nhắn của người dùng, bao gồm cả kết quả tool (được gửi dưới dạng content block `tool_result` bên trong một message có vai trò `user`, không phải một vai trò `tool` riêng)
- `assistant` — phản hồi của mô hình (được đưa vào khi gửi lịch sử), bao gồm cả các yêu cầu sử dụng tool (content block `tool_use`)
- `system` — có thể được đặt qua trường `system` cấp cao nhất (áp dụng từ lượt đầu tiên) hoặc chèn trực tiếp trong `messages` dưới dạng `{"role": "system", ...}` (áp dụng từ điểm đó trở đi, tuân theo các quy tắc vị trí — xem bên dưới)

Kết quả tool không được gửi dưới dạng message có vai trò `"tool"`. Chúng được gửi dưới dạng message có vai trò `user` mà nội dung bao gồm một content block `tool_result`:

```
{
  "role": "user",
  "content": [
    {
      "type": "tool_result",
      "tool_use_id": "toolu_01...",
      "content": "..."
    }
  ]
}
```

`system` cũng có thể xuất hiện trực tiếp như một vai trò trong mảng `messages`, không chỉ qua tham số `system` cấp cao nhất. Điều này nhằm thêm hướng dẫn giữa cuộc hội thoại mà không làm mất hiệu lực của tiền tố (prefix) đã được cache từ trường `system` cấp cao nhất. Nó có các quy tắc vị trí cụ thể:

- Phải theo ngay sau một lượt `user` (kể cả lượt có chứa block `tool_result`) hoặc một lượt `assistant` kết thúc bằng việc sử dụng server tool.
- Phải đứng trước một lượt `assistant` hoặc là phần tử cuối cùng của mảng.
- Không thể nằm giữa một block `tool_use` và `tool_result` tương ứng của nó — làm vậy sẽ trả về lỗi 400.
- Các message `system` xuất hiện sau (kể cả những message chèn giữa cuộc hội thoại) sẽ được ưu tiên hơn các message trước đó và hơn trường `system` cấp cao nhất đối với các lượt tiếp theo.

**Cực kỳ quan trọng:** với mỗi API request bạn phải gửi **toàn bộ lịch sử hội thoại**. Mô hình không lưu trữ trạng thái giữa các request—mỗi lần gọi là độc lập.

## 1.3 Trường `stop_reason` trong Phản hồi

Phản hồi của Claude API bao gồm `stop_reason`, cho biết lý do mô hình dừng sinh nội dung:

| Giá trị | Mô tả | Hành động |
|---|---|---|
| `"end_turn"` | Mô hình đã hoàn tất phản hồi | Hiển thị kết quả cho người dùng |
| `"tool_use"` | Mô hình muốn gọi một tool | Thực thi tool và trả về kết quả |
| `"max_tokens"` | Đã đạt giới hạn token | Phản hồi bị cắt cụt; bạn có thể cần tăng giới hạn |
| `"stop_sequence"` | Gặp một stop sequence | Xử lý dựa trên logic ứng dụng của bạn |

Đối với các hệ thống agentic, `"tool_use"` và `"end_turn"` là quan trọng nhất—chúng điều khiển vòng lặp của agent.

## 1.4 System Prompt

System prompt là một chỉ dẫn đặc biệt định nghĩa context và các quy tắc hành vi. Nó:

- Không thuộc mảng `messages`; nó được truyền riêng trong trường `system`
- Có ưu tiên cao hơn các tin nhắn của người dùng
- Được nạp một lần và áp dụng xuyên suốt cuộc hội thoại
- Được dùng để định nghĩa vai trò, ràng buộc, và định dạng output

**Quan trọng cho bài thi:** cách diễn đạt của system prompt có thể tạo ra các liên kết tool ngoài ý muốn. Ví dụ, một chỉ dẫn như "luôn xác minh khách hàng" có thể khiến mô hình lạm dụng `get_customer`, ngay cả khi điều đó không cần thiết.

## 1.5 Context Window

Context window là tổng lượng văn bản (tính bằng token) mà mô hình có thể xử lý cùng một lúc. Nó bao gồm:

- System prompt
- Toàn bộ lịch sử message
- Định nghĩa tool
- Kết quả tool

**Các vấn đề chính của context window:**

1. **Hiệu ứng lost-in-the-middle:** các mô hình xử lý đáng tin cậy thông tin ở đầu và cuối của một đầu vào dài nhưng có thể bỏ sót các chi tiết ở giữa. Cách giảm thiểu: đặt thông tin quan trọng gần đầu hoặc cuối.

2. **Tích lũy kết quả tool:** mỗi lần gọi tool đều thêm output vào context. Nếu một tool trả về 40+ trường nhưng chỉ 5 trường có ý nghĩa, thì phần lớn context bị lãng phí.
3. **Tóm tắt lũy tiến:** khi nén lịch sử, các giá trị số, phần trăm, và ngày tháng thường bị mất và trở nên mơ hồ ("khoảng", "tầm", "một vài").

---

# Chương 2: Tools và `tool_use`

> Tài liệu: Tool Use

## 2.1 `tool_use` là gì

`tool_use` là cơ chế cho phép Claude gọi các hàm bên ngoài. Model không tự chạy code—nó sinh ra một yêu cầu gọi tool có cấu trúc; code của bạn thực thi yêu cầu đó và trả về kết quả.

## 2.2 Định nghĩa Tool

Mỗi tool được định nghĩa bằng một JSON schema:

```
{
  "name": "get_customer",
  "description": "Finds a customer by email or ID. Returns the customer profile,
including name, email, order history, and account status. Use this tool BEFORE
lookup_order to verify the customer's identity. Accepts an email (format:
user@domain.com) or a numeric customer_id.",
  "input_schema": {
    "type": "object",
    "properties": {
      "email": {"type": "string", "description": "Customer email"},
      "customer_id": {"type": "integer", "description": "Numeric customer ID"}
    },
    "required": []
  }
}
```

**Những khía cạnh cực kỳ quan trọng của phần mô tả tool:**

1. **Phần mô tả là cơ chế lựa chọn chính.** Một LLM chọn tool dựa trên phần mô tả của chúng. Mô tả tối giản (“Truy xuất thông tin khách hàng”) dẫn đến nhầm lẫn khi các tool chồng lấn nhau.
2. **Trong phần mô tả nên bao gồm:**
   - Tool làm gì và trả về gì
   - Định dạng đầu vào và giá trị ví dụ
   - Các edge case và ràng buộc
   - Khi nào nên dùng tool này so với các phương án tương tự

3. **Tránh** các phần mô tả giống hệt hoặc chồng lấn giữa các tool. Nếu `analyze_content` và `analyze_document` có phần mô tả gần như giống hệt nhau, model sẽ nhầm lẫn chúng.
4. **Built-in tool so với MCP tool:** các agent có thể ưu tiên built-in tool (Read, Grep) hơn các MCP tool có chức năng tương tự. Để ngăn điều này, hãy tăng cường phần mô tả của MCP tool—nhấn mạnh những ưu thế cụ thể, dữ liệu độc nhất, hoặc context mà built-in tool không thể cung cấp.

## 2.3 Tham số `tool_choice`

`tool_choice` kiểm soát cách model lựa chọn tool:

| Giá trị | Hành vi | Khi nào dùng |
|---|---|---|
| `{"type": "auto"}` | Model tự quyết định có gọi tool hay trả lời bằng văn bản | Mặc định cho hầu hết các trường hợp |
| `{"type": "any"}` | Model **phải** gọi một tool nào đó | Khi bạn cần đảm bảo có structured output |
| `{"type": "tool", "name": "extract_metadata"}` | Model **phải** gọi một tool cụ thể | Khi bạn cần ép một bước đầu tiên / thứ tự thực thi |

**Các kịch bản quan trọng:**

- `tool_choice: "any"` + nhiều tool trích xuất → model chọn tool tốt nhất, nhưng bạn vẫn nhận được structured output
- Lựa chọn bị ép buộc → khi bạn phải đảm bảo một hành động đầu tiên cụ thể (ví dụ, `extract_metadata` trước khi làm giàu dữ liệu)

## 2.4 JSON Schema cho Structured Output

Dùng `tool_use` kèm JSON schema là cách **đáng tin cậy nhất** để thu được structured output từ Claude. Nó:

- Đảm bảo JSON hợp lệ về cú pháp (không thiếu dấu ngoặc, không có dấu phẩy dư)
- Bắt buộc đúng cấu trúc yêu cầu (các trường bắt buộc đều có mặt)
- **Không** đảm bảo tính đúng đắn về ngữ nghĩa (giá trị vẫn có thể sai)

**Thiết kế schema — các nguyên tắc then chốt:**

```
{
  "type": "object",
  "properties": {
    "category": {
      "type": "string",
      "enum": ["bug", "feature", "docs", "unclear", "other"]
    },
    "category_detail": {
      "type": ["string", "null"],
      "description": "Details if category = 'other' or 'unclear'"
    },
    "severity": {
      "type": "string",
      "enum": ["critical", "high", "medium", "low"]
    },
    "confidence": {
      "type": "number",
      "minimum": 0,
      "maximum": 1
    },
    "optional_field": {
      "type": ["string", "null"],
      "description": "Null if the information was not found in the source"
    }
  },
  "required": ["category", "severity"]
}
```

**Các quy tắc thiết kế schema:**

1. **Bắt buộc so với tùy chọn:** chỉ đánh dấu một trường là bắt buộc nếu thông tin đó luôn có sẵn. Các trường bắt buộc sẽ thúc đẩy model bịa ra giá trị khi thiếu dữ liệu.
2. **Trường nullable:** dùng `"type": ["string", "null"]` cho thông tin có thể vắng mặt. Model có thể trả về `null` thay vì hallucinate.
3. **Enum có `"other"`:** thêm `"other"` + một chuỗi chi tiết để tránh mất dữ liệu nằm ngoài các danh mục đã định trước.
4. **Enum `"unclear"`:** cho các trường hợp model không thể tự tin chọn được một danh mục—một câu trả lời `"unclear"` trung thực vẫn tốt hơn một danh mục sai.

## 2.5 Lỗi Cú pháp so với Lỗi Ngữ nghĩa

| Loại lỗi | Ví dụ | Cách giảm thiểu |
|---|---|---|
| **Cú pháp** | JSON không hợp lệ, sai kiểu trường | `tool_use` kèm JSON schema (loại bỏ hẳn) |
| **Ngữ nghĩa** | Tổng cộng không khớp, giá trị nằm sai trường, hallucination | Kiểm tra validation, retry kèm phản hồi, tự sửa lỗi |

# Chương 3: Claude Agent SDK — Xây dựng Hệ thống Agentic

> Tài liệu: Agent SDK | Hooks | Subagents | Sessions

## 3.1 Agentic Loop là gì

Agentic loop là pattern cốt lõi để thực thi tác vụ một cách tự chủ. Model không chỉ trả lời—nó thực hiện một chuỗi hành động:

```
1. Send a request to Claude with tools
2. Receive a response
3. Check stop_reason:
   - "tool_use" -> execute the tool, append the result to history, go back to step 1
   - "end_turn" -> the task is complete, show the result to the user
4. Repeat until completion
```

**Đây là một cách tiếp cận do model điều khiển:** Claude quyết định gọi tool nào tiếp theo dựa trên context và các kết quả tool trước đó. Điều này khác với cây quyết định được hard-code, nơi chuỗi hành động là cố định.

**Anti-pattern (cần tránh):**

- Phân tích văn bản của assistant để phát hiện hoàn thành ("Task completed")
- Dùng một giới hạn số vòng lặp tùy ý (ví dụ, `max_iterations=5`) làm điều kiện dừng chính
- Kiểm tra xem assistant có sinh ra nội dung văn bản hay không như một tín hiệu hoàn thành

**Cách tiếp cận đúng:** tín hiệu hoàn thành đáng tin cậy duy nhất là `stop_reason == "end_turn"`.

## 3.2 Cấu hình `AgentDefinition`

`AgentDefinition` là đối tượng cấu hình agent trong Claude Agent SDK:

```
agent = AgentDefinition(
    name="customer_support",
    description="Handles customer requests for returns and order issues",
    system_prompt="You are a customer support agent...",
    allowed_tools=["get_customer", "lookup_order", "process_refund",
"escalate_to_human"],
    # Đối với một coordinator:
    # allowed_tools=["Task", "get_customer", ...]
)
```

**Các tham số then chốt:**

- `name` / `description` — định danh và mô tả của agent

- `system_prompt` — system prompt kèm chỉ dẫn
- `allowed_tools` — danh sách các tool được phép (nguyên tắc least privilege)

## 3.3 Hub-and-Spoke: Coordinator và Subagent

Kiến trúc multi-agent thường được xây dựng theo topology hub-and-spoke:

```
        Coordinator
       /     |      \
  Subagent1  Subagent2  Subagent3
   (search)   (analysis)   (synthesis)
```

**Coordinator chịu trách nhiệm:**

- Phân rã tác vụ thành các tác vụ con
- Quyết định cần những subagent nào (lựa chọn động)
- Ủy thác công việc cho các subagent
- Tổng hợp và kiểm tra validation kết quả
- Xử lý lỗi và retry
- Truyền đạt kết quả tới người dùng

**Nguyên tắc cốt lõi: các subagent có context tách biệt.**

- Các subagent **không** tự động kế thừa lịch sử hội thoại của coordinator
- Mọi context cần thiết phải được **truyền vào một cách tường minh** trong prompt của subagent
- Các subagent không chia sẻ bộ nhớ giữa các lần gọi
- Mọi giao tiếp đều đi qua coordinator (để có observability và kiểm soát lỗi)

## 3.4 Tool `Task` để Sinh ra Subagent

Các subagent được sinh ra thông qua tool `Task`:

```python
# allowedTools của coordinator phải bao gồm "Task"
coordinator_agent = AgentDefinition(
    allowed_tools=["Task", "get_customer"]
)
```

**Truyền context tường minh là bắt buộc:**

```
# Tệ: subagent không có context
Task: "Analyze the document"

# Tốt: đầy đủ context trong prompt
Task: "Analyze the following document.
Document: [full document text]
Prior search results: [web search results]
Output format requirements: [schema]"
```

**Sinh song song:** một coordinator có thể gọi nhiều `Task` trong một response—các subagent chạy song song:

```
# Một response của coordinator chứa:
Task 1: "Search for articles about X"
Task 2: "Analyze document Y"
Task 3: "Search for articles about Z"
# Cả ba chạy đồng thời
```

## 3.5 Hooks trong Agent SDK

Hook cho phép chặn và biến đổi tại các điểm cụ thể trong vòng đời của agent.

**PostToolUse** chặn một kết quả tool trước khi nó được cung cấp cho model:

```python
# Ví dụ: chuẩn hóa định dạng ngày tháng từ các MCP tool khác nhau
@hook("PostToolUse")
def normalize_dates(tool_result):
    # Chuyển Unix timestamp -> ISO 8601
    # Chuyển "Mar 5, 2025" -> "2025-03-05"
    return normalized_result
```

**Hook chặn cuộc gọi đi ra** chặn các hành động vi phạm chính sách:

```python
# Ví dụ: chặn hoàn tiền trên $500
@hook("PreToolUse")
def enforce_refund_limit(tool_call):
    if tool_call.name == "process_refund" and tool_call.args.amount > 500:
        return redirect_to_escalation(tool_call)
```

**Khác biệt then chốt: hook so với chỉ dẫn trong prompt**

| Thuộc tính | Hooks | Chỉ dẫn trong prompt |
|---|---|---|
| Đảm bảo | **Deterministic** (100%) | **Xác suất** (>90%, không phải 100%) |
| Khi nào dùng | Quy tắc nghiệp vụ then chốt, nghiệp vụ tài chính, tuân thủ | Tùy chọn chung, khuyến nghị, định dạng |
| Ví dụ | Chặn hoàn tiền > $500 | “Cố gắng giải quyết trước khi escalation” |

**Quy tắc:** khi thất bại gây hậu quả về tài chính, pháp lý, hoặc an toàn—hãy dùng hook, không dùng prompt.

# Chương 4: Model Context Protocol (MCP)

> Tài liệu: MCP | Tools | Resources | Servers

## 4.1 MCP là gì

Model Context Protocol (MCP) là một giao thức mở để kết nối các hệ thống bên ngoài với Claude. MCP định nghĩa ba loại tài nguyên chính:

1. **Tools** — các hàm mà agent có thể gọi để thực hiện hành động (thao tác CRUD, gọi API, thực thi lệnh)
2. **Resources** — dữ liệu mà agent có thể đọc để lấy context (tài liệu, schema cơ sở dữ liệu, danh mục nội dung)
3. **Prompts** — các mẫu prompt định sẵn cho những tác vụ thường gặp

## 4.2 MCP Server

MCP server là một tiến trình triển khai giao thức MCP và cung cấp tools/resources. Khi bạn kết nối tới một MCP server:

- Tất cả các tool được khám phá tự động
- Các tool từ mọi server đã kết nối đều sẵn dùng cùng lúc
- Mô tả của tool quyết định cách model sẽ sử dụng chúng

## 4.3 Cấu hình MCP Server

**Cấu hình theo dự án (`.mcp.json`)** — dùng cho cả nhóm:

```
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_TOKEN": "${GITHUB_TOKEN}"
      }
    },
    "jira": {
      "command": "npx",
      "args": ["-y", "mcp-server-jira"],
      "env": {
        "JIRA_TOKEN": "${JIRA_TOKEN}"
      }
    }
  }
}
```

**Điểm chính:**

- `.mcp.json` được lưu ở thư mục gốc của dự án và quản lý trong version control
- Biến môi trường (`${GITHUB_TOKEN}`) được dùng cho thông tin bí mật—bản thân các token không được commit
- Sẵn dùng cho mọi người đóng góp vào dự án

**Cấu hình theo người dùng (`~/.claude.json`)** — dùng cho các server cá nhân/thử nghiệm:

- Lưu trong thư mục home của người dùng
- Không được chia sẻ qua version control
- Phù hợp cho các thử nghiệm và kiểm thử cá nhân

**Chọn server:**

- Với các tích hợp tiêu chuẩn (Jira, GitHub, Slack), nên ưu tiên các MCP server cộng đồng sẵn có
- Chỉ tự xây dựng server cho những workflow đặc thù, riêng của nhóm

## 4.4 Cờ `isError` trong MCP

Khi một MCP tool gặp lỗi, nó dùng `isError: true` trong phản hồi. Điều này báo cho agent biết lệnh gọi đã thất bại.

**Lỗi có cấu trúc (tốt):**

```
{
  "isError": true,
  "content": {
    "errorCategory": "transient",
    "isRetryable": true,
    "message": "The service is temporarily unavailable. Timeout while calling the
orders API.",
    "attempted_query": "order_id=12345",
    "partial_results": null
  }
}
```

**Lỗi chung chung (anti-pattern):**

```
{
  "isError": true,
  "content": "Operation failed"
}
```

Một lỗi chung chung không cung cấp cho agent thông tin nào để ra quyết định—nên retry, đổi truy vấn, hay escalation?

## 4.5 MCP Resources

Resources là dữ liệu mà agent có thể yêu cầu để lấy context mà không cần thực hiện hành động:

- Danh mục nội dung (ví dụ: danh sách toàn bộ tác vụ của dự án, điều hướng phân cấp)
- Schema cơ sở dữ liệu (hiểu cấu trúc dữ liệu)
- Tài liệu (tham chiếu API, hướng dẫn nội bộ)
- Tóm tắt issue/tác vụ

**Lợi thế của resource:** agent không cần các lệnh gọi tool thăm dò để hiểu dữ liệu nào đang tồn tại. Một resource cung cấp ngay một "tấm bản đồ."

---

# Chương 5: Claude Code — Cấu hình và Workflow

> Tài liệu: Claude Code | Memory / CLAUDE.md | Skills | MCP | Hooks | Sub-agents | GitHub Actions | Headless

## 5.1 Phân cấp CLAUDE.md

CLAUDE.md là (các) tệp chỉ dẫn cho Claude Code. Có một hệ phân cấp ba mức:

```
1. Mức người dùng (User-level): ~/.claude/CLAUDE.md
   - Chỉ áp dụng cho người dùng đó
   - KHÔNG được chia sẻ qua VCS
   - Sở thích cá nhân và phong cách làm việc

2. Mức dự án (Project-level): .claude/CLAUDE.md hoặc một CLAUDE.md ở thư mục gốc
   - Áp dụng cho mọi người đóng góp vào dự án
   - Được quản lý qua VCS
   - Chuẩn viết code, chuẩn kiểm thử, các quyết định kiến trúc

3. Mức thư mục (Directory-level): CLAUDE.md trong các thư mục con
   - Áp dụng khi làm việc với các tệp trong thư mục đó
   - Quy ước riêng cho phần đó của codebase
```

**Lỗi thường gặp:** một thành viên mới của nhóm không nhận được các chỉ dẫn của dự án vì chúng được đặt trong `~/.claude/CLAUDE.md` (mức người dùng) thay vì `.claude/CLAUDE.md` (mức dự án).

## 5.2 Cú pháp `@path` (Import tệp)

CLAUDE.md có thể tham chiếu các tệp bên ngoài bằng `@path`, giúp cấu hình có tính module:

```
# Project CLAUDE.md

Coding standards are described in @./standards/coding-style.md
Test requirements are in @./standards/testing-requirements.md
Project overview is in @README.md and dependencies are in @package.json
```

**Quy tắc cho `@path`:**

- Dùng `@` ngay trước đường dẫn tệp (không có dấu cách)
- Hỗ trợ cả đường dẫn tương đối và tuyệt đối
- Đường dẫn tương đối được phân giải tương đối với tệp chứa import đó
- Độ sâu lồng nhau tối đa của import là 5

Cách này tránh trùng lặp và cho phép mỗi package chỉ bao gồm các chuẩn liên quan.

## 5.3 Thư mục `.claude/rules/`

`.claude/rules/` là một giải pháp thay thế cho một CLAUDE.md nguyên khối, dùng để tổ chức các quy tắc theo chủ đề:

```
.claude/rules/
  testing.md          -- testing conventions
  api-conventions.md  -- API conventions
  deployment.md       -- deployment rules
  react-patterns.md   -- React patterns
```

**Tính năng then chốt: YAML frontmatter với `paths` để nạp có điều kiện:**

```
---
paths: ["src/api/**/*"]
---

For API files, use async/await with explicit error handling.
Each endpoint must return a standard response wrapper.
```

```
---
paths: ["**/*.test.tsx", "**/*.test.ts"]
---

Tests must use describe/it blocks.
Use data factories instead of hardcoding.
Do not mock the database—use a test database.
```

**Cách hoạt động:**

- Một quy tắc **chỉ** được nạp khi Claude Code chỉnh sửa một tệp khớp với mẫu `paths`

- Điều này tiết kiệm context và token—các quy tắc không liên quan sẽ không được nạp
- Mẫu glob cho phép bạn áp dụng quy ước theo loại tệp bất kể vị trí (lý tưởng cho các test nằm rải rác khắp codebase)

**Khi nào dùng `.claude/rules/` với `paths` so với CLAUDE.md mức thư mục:**

- `.claude/rules/` với `paths` — khi các quy ước áp dụng cho những tệp nằm rải rác qua nhiều thư mục (test, migration)
- CLAUDE.md mức thư mục — khi các quy ước gắn với một thư mục cụ thể và không cần ở nơi khác

## 5.4 Slash Command tùy chỉnh và Skill

> **Lưu ý:** ở phiên bản Claude Code hiện tại, các lệnh tùy chỉnh (`.claude/commands/`) được hợp nhất với skill (`.claude/skills/`). Cả hai định dạng đều tạo ra các lệnh `/name`. Hướng dẫn thi tham chiếu tới `.claude/commands/`—định dạng đó vẫn được hỗ trợ.

Slash command là các mẫu prompt tái sử dụng, được gọi qua `/name`:

**Định dạng `.claude/commands/` (cũ, vẫn được hỗ trợ):**

```
.claude/commands/
  review.md        -- /review -- standard code review
  test-gen.md      -- /test-gen -- test generation
```

**Định dạng `.claude/skills/` (hiện tại):**

```
.claude/skills/
  review/SKILL.md  -- /review -- with frontmatter configuration
  test-gen/SKILL.md
```

**Lệnh của dự án** (`.claude/commands/` hoặc `.claude/skills/`):

- Được lưu trong VCS và sẵn dùng cho mọi người khi clone repo
- Đảm bảo các workflow nhất quán trong toàn nhóm

**Lệnh của người dùng** (`~/.claude/commands/` hoặc `~/.claude/skills/`):

- Lệnh cá nhân không được chia sẻ qua VCS
- Dùng cho workflow của từng cá nhân

## 5.5 Skill — `.claude/skills/`

Skill là các lệnh nâng cao được cấu hình qua frontmatter của SKILL.md:

```
---
context: fork
allowed-tools: ["Read", "Grep", "Glob"]
argument-hint: "Path to the directory to analyze"
---

Analyze the code structure in the specified directory.
Output a report on dependencies and architectural patterns.
```

**Các tham số frontmatter:**

| Tham số | Mô tả |
|---|---|
| `context: fork` | Chạy skill trong một subagent tách biệt. Đầu ra dài dòng không làm nhiễu session chính |
| `allowed-tools` | Giới hạn những tool nào được sẵn dùng (bảo mật—ví dụ: skill không thể xóa tệp nếu không được cho phép) |
| `argument-hint` | Gợi ý yêu cầu một đối số khi được gọi mà không có tham số |

**Khi nào dùng skill so với CLAUDE.md:**

- **Skill** — gọi theo nhu cầu cho một tác vụ cụ thể (review, phân tích, sinh code)
- **CLAUDE.md** — các chuẩn và quy ước chung luôn được nạp

**Skill cá nhân (`~/.claude/skills/`):**

- Tạo các biến thể cá nhân dưới những tên khác để bạn không ảnh hưởng tới đồng đội

## 5.6 Planning Mode so với Thực thi trực tiếp

**Planning mode:**

- Model chỉ điều tra và lập kế hoạch; không thực hiện thay đổi
- Dùng Read, Grep, Glob để khám phá codebase
- Tạo ra một kế hoạch triển khai để người dùng phê duyệt
- Khám phá an toàn, không có tác dụng phụ

**Khi nào dùng planning mode:**

- Thay đổi lớn (hàng chục tệp)
- Nhiều cách tiếp cận hợp lý (microservices: định nghĩa ranh giới thế nào?)
- Các quyết định kiến trúc (framework nào? cấu trúc ra sao?)
- Codebase chưa quen thuộc (bạn phải hiểu trước khi thay đổi)
- Migration thư viện ảnh hưởng tới 45+ tệp

**Khi nào dùng thực thi trực tiếp:**

- Sửa lỗi trong một tệp với stack trace rõ ràng
- Thêm một bước validation
- Các thay đổi đã hiểu rõ, không mơ hồ

**Cách tiếp cận kết hợp:**

1. Planning mode để điều tra và thiết kế
2. Người dùng phê duyệt kế hoạch
3. Thực thi trực tiếp để triển khai kế hoạch đã được duyệt

**Subagent Explore** — một subagent chuyên dụng để khám phá codebase:

- Tách đầu ra dài dòng khỏi context chính
- Chỉ trả về một bản tóm tắt
- Ngăn cạn kiệt context window trong các tác vụ nhiều giai đoạn

## 5.7 Lệnh `/compact`

`/compact` là một lệnh tích hợp sẵn để nén context:

- Tóm tắt lịch sử trước đó để giải phóng context window
- Dùng trong các session điều tra dài khi context bị lấp đầy bởi đầu ra tool dài dòng
- Rủi ro: các giá trị số chính xác, ngày tháng và chi tiết cụ thể có thể bị mất trong quá trình tóm tắt

## 5.8 Lệnh `/memory`

`/memory` là một lệnh tích hợp sẵn để quản lý bộ nhớ giữa các session:

- Mở tệp `CLAUDE.md` để chỉnh sửa, cho phép bạn lưu ghi chú, sở thích và context
- Thông tin được giữ lại qua các session và tự động nạp khi khởi động
- Hữu ích để lưu trữ quy ước dự án, sở thích người dùng, các lệnh thường dùng và context công việc hiện tại
- Một giải pháp thay thế cho việc giải thích lại cùng các chỉ dẫn trong mỗi session

## 5.9 Claude Code CLI cho CI/CD

**Cờ `-p` (hoặc `--print`):**

```
claude -p "Analyze this pull request for security issues"
```

- Chế độ không tương tác: xử lý prompt, in ra stdout, rồi thoát
- Không chờ người dùng nhập liệu
- Cách đúng duy nhất để chạy Claude trong các pipeline CI/CD

**Structured output cho CI:**

```
claude -p "Review this PR" --output-format json --json-schema
'{"type":"object",...}'
```

- `--output-format json` — đầu ra ở dạng JSON
- `--json-schema` — validation đầu ra so với một schema
- Kết quả có thể được phân tích cú pháp để tự động đăng các bình luận inline trên PR

**Tách biệt context của session:** Cùng một session Claude đã tạo ra code thường kém hiệu quả hơn khi review nó (model giữ lại context lập luận của mình và ít có khả năng phản biện chính các quyết định của nó). Hãy dùng một instance độc lập để review.

**Ngăn các bình luận trùng lặp:** Khi review lại sau các commit mới, hãy đưa kết quả review trước đó vào context và chỉ dẫn Claude chỉ báo cáo các vấn đề mới hoặc chưa được giải quyết.

## 5.10 `fork_session` và Quản lý Session

**`--resume <session-name>`** tiếp tục một session đã đặt tên:

```
claude --resume investigation-auth-bug
```

- Tiếp tục một cuộc hội thoại trước đó với context đã lưu
- Hữu ích cho các cuộc điều tra dài trải qua nhiều session
- Rủi ro: nếu các tệp đã thay đổi kể từ session trước, kết quả của tool có thể bị lỗi thời

**`fork_session`** tạo một nhánh độc lập từ context dùng chung:

```
Codebase investigation
         |
    fork_session
    /           \
Approach A:      Approach B:
Redux            Context API
```

- Cả hai fork đều kế thừa context cho tới điểm phân nhánh
- Sau đó, chúng phân kỳ độc lập với nhau
- Hữu ích để so sánh các cách tiếp cận hoặc thử nghiệm các chiến lược

**Khi nào nên bắt đầu một session mới thay vì tiếp tục (resume):**

- Kết quả của tool đã lỗi thời (các tệp đã thay đổi)
- Đã trôi qua nhiều thời gian và context đã suy giảm
- Tốt hơn nên khởi động lại với "Đây là bản tóm tắt ngắn về những gì chúng ta đã tìm thấy: ..." thay vì tiếp tục với dữ liệu tool cũ

---

# Chương 6: Prompt Engineering — Các kỹ thuật nâng cao

> Tài liệu: Prompt Engineering | Anthropic Cookbook

## 6.1 Few-shot Prompting

Few-shot prompting là việc đưa vào prompt 2–4 ví dụ input/output để minh họa hành vi mong đợi.

**Vì sao few-shot hiệu quả hơn so với mô tả bằng văn bản:**

- Một chỉ dẫn mơ hồ như “hãy chính xác hơn” có thể được hiểu theo nhiều cách
- Một ví dụ cho thấy rõ ràng, không nhập nhằng định dạng và logic ra quyết định mong đợi
- Mô hình khái quát hóa mẫu (pattern) cho các trường hợp mới (nó không chỉ lặp lại các ví dụ)

**Các loại ví dụ few-shot và khi nào dùng chúng:**

1. **Ví dụ cho các kịch bản nhập nhằng:**

```
Request: "My order is broken"
Action: Call get_customer -> lookup_order -> check status.
Rationale: “broken” may mean a damaged item; you need order details.

Request: "Get me a manager"
Action: Immediately call escalate_to_human.
Rationale: The customer explicitly requests a human. Do not attempt to solve
autonomously.
```

2. **Ví dụ cho việc định dạng output:**

```
Finding example:
{
  "location": "src/auth/login.ts:42",
  "issue": "SQL injection in the username parameter",
  "severity": "critical",
  "suggested_fix": "Use a parameterized query"
}
```

3. **Ví dụ để phân biệt code chấp nhận được với code có vấn đề:**

```
// Chấp nhận được (không gắn cờ):
const items = data.filter(x => x.active);

// Có vấn đề (gắn cờ):
const items = data.filter(x => x.active == true); // Dùng so sánh nghiêm ngặt ===
```

4. **Ví dụ cho việc trích xuất từ các định dạng tài liệu khác nhau:**

```
Document with inline citations:
"As shown in the study (Smith, 2023), the rate is 42%."
-> {"value": "42%", "source": "Smith, 2023", "type": "inline_citation"}

Document with bibliography references:
"The rate is 42%. [1]"
-> {"value": "42%", "source": "reference_1", "type": "bibliography"}
```

5. **Ví dụ cho các phép đo không chính thức:**

```
Text: "about two handfuls of rice"
-> {"amount": "~100g", "original_text": "two handfuls", "precision": "approximate"}

Text: "a pinch of salt"
-> {"amount": "~1g", "original_text": "a pinch", "precision": "approximate"}
```

Few-shot đặc biệt hiệu quả khi trích xuất các đơn vị đo lường không chính thức và phi tiêu chuẩn vốn quá đa dạng để chỉ dựa vào các chỉ dẫn theo quy tắc thuần túy.

**Các quy tắc chuẩn hóa định dạng trong prompt:** Khi sử dụng JSON schema nghiêm ngặt cho structured output, hãy thêm các quy tắc chuẩn hóa vào prompt:

```
Normalization:
- Dates: always ISO 8601 (YYYY-MM-DD); "yesterday" -> compute an absolute date
- Currency: numeric amount + currency code; "five bucks" -> {"amount": 5, "currency":
"USD"}
- Percentages: decimal fraction; "half" -> 0.5
```

Điều này ngăn ngừa các lỗi ngữ nghĩa khi JSON hợp lệ về mặt cú pháp nhưng các giá trị lại không nhất quán.

## 6.2 Tiêu chí rõ ràng so với chỉ dẫn mơ hồ

**Tệ (mơ hồ):**

```
Check code comments for accuracy.
Be conservative—report only high-confidence findings.
```

**Tốt (tiêu chí rõ ràng):**

```
Flag a comment as problematic ONLY if:
1. The comment describes behavior that CONTRADICTS the actual code behavior
2. The comment references a non-existent function or variable
3. A TODO/FIXME comment refers to a bug that has already been fixed in code

Do NOT flag:
- Comments that are merely stylistically outdated
- Comments with minor wording inaccuracies
- Missing comments (that is a separate category)
```

**Định nghĩa tiêu chí mức độ nghiêm trọng kèm ví dụ:**

```
CRITICAL: Runtime failure for users
  Example: NullPointerException while processing a payment

HIGH: Security vulnerability
  Example: SQL injection, XSS, missing authorization checks

MEDIUM: Logic bug without immediate impact
  Example: Wrong sorting, off-by-one error

LOW: Code quality
  Example: Duplication, suboptimal algorithm for small data
```

## 6.3 Prompt Chaining

Prompt chaining chia một tác vụ phức tạp thành một chuỗi các bước tập trung:

```
Step 1: Analyze auth.ts (local issues only)
       -> Output: list of issues in auth.ts

Step 2: Analyze database.ts (local issues only)
       -> Output: list of issues in database.ts

Step 3: Integration pass (cross-file dependencies)
       -> Output: issues at module boundaries
```

**Vì sao điều này quan trọng:**

- Tránh **sự pha loãng chú ý (attention dilution)**—khi mô hình nhận quá nhiều file cùng lúc, nó có thể bỏ sót lỗi trong một số file đồng thời chỉ đưa ra nhận xét hời hợt cho những file khác
- Đảm bảo chất lượng phân tích nhất quán cho mỗi file
- Cho phép phân tích riêng các tương tác giữa các file (cross-file)

**Khi nào dùng prompt chaining so với phân rã động (dynamic decomposition):**

- **Prompt chaining** — các tác vụ có thể dự đoán, lặp lại được (code review, migration file)
- **Dynamic decomposition** — các cuộc điều tra mở (open-ended) nơi các tác vụ con chỉ trở nên rõ ràng trong quá trình thực thi

## 6.4 Mẫu “Phỏng vấn” (Interview)

Trước khi triển khai một giải pháp, Claude đặt các câu hỏi làm rõ:

```
Claude: "Before implementing caching for the API, a few questions:
1. Which cache invalidation strategy do you prefer—TTL or event-based?
2. Is stale data acceptable when the cache is unavailable?
3. Should caching be per-user or global?
4. What is the expected data volume to cache?"
```

**Khi nào điều này hữu ích:**

- Lĩnh vực không quen thuộc (fintech, y tế, hệ thống pháp lý)
- Các tác vụ có hệ quả không hiển nhiên (chiến lược cache, các chế độ lỗi)
- Có nhiều cách tiếp cận khả thi mà lựa chọn tốt nhất phụ thuộc vào ngữ cảnh

## 6.5 Validation và Retry-with-Feedback

Khi dữ liệu trích xuất không vượt qua validation:

```
Step 1: Extract data from the document
Step 2: Validate (Pydantic, JSON Schema, business rules)
Step 3: If there's an error—retry with context:
  - The original document
  - The previous (incorrect) extraction
  - The specific error: "Field 'total' = 150, but sum(line_items) = 145. Re-check
values."
```

**Khi nào retry sẽ hiệu quả:**

- Lỗi định dạng (ngày sai định dạng)
- Lỗi cấu trúc (một trường được đặt sai vị trí)
- Sự không nhất quán về số học (mô hình có thể kiểm tra lại)

**Khi nào retry sẽ KHÔNG giúp ích:**

- Thông tin không có trong tài liệu nguồn
- Ngữ cảnh cần thiết nằm bên ngoài (dữ liệu nằm trong một tài liệu khác không được cung cấp)

**Pydantic như một công cụ validation:** Pydantic là một thư viện Python để validation dữ liệu dựa trên schema. Đối với kỳ thi, các điểm then chốt là:

- **Validation cấu trúc:** kiểu dữ liệu, tính bắt buộc, ràng buộc enum được kiểm tra trong code sau khi nhận JSON từ Claude
- **Validation ngữ nghĩa:** các validator tùy chỉnh thực thi logic nghiệp vụ (tổng các mục bằng tổng cuối; start_date < end_date)
- **Vòng lặp validate–retry:** khi validation của Pydantic thất bại, dựng một thông báo lỗi và prompt lại Claude với ngữ cảnh lỗi
- **Sinh JSON Schema:** các model Pydantic có thể sinh JSON Schema cho `tool_use`, cung cấp một nguồn chân lý duy nhất (single source of truth)

## 6.6 Tự sửa lỗi (Self-correction)

Một mẫu để phát hiện mâu thuẫn nội tại:

```
{
  "stated_total": "$150.00",
  "calculated_total": "$145.00",
  "conflict_detected": true,
  "line_items": [
    {"name": "Widget A", "price": 75.00},
    {"name": "Widget B", "price": 70.00}
  ]
}
```

Mô hình trích xuất cả giá trị được nêu ra và một giá trị được tính toán—nếu chúng khác nhau, `conflict_detected` cho phép bạn xử lý sự chênh lệch.

---

# Chương 7: Message Batches API

> Tài liệu: Message Batches

## 7.1 Tổng quan

Message Batches API cho phép bạn gửi các batch yêu cầu để xử lý bất đồng bộ:

| Thuộc tính | Giá trị |
|---|---|
| Tiết kiệm | **50%** so với các lệnh gọi đồng bộ |
| Cửa sổ xử lý | Lên đến **24 giờ** (không có cam kết SLA về latency) |
| Tool calling nhiều lượt | **Không được hỗ trợ** (một yêu cầu = một phản hồi) |
| Tương quan | Trường `custom_id` để liên kết yêu cầu và phản hồi |

## 7.2 Khi nào dùng Batch API so với Synchronous API

| Tác vụ | API | Vì sao |
|---|---|---|
| Kiểm tra PR trước khi merge | **Synchronous** | Lập trình viên đang chờ; 24 giờ là không chấp nhận được |
| Báo cáo nợ kỹ thuật qua đêm | **Batch** | Kết quả cần có vào sáng hôm sau; tiết kiệm 50% |
| Kiểm toán bảo mật hằng tuần | **Batch** | Không khẩn cấp; tiết kiệm 50% |
| Code review tương tác | **Synchronous** | Yêu cầu phản hồi tức thì |
| Xử lý 10.000 tài liệu | **Batch** | Xử lý khối lượng lớn; tiết kiệm là đáng kể |

## 7.3 Sử dụng `custom_id`

```
{
  "custom_id": "doc-invoice-2024-001",
  "params": {
    "model": "claude-sonnet-4-6",
    "max_tokens": 1024,
    "messages": [{"role": "user", "content": "Extract data from: ..."}]
  }
}
```

`custom_id` cho phép bạn:

- Liên kết kết quả với tài liệu gốc
- Khi thất bại, gửi lại chỉ những tài liệu bị lỗi
- Tránh xử lý lại những tài liệu đã thành công

## 7.4 Xử lý các thất bại trong Batch

1. Gửi một batch gồm 100 tài liệu
2. 95 tài liệu thành công; 5 tài liệu thất bại (vượt giới hạn context)
3. Xác định các tài liệu lỗi bằng `custom_id`
4. Điều chỉnh chiến lược (ví dụ, chia các tài liệu dài thành các đoạn nhỏ)
5. Gửi lại chỉ 5 tài liệu bị lỗi

## 7.5 Lập kế hoạch SLA

Nếu bạn cần kết quả trong 30 giờ và Batch API có thể mất đến 24 giờ:

- Cửa sổ gửi: 30 - 24 = **6 giờ**
- Các batch phải được gửi không muộn hơn 24 giờ trước hạn chót
- Đối với việc gửi thường xuyên, hãy chia thành các cửa sổ 4 giờ

---

# Chương 8: Các chiến lược phân rã tác vụ

## 8.1 Pipeline cố định (Prompt Chaining)

Mỗi bước được định nghĩa trước:

```
Document -> Metadata extraction -> Data extraction -> Validation -> Enrichment -> Final output
```

**Khi nào dùng:**

- Cấu trúc tác vụ có thể dự đoán (các bản review luôn theo cùng một template)
- Tất cả các bước đều được biết trước
- Bạn cần sự ổn định và khả năng tái lập

## 8.2 Phân rã thích ứng động (Dynamic Adaptive Decomposition)

Các tác vụ con được sinh ra dựa trên các kết quả trung gian:

```
1. "Add tests for a legacy codebase"
2. -> First: map the structure (Glob, Grep)
3. -> Found: 3 modules with no tests, 2 with partial coverage
4. -> Prioritize: start with the payments module (high risk)
5. -> During work: discovered a dependency on an external API
6. -> Adapt: add a mock for the external API before writing tests
```

**Khi nào dùng:**

- Các tác vụ điều tra mở (open-ended)
- Khi không biết trước toàn bộ phạm vi
- Khi mỗi bước phụ thuộc vào kết quả của bước trước

## 8.3 Code Review nhiều lượt (Multi-pass)

Đối với các pull request có 10+ file:

```
Pass 1 (per-file): Analyze auth.ts -> list local issues
Pass 1 (per-file): Analyze database.ts -> list local issues
Pass 1 (per-file): Analyze routes.ts -> list local issues
...
Pass 2 (integration): Analyze relationships between files
  -> Cross-file issues: inconsistent types, circular dependencies
```

**Vì sao một lượt duy nhất qua 14 file là tệ:**

- Sự pha loãng chú ý (attention dilution): phân tích sâu cho một số file, hời hợt cho những file khác
- Nhận xét không nhất quán: một mẫu bị gắn cờ ở file này nhưng được chấp nhận ở file khác
- Bỏ sót lỗi: các lỗi hiển nhiên bị bỏ qua do quá tải nhận thức

---

# Chương 9: Escalation và Human-in-the-Loop

## 9.1 Khi nào escalation lên cho con người

**Các tác nhân kích hoạt escalation (quy tắc rõ ràng):**

| Tình huống | Hành động |
|---|---|

| | |
|---|---|
| Khách hàng yêu cầu rõ ràng “get me a manager” | Escalation ngay lập tức; không cố tự giải quyết |
| Chính sách không bao quát yêu cầu | Escalation (ví dụ, so giá với đối thủ cạnh tranh khi chính sách không đề cập) |
| Agent không thể tiến triển | Escalation sau một số lần thử hợp lý |
| Thao tác tài chính vượt ngưỡng | Escalation (tốt nhất nên thực thi qua một hook, không phải bằng prompt) |
| Nhiều kết quả khớp khi tìm kiếm khách hàng | Yêu cầu thêm định danh; không phỏng đoán |

**Điều gì KHÔNG phải là tác nhân kích hoạt đáng tin cậy:**

| **Phương pháp không đáng tin cậy** | **Vì sao nó thất bại** |
|---|---|
| Phân tích cảm xúc (sentiment analysis) | Tâm trạng khách hàng không tương quan với độ phức tạp của vụ việc |
| Mức độ tự tin do mô hình tự đánh giá (1–10) | Mô hình có thể sai một cách tự tin; hiệu chỉnh (calibration) kém |
| Một bộ phân loại tự động | Overengineering; có thể đòi hỏi dữ liệu huấn luyện mà bạn không có |

## 9.2 Các mẫu Escalation

**Escalation ngay lập tức:**

```
Customer: "I want to speak to a manager"
Agent: [immediately calls escalate_to_human]
NOT: "I can help with your issue, let me..."
```

**Escalation sau khi cố gắng giải quyết:**

```
Customer: "My refrigerator broke two days after purchase"
Agent: [checks the order, offers a warranty replacement]
If the customer is not satisfied -> escalate
```

**Escalation tinh tế (ghi nhận → giải quyết → escalation khi khách hàng nhắc lại):**

```
Customer: "This is outrageous, I'm very unhappy with the quality!"
Agent: [acknowledges frustration] "I understand your frustration."
       [offers resolution] "I can offer a replacement or a refund."
Customer: "No, I want to talk to someone!"
Agent: [customer insists again -> immediate escalation]
```

Nguyên tắc then chốt: ghi nhận cảm xúc trước, sau đó đề xuất một giải pháp cụ thể, và chỉ escalation nếu khách hàng nhắc lại mong muốn gặp người thật. Đừng escalation ngay lần đầu khách hàng bày tỏ sự

không hài lòng (điều đó không giống với việc yêu cầu gặp quản lý).

**Escalation cho một lỗ hổng chính sách:**

```
Customer: "Competitor X has this item 30% cheaper—give me a discount"
Policy: covers price adjustments only on your own site
Agent: [escalates — policy does not cover competitor price matching]
```

## 9.3 Các giao thức bàn giao có cấu trúc (Structured Handoff)

Khi escalation, agent nên chuyển một bản tóm tắt có cấu trúc cho con người:

```
{
  "customer_id": "CUST-12345",
  "customer_name": "Ivan Petrov",
  "issue_summary": "Refund request for a damaged item",
  "order_id": "ORD-67890",
  "root_cause": "Item arrived damaged; photos attached",
  "actions_taken": [
    "Verified customer via get_customer",
    "Confirmed order via lookup_order",
    "Offered a standard replacement — customer insists on a refund"
  ],
  "refund_amount": "$89.99",
  "recommended_action": "Approve a full refund",
  "escalation_reason": "Customer requested to speak with a manager"
}
```

Người vận hành là con người không có quyền truy cập toàn bộ bản ghi hội thoại—họ chỉ thấy bản tóm tắt này. Do đó nó phải đầy đủ và tự chứa (self-contained).

## 9.4 Hiệu chỉnh độ tin cậy và giám sát của con người

Đối với các hệ thống trích xuất dữ liệu:

1. **Điểm tin cậy ở cấp độ trường:** mô hình xuất ra một điểm tin cậy cho mỗi trường được trích xuất
2. **Hiệu chỉnh (Calibration):** dùng các tập validation đã gắn nhãn để tinh chỉnh các ngưỡng
3. **Định tuyến (Routing):**
   - Độ tin cậy cao + độ chính xác ổn định -> xử lý tự động
   - Độ tin cậy thấp hoặc nguồn nhập nhằng -> con người review

**Lấy mẫu ngẫu nhiên phân tầng (Stratified random sampling):**

- Ngay cả với các lần trích xuất có độ tin cậy cao, hãy thường xuyên kiểm toán một mẫu
- Độ chính xác tổng hợp 97% có thể che giấu 40% lỗi đối với một loại tài liệu cụ thể
- Phân tích độ chính xác theo loại tài liệu và theo trường, không chỉ tổng thể

---

# Chương 10: Xử lý lỗi trong các hệ thống đa agent

## 10.1 Các nhóm lỗi

| Nhóm | Ví dụ | Có thể retry | Hành động của agent |
|---|---|---|---|
| **Tạm thời (Transient)** | Timeout, 503, lỗi mạng | Có | Retry với exponential backoff |
| **Validation** | Định dạng input không hợp lệ, thiếu trường bắt buộc | Không (sửa input) | Điều chỉnh yêu cầu và retry |
| **Nghiệp vụ (Business)** | Vi phạm chính sách, vượt ngưỡng | Không | Giải thích cho người dùng; đề xuất phương án thay thế |
| **Quyền (Permission)** | Bị từ chối truy cập | Không | Escalation |

## 10.2 Các anti-pattern trong xử lý lỗi

| Anti-pattern | Vấn đề | Cách tiếp cận đúng |
|---|---|---|
| Trạng thái chung chung “search unavailable” | Coordinator không thể quyết định cách khôi phục | Trả về loại lỗi, query, kết quả một phần, phương án thay thế |
| Ém lỗi âm thầm (kết quả rỗng = thành công) | Coordinator tưởng rằng không có kết quả khớp, nhưng thực ra đó là một thất bại | Phân biệt “không có kết quả” với “tìm kiếm thất bại” |
| Hủy toàn bộ workflow khi một lần thất bại | Bạn mất tất cả các kết quả một phần | Tiếp tục với kết quả một phần; chú thích các lỗ hổng |
| Retry vô hạn bên trong một subagent | Latency và lãng phí tài nguyên | Khôi phục cục bộ (1–2 lần retry), sau đó lan truyền lên coordinator |

## 10.3 Một lỗi subagent có cấu trúc

```
{
  "status": "partial_failure",
  "failure_type": "timeout",
  "attempted_query": "AI impact on music industry 2024",
  "partial_results": [
    {"title": "AI Music Generation Report", "url": "...", "relevance": 0.8}
  ],
  "alternative_approaches": [
    "Try a narrower query: 'AI music composition tools'",
    "Use an alternative data source"
  ],
  "coverage_impact": "Not covered: AI impact on music production"
}
```

Điều này cung cấp cho coordinator thông tin cần thiết để quyết định:

- Retry với một query đã chỉnh sửa?
- Dùng kết quả một phần?
- Ủy thác cho một subagent khác?
- Tiếp tục mà không có phần này và chú thích lỗ hổng?

## 10.4 Chú thích độ bao phủ trong bản tổng hợp cuối cùng

```
## Báo cáo: Tác động của AI lên các ngành sáng tạo

### Visual Art (BAO PHỦ ĐẦY ĐỦ)
[research results]

### Music (BAO PHỦ MỘT PHẦN — search agent timeout)
[partial results]
⚠️ Note: coverage for this section is limited due to a timeout in the search agent.

### Literature (BAO PHỦ ĐẦY ĐỦ)
[research results]
```

---

# Chương 11: Quản lý Context trong Hệ thống Production

## 11.1 Trích xuất các Sự kiện ra một Block Riêng

Thay vì dựa vào lịch sử hội thoại (vốn bị suy giảm trong quá trình tóm tắt), hãy trích xuất các sự kiện quan trọng vào một block có cấu trúc:

```
=== CASE FACTS (updated whenever a new fact appears) ===
Customer ID: CUST-12345
Order ID: ORD-67890
Order Date: 2025-01-15
Order Amount: $89.99
Issue: Damaged item on delivery
Customer Request: Full refund
Status: Pending manager approval
===
```

Đưa block này vào mọi prompt, bất kể lịch sử được tóm tắt như thế nào.

## 11.2 Cắt gọn Kết quả Tool

Nếu `lookup_order` trả về hơn 40 trường nhưng bạn chỉ cần 5 trường cho tác vụ hiện tại:

```python
# PostToolUse hook: chỉ giữ lại các trường liên quan
@hook("PostToolUse", tool="lookup_order")
def trim_order_fields(result):
    return {
        "order_id": result["order_id"],
        "status": result["status"],
        "total": result["total"],
        "items": result["items"],
        "return_eligible": result["return_eligible"]
    }
```

Cách này tiết kiệm context và giảm nhiễu.

## 11.3 Đầu vào Nhận biết Vị trí

Đặt thông tin then chốt với lưu ý về hiệu ứng lost-in-the-middle:

```
[KEY FINDINGS — at the top]
Found 3 critical vulnerabilities...

[DETAILED RESULTS — middle]
=== File auth.ts ===
...
=== File database.ts ===
...

[ACTION ITEMS — at the end]
Priority: fix auth.ts vulnerabilities before merge.
```

## 11.4 Tệp Scratchpad

Trong các cuộc điều tra dài, agent có thể ghi các phát hiện quan trọng vào một tệp scratchpad:

```
# investigation-scratchpad.md
## Key findings
- PaymentProcessor in src/payments/processor.ts inherits from BaseProcessor
- refund() is called from 3 places: OrderController, AdminPanel, CronJob
- External PaymentGateway API has a rate limit of 100 req/min
- Migration #47 added refund_reason (NOT NULL) — 2024-12-01
```

Khi context bị suy giảm (hoặc trong một session mới), agent có thể tham khảo scratchpad thay vì chạy lại quá trình khám phá.

## 11.5 Ủy quyền cho Subagent để Bảo vệ Context

```
Main agent: "Investigate dependencies of the payments module"
  -> Subagent (Explore): reads 15 files, traces imports
  -> Returns: "Payments depends on AuthService, OrderModel, and the external
PaymentGateway API"

Main agent: keeps one line in context instead of 15 files
```

**Lớp context riêng biệt:** Trong các hệ thống multi-agent, mỗi subagent hoạt động trong một ngân sách context giới hạn—nó chỉ nhận thông tin cần thiết cho tác vụ của mình. Coordinator đóng vai trò như một lớp context riêng biệt: nó tổng hợp các đầu ra của subagent, lưu trữ trạng thái toàn cục, và phân bổ context. Điều này ngăn chặn "rò rỉ context", khi một agent chiếm dụng cửa sổ context bằng thông tin không liên quan đến các agent khác.

**Ngân sách context bị ràng buộc cho subagent:**

- Gửi context tối thiểu: một tác vụ cụ thể + dữ liệu cần thiết
- Hướng dẫn subagent trả về kết quả có cấu trúc, không phải các bãi dữ liệu thô
- Dùng `allowedTools` để giới hạn bộ tool của subagent—ít tool hơn nghĩa là ít phân tâm hơn và chi phí context thấp hơn

## 11.6 Lưu trữ Trạng thái có Cấu trúc (để phục hồi sau sự cố)

Mỗi agent xuất trạng thái của nó ra một vị trí đã biết:

```
// agent-state/web-search-agent.json
{
  "status": "completed",
  "queries_executed": ["AI music 2024", "AI music composition"],
  "results_count": 12,
  "key_findings": [...],
  "coverage": ["music composition", "music production"],
  "gaps": ["music distribution", "music licensing"]
}
```

Coordinator nạp một manifest khi tiếp tục:

```
// agent-state/manifest.json
{
  "web-search": "completed",
  "doc-analysis": "in_progress",
  "synthesis": "not_started"
}
```

---

# Chương 12: Bảo toàn Provenance

## 12.1 Vấn đề Mất Quy kết Nguồn

Khi tóm tắt kết quả từ nhiều nguồn, liên kết “tuyên bố → nguồn” có thể bị mất:

```
Bad: "The AI music market is estimated at $3.2B." (No source, no year.)

Good:
{
  "claim": "The AI music market is estimated at $3.2B.",
  "source_url": "https://example.com/report",
  "source_name": "Global AI Music Report 2024",
  "publication_date": "2024-06-15",
  "confidence": 0.9
}
```

## 12.2 Xử lý Dữ liệu Xung đột

Khi hai nguồn cung cấp các giá trị khác nhau:

```
{
  "claim": "Share of AI-generated music on streaming platforms",
  "values": [
    {
      "value": "12%",
      "source": "Spotify Annual Report 2024",
      "date": "2024-03",
      "methodology": "Automated classification"
    },
    {
      "value": "8%",
      "source": "Music Industry Association Survey",
      "date": "2024-07",
      "methodology": "Survey of 500 labels"
    }
  ],
  "conflict_detected": true,
  "possible_explanation": "Difference in methodology and time period"
}
```

Đừng tùy tiện chọn một giá trị. Hãy giữ lại cả hai kèm theo quy kết nguồn và để coordinator quyết định.

## 12.3 Bao gồm Ngày tháng để Diễn giải Đúng

Nếu không có ngày tháng, các khác biệt theo thời gian có thể bị hiểu nhầm thành mâu thuẫn:

```
Bad: "Source A says 10%, source B says 15%. Contradiction."
Good: "Source A (2023) says 10%, source B (2024) says 15%. Likely +5% growth over a year."
```

## 12.4 Trình bày theo Loại Nội dung

Đừng ép mọi thứ vào một định dạng duy nhất:

- Dữ liệu tài chính -> bảng
- Tin tức và phân tích -> văn xuôi
- Phát hiện kỹ thuật -> danh sách có cấu trúc
- Chuỗi thời gian -> sắp xếp theo trình tự thời gian

---

# Chương 13: Các Tool Tích hợp sẵn của Claude Code

## 13.1 Tham chiếu Lựa chọn Tool

| Tác vụ | Tool | Ví dụ |
|---|---|---|
| Tìm tệp theo tên/mẫu | **Glob** | `**/*.test.tsx`, `src/components/**/*.ts` |
| Tìm kiếm bên trong tệp | **Grep** | Tên hàm, thông báo lỗi, import |
| Đọc toàn bộ một tệp | **Read** | Nạp một tệp để phân tích |
| Ghi một tệp mới | **Write** | Tạo một tệp từ đầu |
| Chỉnh sửa chính xác một tệp có sẵn | **Edit** | Thay thế một đoạn cụ thể qua khớp văn bản duy nhất |
| Chạy một lệnh shell | **Bash** | git, npm, chạy test, build |

## 13.2 Chiến lược Điều tra Tăng dần

Đừng đọc tất cả các tệp cùng lúc. Hãy xây dựng hiểu biết một cách tăng dần:

```
1. Grep: find entry points (function definition, export)
2. Read: read the found files
3. Grep: find usages (import, calls)
4. Read: read consumer files
5. Repeat until you have a complete picture
```

## 13.3 Phương án Dự phòng: Read + Write thay vì Edit

Khi Edit thất bại do khớp văn bản không duy nhất:

1. Read — nạp toàn bộ nội dung tệp
2. Sửa đổi nội dung bằng chương trình
3. Write — ghi phiên bản đã cập nhật

---

# PHẦN II: GHI CHÚ THEO LĨNH VỰC THI

---

# Lĩnh vực 1: Kiến trúc và Điều phối Agent (27%)

## 1.1 Thiết kế Agentic Loop để Thực thi Tác vụ Tự chủ

### Kiến thức trọng tâm:

- Vòng đời của agent loop: gửi một yêu cầu Claude, kiểm tra `stop_reason` (`"tool_use"` so với `"end_turn"`), thực thi tool, trả kết quả về cho vòng lặp kế tiếp
- Kết quả của tool được nối thêm vào lịch sử hội thoại để mô hình có thể quyết định hành động tiếp theo
- Ra quyết định do mô hình điều khiển (Claude chọn tool kế tiếp) so với cây quyết định cứng (hard-coded)

### Kỹ năng trọng tâm:

- Điều khiển luồng: tiếp tục vòng lặp khi `stop_reason = "tool_use"` và dừng khi gặp `"end_turn"`
- Nối kết quả của tool vào context giữa các vòng lặp
- Các anti-pattern cần tránh: phân tích văn bản của assistant để xác định hoàn thành, dùng giới hạn số vòng lặp tùy ý làm cơ chế dừng chính

## 1.2 Điều phối Hệ thống Multi-agent (Coordinator–Subagent)

### Kiến thức trọng tâm:

- Kiến trúc hub-and-spoke: coordinator nắm toàn bộ giao tiếp giữa các agent, xử lý lỗi và định tuyến
- Các subagent hoạt động với context tách biệt—chúng không tự động kế thừa lịch sử của coordinator
- Trách nhiệm của coordinator: phân rã tác vụ, ủy thác, tổng hợp kết quả, lựa chọn subagent một cách động
- Rủi ro coordinator phân rã quá hẹp

### Kỹ năng trọng tâm:

- Chia phạm vi nghiên cứu giữa các subagent để giảm thiểu trùng lặp
- Triển khai vòng lặp tinh chỉnh lặp lại (coordinator đánh giá phần tổng hợp và định tuyến lại tác vụ)
- Định tuyến toàn bộ giao tiếp qua coordinator để có khả năng quan sát

## 1.3 Cấu hình các Lời gọi Subagent, Truyền Context và Spawning

### Kiến thức trọng tâm:

- Tool `Task` spawn các subagent; `allowedTools` của coordinator phải bao gồm `"Task"`
- Context của subagent phải được đưa vào prompt một cách tường minh; subagent không kế thừa context của agent cha
- Cấu hình `AgentDefinition`: mô tả, system prompt, ràng buộc tool
- Quản lý session qua `fork_session` để khám phá các phương án thay thế

### Kỹ năng trọng tâm:

- Đưa toàn bộ đầu ra từ các agent trước vào prompt của subagent
- Dùng định dạng có cấu trúc để tách dữ liệu khỏi metadata khi truyền context
- Spawn các subagent song song qua nhiều lời gọi `Task` trong một lượt duy nhất của coordinator
- Viết prompt cho coordinator theo hướng mục tiêu và tiêu chí chất lượng thay vì hướng dẫn từng bước

## 1.4 Triển khai Workflow Nhiều bước với các Mẫu Thực thi và Bàn giao

### Kiến thức trọng tâm:

- Sự khác biệt giữa **thực thi theo lập trình** (hook, điều kiện tiên quyết) và **hướng dẫn bằng prompt** để sắp xếp thứ tự một workflow
- Khi bạn cần các đảm bảo deterministic (ví dụ, xác minh danh tính trước các thao tác tài chính), chỉ dùng prompt là không đủ
- Giao thức bàn giao có cấu trúc trong quá trình escalation (ID khách hàng, lý do, hành động được khuyến nghị)

### Kỹ năng trọng tâm:

- Điều kiện tiên quyết theo lập trình để chặn các lời gọi phía sau cho đến khi các bước trước hoàn tất (ví dụ, chặn `process_refund` cho đến khi `get_customer` trả về một ID đã được xác minh)
- Phân rã các yêu cầu khách hàng nhiều khía cạnh thành các mục riêng biệt
- Tạo các bản tóm tắt có cấu trúc khi escalation cho con người

## 1.5 Agent SDK Hook để Chặn Lời gọi Tool và Chuẩn hóa Dữ liệu

### Kiến thức trọng tâm:

- Các mẫu hook (ví dụ, `PostToolUse`) để chặn kết quả của tool trước khi mô hình tiêu thụ chúng
- Các hook chặn lời gọi đi ra để thực thi quy tắc tuân thủ (ví dụ, chặn hoàn tiền vượt ngưỡng)
- Hook cung cấp **đảm bảo deterministic** so với hướng dẫn bằng prompt vốn chỉ cung cấp **mức tuân thủ xác suất**

### Kỹ năng trọng tâm:

- Các hook `PostToolUse` để chuẩn hóa định dạng dữ liệu (Unix timestamp, ISO 8601, mã trạng thái dạng số)
- Các hook chặn để ngăn các hành động vi phạm chính sách kèm chuyển hướng sang escalation
- Chọn hook thay vì prompt khi quy tắc nghiệp vụ đòi hỏi sự tuân thủ được đảm bảo

## 1.6 Chiến lược Phân rã Tác vụ cho Workflow Phức tạp

### Kiến thức trọng tâm:

- **Pipeline cố định** (prompt chaining) so với **phân rã thích ứng động** dựa trên kết quả trung gian
- Prompt chaining: các bước tuần tự (phân tích từng file riêng biệt, sau đó chạy một lượt tích hợp)
- Kế hoạch điều tra thích ứng sinh ra các tác vụ con dựa trên những gì đã được phát hiện

### Kỹ năng trọng tâm:

- Dùng prompt chaining cho các đánh giá nhiều khía cạnh có thể dự đoán được; dùng phân rã động cho các điều tra mở
- Chia các đợt review code lớn thành phân tích từng file cộng thêm một lượt tích hợp xuyên file riêng biệt
- Phân rã các tác vụ mở: lập bản đồ cấu trúc trước, sau đó xây dựng một kế hoạch được ưu tiên hóa

## 1.7 Trạng thái Session, Resuming và Forking

### Kiến thức trọng tâm:

- `--resume <session-name>` để tiếp tục các session có tên
- `fork_session` để tạo các nhánh điều tra độc lập từ context dùng chung
- Tầm quan trọng của việc thông báo cho agent về các thay đổi file khi resume session
- Một session mới với bản tóm tắt có cấu trúc có thể đáng tin cậy hơn so với việc resume với kết quả đã lỗi thời

### Kỹ năng trọng tâm:

- Dùng `--resume` để tiếp tục các session điều tra có tên
- Dùng `fork_session` để so sánh các phương án song song
- Chọn giữa resume (context vẫn còn hiện hành) so với khởi tạo một session mới (kết quả đã lỗi thời)

---

# Lĩnh vực 2: Thiết kế Tool và Tích hợp MCP (18%)

## 2.1 Thiết kế Giao diện Tool với Mô tả Rõ ràng

### Kiến thức trọng tâm:

- Mô tả tool là **cơ chế chính** mà một LLM dùng để chọn tool; mô tả tối thiểu dẫn đến việc lựa chọn không đáng tin cậy
- Tầm quan trọng của việc đưa vào định dạng đầu vào, các truy vấn ví dụ, các edge case và ranh giới khả dụng
- Mô tả mơ hồ hoặc chồng lấp gây ra định tuyến sai
- Cách diễn đạt trong system prompt có thể tạo ra những liên tưởng ngoài ý muốn với các tool

### Kỹ năng trọng tâm:

- Viết mô tả phân biệt rõ ràng mỗi tool với các phương án tương tự
- Đổi tên tool để loại bỏ sự chồng lấp về chức năng (ví dụ, `analyze_content` -> `extract_web_results`)
- Chia các tool đa năng thành các tool chuyên biệt với hợp đồng đầu vào/đầu ra rõ ràng

## 2.2 Triển khai Phản hồi Lỗi có Cấu trúc cho Tool MCP

### Kiến thức trọng tâm:

- Cờ `isError` trong phản hồi tool MCP
- Sự khác biệt giữa **lỗi tạm thời** (timeout), **lỗi validation** (đầu vào sai), **lỗi nghiệp vụ** (vi phạm chính sách), và **lỗi truy cập/quyền hạn**
- Lỗi chung chung ("Operation failed") ngăn cản các quyết định phục hồi đúng đắn
- Sự khác biệt giữa lỗi có thể retry và không thể retry

### Kỹ năng trọng tâm:

- Trả về metadata có cấu trúc như `errorCategory` (transient/validation/permission), `isRetryable`, và một thông điệp mà con người đọc được
- Dùng `retryable: false` cho các vi phạm quy tắc nghiệp vụ kèm giải thích rõ ràng hướng đến người dùng
- Thực hiện phục hồi cục bộ bên trong subagent đối với các lỗi tạm thời; chỉ lan truyền những lỗi mà chúng không thể giải quyết
- Phân biệt lỗi truy cập (quyết định retry) với kết quả rỗng hợp lệ (không có kết quả khớp)

## 2.3 Phân bổ Tool Giữa các Agent và Cấu hình `tool_choice`

### Kiến thức trọng tâm:

- Quá nhiều tool cho mỗi agent (ví dụ, 18 thay vì 4–5) **làm giảm** độ tin cậy của việc lựa chọn tool
- Các agent có tool ngoài chuyên môn của mình có xu hướng sử dụng chúng sai cách
- Truy cập tool có phạm vi: chỉ những tool liên quan đến vai trò cộng thêm một tập hạn chế các tiện ích liên vai trò
- `tool_choice`: `"auto"`, `"any"`, và lựa chọn tool bắt buộc (`{"type": "tool", "name": "..."}`)

### Kỹ năng trọng tâm:

- Giới hạn bộ tool của mỗi subagent vào những gì liên quan đến vai trò của nó
- Thay thế các tool tổng quát bằng các phương án bị ràng buộc (ví dụ, `fetch_url` -> `load_document`)
- Dùng `tool_choice: "any"` để đảm bảo có một lời gọi tool thay vì một câu trả lời văn bản
- Bắt buộc một tool cụ thể để đảm bảo thứ tự thực thi

## 2.4 Tích hợp MCP Server vào Claude Code và Workflow của Agent

### Kiến thức trọng tâm:

- Phạm vi MCP server: dự án (`.mcp.json`) cho các nhóm so với người dùng (`~/.claude.json`) cho thử nghiệm
- Thay thế biến môi trường trong `.mcp.json` (ví dụ, `${GITHUB_TOKEN}`) để quản lý secret
- Tool từ tất cả MCP server đã kết nối được khám phá khi kết nối và khả dụng đồng thời
- MCP resource như "danh mục nội dung" (bản tóm tắt tác vụ, schema cơ sở dữ liệu) để giảm các lời gọi tool mang tính khám phá

### Kỹ năng trọng tâm:

- Cấu hình các MCP server dùng chung trong `.mcp.json` của dự án với token dựa trên biến môi trường
- Giữ các server cá nhân/thử nghiệm trong `~/.claude.json`
- Ưu tiên các MCP server cộng đồng hơn các server tùy chỉnh cho các tích hợp tiêu chuẩn

## 2.5 Lựa chọn và Áp dụng các Tool Tích hợp sẵn (Read, Write, Edit, Bash, Grep, Glob)

### Kiến thức trọng tâm:

- **Grep**: tìm kiếm trong nội dung file (tên hàm, thông điệp lỗi, import)
- **Glob**: tìm file theo mẫu tên/phần mở rộng
- **Read/Write**: thao tác toàn bộ file; **Edit**: thay đổi chính xác qua các khớp văn bản duy nhất
- Nếu Edit thất bại do các khớp không duy nhất, hãy chuyển sang dùng Read + Write

### Kỹ năng trọng tâm:

- Dùng Grep để tìm kiếm nội dung và Glob để khám phá file theo mẫu
- Xây dựng sự hiểu biết một cách tăng dần: Grep các điểm vào, sau đó Read để truy vết các luồng
- Truy vết việc sử dụng hàm qua các module bao bọc (wrapper)

---

# Lĩnh vực 3: Cấu hình Claude Code và Workflow (20%)

## 3.1 Cấu hình CLAUDE.md với Phân cấp, Phạm vi và Tổ chức Mô-đun hóa

### Kiến thức trọng tâm:

- Phân cấp CLAUDE.md: người dùng (`~/.claude/CLAUDE.md`), dự án (`.claude/CLAUDE.md` hoặc `CLAUDE.md` ở thư mục gốc), và cấp thư mục (CLAUDE.md trong các thư mục con)
- Cài đặt cấp người dùng chỉ áp dụng cho một người dùng và không được chia sẻ qua VCS
- Cú pháp `@path` để tham chiếu các file bên ngoài (ví dụ, `@./standards/coding-style.md`) nhằm mô-đun hóa CLAUDE.md
- Thư mục `.claude/rules/` chứa các file quy tắc tập trung theo chủ đề thay cho một CLAUDE.md đơn khối

### Kỹ năng trọng tâm:

- Chẩn đoán các vấn đề phân cấp (một thành viên mới của nhóm bỏ lỡ hướng dẫn vì chúng ở cấp người dùng thay vì cấp dự án)
- Dùng `@path` (ví dụ, `@./standards/testing.md`) để đưa các tiêu chuẩn vào một cách chọn lọc trong CLAUDE.md của từng package
- Chia một CLAUDE.md lớn thành nhiều file `.claude/rules/` (testing.md, api-conventions.md, deployment.md)

## 3.2 Tạo và Cấu hình các Slash Command và Skill Tùy chỉnh

### Kiến thức trọng tâm:

- **Command của dự án** trong `.claude/commands/` (chia sẻ qua VCS) so với **command của người dùng** trong `~/.claude/commands/`
- Skill trong `.claude/skills/` với frontmatter `SKILL.md`: `context: fork`, `allowed-tools`, `argument-hint`
- `context: fork` chạy skill trong một context subagent tách biệt để nó không làm ô nhiễm session chính
- Các biến thể skill cá nhân có thể nằm trong `~/.claude/skills/` dưới các tên khác nhau

### Kỹ năng trọng tâm:

- Lưu các slash command của dự án trong `.claude/commands/` để cả nhóm đều có chúng
- Dùng `context: fork` để cô lập các skill có đầu ra dài dòng
- Dùng `allowed-tools` để hạn chế những tool mà một skill có thể sử dụng
- Dùng `argument-hint` để nhắc nhở lập trình viên về các tham số bắt buộc

## 3.3 Sử dụng Quy tắc Theo đường dẫn để Tải Quy ước có Điều kiện

### Kiến thức trọng tâm:

- Các file `.claude/rules/` có thể bao gồm frontmatter YAML `paths` để kích hoạt quy tắc dựa trên các mẫu glob
- Các quy tắc theo phạm vi đường dẫn **chỉ** tải khi chỉnh sửa các file khớp, giúp tiết kiệm context và token
- Các quy tắc đường dẫn dựa trên glob có thể được ưu tiên hơn CLAUDE.md cấp thư mục khi các quy ước áp dụng xuyên nhiều thư mục (ví dụ, các bài test)

### Kỹ năng trọng tâm:

- Tạo các file `.claude/rules/` với `paths: ["terraform/**/*"]` để chỉ tải khi làm việc trên các file khớp
- Dùng các mẫu glob (`**/*.test.tsx`) để áp dụng quy ước theo loại file bất kể vị trí
- Ưu tiên các quy tắc theo đường dẫn hơn CLAUDE.md cấp thư mục khi các quy ước trải rộng khắp codebase

## 3.4 Quyết định Khi nào Dùng Planning Mode so với Thực thi Trực tiếp

### Kiến thức trọng tâm:

- **Planning mode**: cho các tác vụ phức tạp với thay đổi lớn, nhiều phương án khả thi và các quyết định kiến trúc
- **Thực thi trực tiếp**: cho các thay đổi đơn giản, đã được hiểu rõ (ví dụ, thêm một validation duy nhất)
- Planning mode cho phép khám phá codebase một cách an toàn trước khi thực hiện thay đổi
- Subagent Explore cô lập đầu ra khám phá dài dòng

### Kỹ năng trọng tâm:

- Dùng planning mode cho các tác vụ có hệ quả về kiến trúc (microservice, các migration chạm tới hơn 45 file)
- Dùng thực thi trực tiếp cho các đợt sửa lỗi có stack trace rõ ràng và một file duy nhất
- Dùng subagent Explore để ngăn việc cạn kiệt context window trong các tác vụ nhiều giai đoạn
- Kết hợp các cách tiếp cận: lập kế hoạch để khám phá, sau đó thực thi để triển khai

## 3.5 Tinh chỉnh Lặp lại để Cải thiện Tiệm tiến

### Kiến thức trọng tâm:

- Các ví dụ đầu vào/đầu ra cụ thể là cách hiệu quả nhất để truyền đạt kỳ vọng
- **Lặp lại theo hướng test**: viết test trước, sau đó lặp lại dựa trên các thất bại
- Mẫu “phỏng vấn”: Claude đặt câu hỏi để làm lộ ra các cân nhắc thiết kế không hiển nhiên

- Khi nào nên cung cấp tất cả vấn đề trong một thông điệp (phụ thuộc lẫn nhau) so với cung cấp tuần tự (độc lập)

### Kỹ năng trọng tâm:

- Cung cấp 2–3 ví dụ đầu vào/đầu ra cụ thể để làm rõ các yêu cầu biến đổi
- Xây dựng các bộ test với hành vi kỳ vọng, các edge case và các yêu cầu hiệu năng trước khi triển khai
- Dùng mẫu phỏng vấn để làm lộ ra các khía cạnh thiết kế (vô hiệu hóa cache, các chế độ lỗi)
- Cung cấp các test case cụ thể với đầu vào mẫu và đầu ra kỳ vọng cho các edge case

## 3.6 Tích hợp Claude Code vào Pipeline CI/CD

### Kiến thức trọng tâm:

- Cờ `-p` (hoặc `--print`) cho chế độ non-interactive trong các pipeline tự động hóa
- `--output-format json` và `--json-schema` cho structured output trong CI
- CLAUDE.md cung cấp context của dự án (tiêu chuẩn test, tiêu chí review) cho Claude Code được kích hoạt bởi CI
- **Cô lập context của session**: chính session đã sinh ra code thì kém hiệu quả hơn trong việc review nó so với một instance độc lập

### Kỹ năng trọng tâm:

- Chạy Claude Code trong CI với `-p` để tránh bị treo khi chờ đầu vào tương tác
- Dùng `--output-format json` + `--json-schema` cho các kết quả có cấu trúc (ví dụ, các comment inline trên pull request)
- Đưa kết quả review trước đó vào khi chạy lại sau các commit mới (chỉ báo cáo các vấn đề mới/chưa được sửa)
- Tài liệu hóa tiêu chuẩn test và các fixture sẵn có trong CLAUDE.md để cải thiện chất lượng sinh test
- Đưa các file test hiện có vào context khi sinh test mới nhằm tránh trùng lặp và giữ phong cách nhất quán

---

# Lĩnh vực 4: Prompt Engineering và Structured Output (20%)

## 4.1 Thiết kế Prompt với Tiêu chí Tường minh để Cải thiện Độ chính xác

### Kiến thức trọng tâm:

- Tiêu chí tường minh hiệu quả hơn các hướng dẫn mơ hồ (ví dụ, “chỉ gắn cờ comment khi chúng mâu thuẫn với code” so với “kiểm tra độ chính xác của comment”)

- Hướng dẫn chung chung kiểu “hãy thận trọng hơn” hoạt động kém hơn so với các tiêu chí phân loại cụ thể
- Tác động của false positive lên niềm tin của lập trình viên: tỷ lệ false-positive cao ở một số hạng mục làm xói mòn niềm tin vào các hạng mục chính xác

### Kỹ năng trọng tâm:

- Định nghĩa tiêu chí review: cái gì cần báo cáo (bug, bảo mật) so với cái gì cần bỏ qua (lỗi style nhỏ)
- Tạm thời vô hiệu hóa các hạng mục có tỷ lệ false-positive cao
- Định nghĩa tiêu chí mức độ nghiêm trọng tường minh với ví dụ code cho mỗi cấp độ

## 4.2 Sử dụng Few-shot Prompting để Cải thiện Tính Nhất quán của Đầu ra

### Kiến thức trọng tâm:

- Các ví dụ few-shot là phương pháp hiệu quả nhất để tạo ra đầu ra được định dạng nhất quán và có tính hành động
- Few-shot có thể minh họa cách xử lý các trường hợp mơ hồ (lựa chọn tool, các lỗ hổng trong độ phủ test)
- Few-shot giúp mô hình tổng quát hóa sang các mẫu mới thay vì chỉ lặp lại các giá trị mặc định
- Few-shot có thể giảm hiện tượng ảo giác (hallucination) trong các tác vụ trích xuất

### Kỹ năng trọng tâm:

- Cung cấp 2–4 ví dụ có mục tiêu cho các kịch bản mơ hồ kèm lý do
- Đưa vào các ví dụ few-shot minh họa định dạng đầu ra (vị trí, vấn đề, mức độ nghiêm trọng, đề xuất sửa)
- Cung cấp các ví dụ phân biệt các mẫu code chấp nhận được với các vấn đề thực sự
- Cung cấp các ví dụ về việc trích xuất đúng từ các tài liệu có cấu trúc khác nhau

## 4.3 Thực thi Structured Output với `tool_use` và JSON Schema

### Kiến thức trọng tâm:

- `tool_use` với JSON Schema là cách đáng tin cậy nhất để đảm bảo đầu ra tuân thủ schema và loại bỏ các lỗi cú pháp JSON
- Với `tool_choice: "auto"` mô hình có thể trả về văn bản; với `"any"` nó phải gọi một tool; lựa chọn bắt buộc thì chọn một tool cụ thể
- JSON Schema nghiêm ngặt loại bỏ các lỗi cú pháp nhưng không ngăn được các lỗi ngữ nghĩa (tổng không khớp; giá trị nằm sai trường)
- Thiết kế schema: trường bắt buộc so với tùy chọn; enum với “other” cộng một chuỗi chi tiết để có khả năng mở rộng

### Kỹ năng trọng tâm:

- Định nghĩa các tool trích xuất với JSON Schema và parse dữ liệu từ kết quả `tool_use`
- Dùng `tool_choice: "any"` để đảm bảo structured output khi tồn tại nhiều schema
- Bắt buộc một lời gọi tool cụ thể: `tool_choice: {"type": "tool", "name": "extract_metadata"}`
- Để các trường ở dạng tùy chọn/nullable khi nguồn có thể không chứa thông tin nhằm tránh bịa ra giá trị
- Dùng các giá trị enum như `"unclear"` và `"other"` cộng các trường chi tiết để phân loại có khả năng mở rộng

## 4.4 Triển khai Validation, Retry và Vòng lặp Phản hồi cho Chất lượng Trích xuất

### Kiến thức trọng tâm:

- Retry kèm phản hồi lỗi: đưa các lỗi validation cụ thể vào prompt retry để hướng dẫn sửa lỗi
- Retry không hiệu quả khi thông tin đơn giản là không có trong nguồn
- Thiết kế vòng lặp phản hồi: theo dõi mẫu đã kích hoạt một phát hiện (`detected_pattern`)
- Lỗi ngữ nghĩa (tổng không khớp) so với lỗi cú pháp (được xử lý bởi `tool_use`)

### Kỹ năng trọng tâm:

- Các prompt tiếp nối với tài liệu gốc, một bản trích xuất sai, và các lỗi validation cụ thể
- Nhận diện khi nào retry sẽ không hiệu quả (thông tin cần thiết chỉ nằm trong một tài liệu bên ngoài)
- Đưa các trường `detected_pattern` vào các phát hiện để phân tích false positive
- Thiết kế tự sửa lỗi bằng cách trích xuất cả `calculated_total` lẫn `stated_total` để phát hiện sai lệch

## 4.5 Thiết kế Chiến lược Batch Processing Hiệu quả

### Kiến thức trọng tâm:

- Message Batches API: tiết kiệm 50%, cửa sổ xử lý lên đến 24 giờ, không có đảm bảo SLA về độ trễ
- Batch processing phù hợp cho các tác vụ không chặn (báo cáo qua đêm, audit) và không phù hợp cho các tác vụ chặn (kiểm tra trước khi merge)
- Batch API không hỗ trợ gọi tool nhiều lượt trong một yêu cầu duy nhất
- Các trường `custom_id` để tương quan request/response trong các batch

### Kỹ năng trọng tâm:

- Dùng API đồng bộ cho các kiểm tra chặn; dùng Batch API cho các khối lượng công việc qua đêm/hàng tuần
- Lập kế hoạch nhịp độ gửi batch dựa trên nhu cầu SLA (ví dụ, cửa sổ 4 giờ cho một đảm bảo 30 giờ với 24 giờ xử lý)
- Xử lý các thất bại bằng cách gửi lại chỉ những tài liệu thất bại (được nhận diện bởi `custom_id`)
- Lặp lại trên prompt bằng một mẫu trước khi chạy xử lý quy mô lớn

## 4.6 Thiết kế Kiến trúc Review Đa instance và Đa lượt

### Kiến thức trọng tâm:

- Hạn chế của tự review: mô hình giữ lại context suy luận của nó và ít có khả năng thách thức các quyết định của chính mình
- Các instance review độc lập (không có context sinh ra) tốt hơn trong việc tìm các vấn đề tinh tế
- Review đa lượt: phân tích cục bộ theo từng file cộng thêm một lượt tích hợp xuyên file để tránh sự pha loãng chú ý

### Kỹ năng trọng tâm:

- Dùng một instance Claude thứ hai độc lập để review các thay đổi mà không có context sinh ra
- Chia các đợt review nhiều file thành các lượt theo từng file cộng thêm các lượt tích hợp để phân tích luồng dữ liệu xuyên file
- Dùng các lượt xác minh với độ tự tin tự đánh giá để định tuyến các đợt review một cách được hiệu chỉnh

---

# Lĩnh vực 5: Quản lý Context và Độ tin cậy (15%)

## 5.1 Quản lý Context Hội thoại để Bảo toàn Thông tin Quan trọng

### Kiến thức trọng tâm:

- Rủi ro của việc tóm tắt tiệm tiến: các giá trị số, phần trăm và ngày tháng bị cô đọng thành các bản tóm tắt mơ hồ
- Hiệu ứng lost-in-the-middle: các mô hình xử lý đáng tin cậy phần đầu và phần cuối của các đầu vào dài, nhưng có thể bỏ sót các phát hiện ở phần giữa
- Đầu ra của tool có thể tích lũy trong context không tương xứng với mức độ liên quan (hơn 40 trường trong khi chỉ cần 5)
- Tầm quan trọng của việc gửi toàn bộ lịch sử hội thoại trong các yêu cầu API tiếp theo

### Kỹ năng trọng tâm:

- Trích xuất các sự kiện giao dịch vào một khối “case facts” bền vững nằm ngoài lịch sử đã được tóm tắt
- Cắt gọn các đầu ra tool dài dòng xuống còn các trường liên quan
- Đặt các phát hiện then chốt ở đầu của dữ liệu tổng hợp với các tiêu đề mục tường minh
- Yêu cầu các subagent đưa metadata (ngày tháng, nguồn) vào các đầu ra có cấu trúc

## 5.2 Thiết kế các Mẫu Escalation Hiệu quả và Giải quyết Sự Mơ hồ

### Kiến thức trọng tâm:

- Các tác nhân kích hoạt escalation phù hợp: yêu cầu rõ ràng cần con người, lỗ hổng/ngoại lệ chính sách, không có khả năng tiến triển
- Escalation tức thì (yêu cầu rõ ràng) so với cố gắng giải quyết (trong phạm vi của agent)
- Phân tích cảm xúc và việc mô hình tự đánh giá độ tự tin là các đại lượng thay thế không đáng tin cậy cho độ phức tạp của vụ việc
- Nhiều khách hàng trùng khớp đòi hỏi phải hỏi thêm định danh, không phải đoán theo heuristic

### Kỹ năng trọng tâm:

- Tiêu chí escalation tường minh với các ví dụ few-shot trong system prompt
- Thực thi ngay các yêu cầu rõ ràng cần con người mà không cần điều tra thêm
- Escalation khi chính sách mơ hồ hoặc im lặng đối với một yêu cầu cụ thể
- Hỏi thêm định danh khi kết quả tool chứa nhiều khớp

## 5.3 Triển khai Chiến lược Lan truyền Lỗi trong Hệ thống Multi-agent

### Kiến thức trọng tâm:

- Context lỗi có cấu trúc (loại thất bại, truy vấn, kết quả một phần, các phương án thay thế) cho phép coordinator phục hồi thông minh hơn
- Phân biệt lỗi truy cập (timeout đòi hỏi một quyết định retry) với kết quả rỗng hợp lệ (không có khớp)
- Trạng thái lỗi chung chung (“search unavailable”) che giấu context có giá trị khỏi coordinator
- Cả việc lặng lẽ chặn lỗi lẫn việc hủy bỏ toàn bộ workflow chỉ vì một thất bại đều là anti-pattern

### Kỹ năng trọng tâm:

- Trả về context lỗi có cấu trúc: loại thất bại, những gì đã được thử, kết quả một phần, các phương án thay thế khả dĩ
- Phân biệt lỗi truy cập với kết quả rỗng hợp lệ
- Thực hiện phục hồi cục bộ trong các subagent đối với các lỗi tạm thời; chỉ lan truyền các lỗi không thể phục hồi kèm kết quả một phần
- Chú thích độ phủ trong phần tổng hợp: cái gì được hỗ trợ tốt so với chỗ nào còn lỗ hổng

## 5.4 Quản lý Context Hiệu quả khi Điều tra các Codebase Lớn

### Kiến thức trọng tâm:

- Suy giảm context trong các session dài: mô hình bắt đầu tạo ra các câu trả lời không ổn định và dẫn chiếu đến “các mẫu điển hình” thay vì các lớp cụ thể

- Các file scratchpad bảo toàn các phát hiện then chốt qua các ranh giới context
- Ủy thác cho các subagent cô lập đầu ra khám phá dài dòng
- Lưu trữ trạng thái có cấu trúc cho phép phục hồi sau khi sự cố (crash recovery)

### Kỹ năng trọng tâm:

- Spawn các subagent cho các câu hỏi cụ thể trong khi giữ việc điều phối cấp cao trong agent chính
- Dùng các file scratchpad để lưu các phát hiện then chốt và dẫn chiếu chúng sau này
- Tóm tắt các phát hiện then chốt trước khi spawn các subagent của giai đoạn kế tiếp
- Dùng `/compact` để giảm mức sử dụng context trong các đợt điều tra dài

## 5.5 Thiết kế Workflow với Giám sát của Con người và Hiệu chỉnh Độ tự tin

### Kiến thức trọng tâm:

- Các chỉ số tổng hợp (ví dụ, độ chính xác tổng thể 97%) có thể che giấu hiệu năng kém trên các loại tài liệu hoặc trường cụ thể
- Lấy mẫu ngẫu nhiên phân tầng đo lường tỷ lệ lỗi trong các đợt trích xuất có độ tự tin cao
- Hiệu chỉnh độ tự tin ở cấp trường bằng cách dùng các tập validation đã gán nhãn
- Xác nhận độ chính xác theo loại tài liệu và phân đoạn trường trước khi tự động hóa

### Kỹ năng trọng tâm:

- Triển khai lấy mẫu ngẫu nhiên phân tầng để phát hiện các mẫu lỗi mới
- Phân tích độ chính xác theo loại tài liệu và trường để xác nhận hiệu năng ổn định
- Xuất ra các điểm độ tự tin ở cấp trường và hiệu chỉnh ngưỡng review bằng dữ liệu đã gán nhãn
- Định tuyến các đợt trích xuất có độ tự tin thấp hoặc nguồn mơ hồ sang con người để review

## 5.6 Bảo toàn Provenance và Xử lý Tính Bất định trong Tổng hợp Đa nguồn

### Kiến thức trọng tâm:

- Quy gán bị mất trong quá trình tóm tắt nếu không bảo toàn các ánh xạ “claim → source”
- Các ánh xạ có cấu trúc phải được bảo toàn trong quá trình tổng hợp
- Xử lý các thống kê mâu thuẫn bằng cách chú thích các xung đột kèm quy gán thay vì tùy tiện chọn một giá trị
- Đưa vào ngày xuất bản/thu thập để tránh việc đọc nhầm các khác biệt theo thời gian thành mâu thuẫn

### Kỹ năng trọng tâm:

- Yêu cầu các subagent xuất ra các ánh xạ “claim → source” (URL, tên tài liệu, trích dẫn)
- Cấu trúc các báo cáo để tách các phát hiện ổn định khỏi các phát hiện đang tranh chấp
- Bảo toàn các giá trị mâu thuẫn kèm chú thích và chuyển chúng cho coordinator để đối soát

- Đưa vào ngày xuất bản để diễn giải theo thời gian đúng đắn
- Render nội dung theo loại: dữ liệu tài chính dạng bảng, tin tức dạng văn xuôi, các phát hiện kỹ thuật dạng danh sách có cấu trúc

---

# Ví dụ câu hỏi thi kèm giải thích

## Câu hỏi 1 (Kịch bản: Customer Support Agent)

**Tình huống:** Dữ liệu cho thấy trong 12% trường hợp, agent bỏ qua `get_customer` và gọi `lookup_order` chỉ dựa trên tên của khách hàng, dẫn đến hoàn tiền sai.

**Thay đổi nào hiệu quả nhất?**

- A) Thêm một điều kiện tiên quyết bằng code để chặn `lookup_order` và `process_refund` cho đến khi lấy được ID từ `get_customer` **[ĐÁP ÁN ĐÚNG]**
- B) Cải thiện system prompt
- C) Thêm các ví dụ few-shot
- D) Triển khai một bộ phân loại định tuyến (routing classifier)

**Vì sao A:** Khi logic nghiệp vụ quan trọng đòi hỏi một trình tự tool cụ thể, phần mềm mang lại **các đảm bảo deterministic** mà các cách tiếp cận dựa trên prompt (B, C) không thể có được. D giải quyết vấn đề về tính sẵn sàng, không phải về thứ tự gọi tool.

---

## Câu hỏi 2 (Kịch bản: Customer Support Agent)

**Tình huống:** Agent thường gọi `get_customer` thay vì `lookup_order` cho các câu hỏi liên quan đến đơn hàng. Mô tả của các tool sơ sài và giống nhau.

**Bước đầu tiên là gì?**

- A) Các ví dụ few-shot
- B) Mở rộng mô tả của mỗi tool với định dạng đầu vào, ví dụ và ranh giới áp dụng **[ĐÁP ÁN ĐÚNG]**
- C) Thêm một lớp định tuyến (routing layer)
- D) Gộp các tool lại

**Vì sao B:** Mô tả tool là cơ chế lựa chọn chính của mô hình. Đây là cách sửa tốn ít công sức nhất nhưng tác động lớn nhất. A bổ sung token mà không giải quyết nguyên nhân gốc rễ. C là thiết kế quá mức (overengineering). D đòi hỏi nhiều công sức hơn mức cần thiết.

---

## Câu hỏi 3 (Kịch bản: Customer Support Agent)

**Tình huống:** Agent chỉ giải quyết được 55% sự vụ trong khi mục tiêu là 80%. Nó escalation cả những trường hợp đơn giản và lại cố tự xử lý các ngoại lệ chính sách phức tạp.

**Bạn cải thiện việc hiệu chỉnh (calibration) như thế nào?**

- A) Thêm các tiêu chí escalation rõ ràng kèm các ví dụ few-shot **[ĐÁP ÁN ĐÚNG]**
- B) Để mô hình tự đánh giá độ tin cậy (1–10) với escalation tự động
- C) Một bộ phân loại riêng được huấn luyện trên dữ liệu lịch sử
- D) Phân tích cảm xúc (sentiment analysis)

**Vì sao A:** Cách này giải quyết trực tiếp nguyên nhân gốc rễ—ranh giới quyết định không rõ ràng. B không đáng tin cậy (mô hình có thể sai mà vẫn tự tin). C là thiết kế quá mức. D giải quyết một vấn đề khác (tâm trạng != độ phức tạp).

---

## Câu hỏi 4 (Kịch bản: Code Generation with Claude Code)

**Tình huống:** Bạn cần một lệnh `/review` tùy chỉnh cho việc review code theo chuẩn, có sẵn cho cả nhóm khi họ clone repository.

**Bạn nên tạo file lệnh ở đâu?**

- A) `.claude/commands/` trong repository của dự án **[ĐÁP ÁN ĐÚNG]**
- B) `~/.claude/commands/`
- C) `CLAUDE.md` ở thư mục gốc
- D) `.claude/config.json`

**Vì sao A:** Các lệnh cấp dự án được lưu trong `.claude/commands/` được quản lý phiên bản và tự động có sẵn cho mọi người. B dành cho các lệnh cá nhân. C dành cho hướng dẫn, không phải định nghĩa lệnh. D không tồn tại.

---

## Câu hỏi 5 (Kịch bản: Code Generation with Claude Code)

**Tình huống:** Bạn cần tái cấu trúc một monolith thành microservices (hàng chục file, các quyết định về ranh giới dịch vụ).

**Bạn nên dùng cách tiếp cận nào?**

- A) Planning mode: khám phá codebase, hiểu các phụ thuộc, thiết kế cách tiếp cận **[ĐÁP ÁN ĐÚNG]**
- B) Thực thi trực tiếp theo từng phần nhỏ tăng dần
- C) Thực thi trực tiếp với hướng dẫn chi tiết được đưa ra ngay từ đầu
- D) Thực thi trực tiếp và chuyển sang planning khi gặp khó

**Vì sao A:** Planning mode được thiết kế cho các thay đổi lớn, nhiều cách tiếp cận khả thi và các quyết định kiến trúc. B có nguy cơ phải làm lại tốn kém. C giả định bạn đã biết sẵn cấu trúc. D mang tính phản ứng bị động.

---

## Câu hỏi 6 (Kịch bản: Code Generation with Claude Code)

**Tình huống:** Một codebase có các quy ước khác nhau giữa các khu vực (React, API, database). Các test được đặt cùng chỗ với code. Bạn muốn các quy ước được áp dụng tự động.

**Bạn nên dùng cách tiếp cận nào?**

- A) Các file `.claude/rules/` với YAML frontmatter và glob pattern **[ĐÁP ÁN ĐÚNG]**
- B) Đặt mọi thứ trong CLAUDE.md ở thư mục gốc
- C) Skills trong `.claude/skills/`
- D) CLAUDE.md trong mọi thư mục

**Vì sao A:** `.claude/rules/` với glob pattern (ví dụ `**/*.test.tsx`) cho phép áp dụng quy ước tự động dựa trên đường dẫn file—lý tưởng cho các test nằm rải rác khắp codebase. B dựa vào suy luận của mô hình. C mang tính thủ công/theo yêu cầu. D không hoạt động tốt khi các file liên quan nằm ở nhiều thư mục.

---

## Câu hỏi 7 (Kịch bản: Multi-agent Research System)

**Tình huống:** Hệ thống nghiên cứu chủ đề "tác động của AI lên các ngành sáng tạo", nhưng báo cáo chỉ đề cập đến nghệ thuật thị giác. Coordinator đã phân rã chủ đề thành: "AI trong nghệ thuật số", "AI trong thiết kế đồ họa", "AI trong nhiếp ảnh".

**Nguyên nhân là gì?**

- A) Agent tổng hợp (synthesis) không phát hiện ra các lỗ hổng
- B) Coordinator đã phân rã nhiệm vụ quá hẹp **[ĐÁP ÁN ĐÚNG]**
- C) Agent tìm kiếm web không tìm đủ kỹ
- D) Agent phân tích tài liệu đã lọc bỏ các nguồn không thuộc nghệ thuật thị giác

**Vì sao B:** Log cho thấy coordinator chỉ phân rã "các ngành sáng tạo" thành các chủ đề con về thị giác, bỏ sót hoàn toàn âm nhạc, văn học và điện ảnh. Các subagent đã thực thi đúng—vấn đề nằm ở những gì chúng được giao.

---

## Câu hỏi 8 (Kịch bản: Multi-agent Research System)

**Tình huống:** Một subagent tìm kiếm web bị timeout khi nghiên cứu một chủ đề phức tạp. Bạn cần thiết kế cách thông tin lỗi được truyền ngược về coordinator.

**Cách tiếp cận lan truyền lỗi nào hỗ trợ tốt nhất cho việc khôi phục một cách thông minh?**

- A) Trả về context lỗi có cấu trúc cho coordinator: loại lỗi, query, kết quả một phần và các phương án thay thế **[ĐÁP ÁN ĐÚNG]**
- B) Triển khai retry tự động với exponential backoff bên trong subagent, rồi trả về một trạng thái chung chung "search unavailable"
- C) Bắt timeout bên trong subagent và trả về một tập kết quả rỗng được đánh dấu là thành công

- D) Lan truyền ngoại lệ timeout lên một handler ở cấp cao nhất để chấm dứt toàn bộ workflow

**Vì sao A:** Context lỗi có cấu trúc cung cấp cho coordinator những gì nó cần để quyết định có nên retry với một query đã chỉnh sửa, thử một cách tiếp cận thay thế, hay tiếp tục với kết quả một phần. B che giấu context phía sau một trạng thái chung chung. C che đậy thất bại như thể thành công. D hủy bỏ toàn bộ workflow một cách không cần thiết.

---

## Câu hỏi 9 (Kịch bản: Multi-agent Research System)

**Tình huống:** Agent tổng hợp (synthesis) thường cần xác minh các tuyên bố cụ thể trong khi hợp nhất kết quả. Hiện tại, khi cần xác minh, agent tổng hợp trao quyền điều khiển lại cho coordinator, coordinator gọi agent tìm kiếm web rồi chạy lại quá trình tổng hợp với kết quả mới. Điều này thêm 2–3 lượt round trip cho mỗi nhiệm vụ và làm tăng độ trễ thêm 40%. Đánh giá của bạn cho thấy 85% các lần kiểm tra này là kiểm tra dữ kiện đơn giản (ngày tháng, tên, số liệu thống kê), trong khi 15% đòi hỏi điều tra sâu hơn.

**Bạn giảm chi phí phụ trội (overhead) như thế nào trong khi vẫn duy trì độ tin cậy?**

- A) Cấp cho agent tổng hợp một tool `verify_fact` giới hạn cho các kiểm tra đơn giản, và tiếp tục định tuyến việc xác minh phức tạp qua coordinator **[ĐÁP ÁN ĐÚNG]**
- B) Gom tất cả các nhu cầu xác minh vào một batch và trả về cho coordinator ở cuối
- C) Cấp cho agent tổng hợp toàn quyền truy cập tất cả các tool tìm kiếm web
- D) Chủ động cache thêm context xung quanh mỗi nguồn

**Vì sao A:** Cách này áp dụng nguyên tắc least privilege: agent tổng hợp nhận được đúng những gì nó cần cho trường hợp phổ biến chiếm 85% (kiểm tra dữ kiện đơn giản) trong khi vẫn giữ lại đường đi qua trung gian coordinator cho các điều tra phức tạp. B đưa vào các phụ thuộc gây nghẽn (các bước tổng hợp sau có thể phụ thuộc vào các dữ kiện đã xác minh trước đó). C phá vỡ sự tách bạch trách nhiệm. D dựa vào việc cache mang tính phỏng đoán, không thể dự đoán nhu cầu một cách đáng tin cậy.

---

## Câu hỏi 10 (Kịch bản: Claude Code for CI)

**Tình huống:** Một pipeline chạy `claude "Analyze this pull request for security issues"`, nhưng treo lại để chờ đầu vào tương tác.

**Cách tiếp cận đúng là gì?**

- A) Dùng cờ `-p`: `claude -p "Analyze this pull request for security issues"` **[ĐÁP ÁN ĐÚNG]**
- B) Đặt `CLAUDE_HEADLESS=true`
- C) Chuyển hướng stdin từ `/dev/null`
- D) Dùng `--batch`

**Vì sao A:** `-p` (hay `--print`) là cách được tài liệu hóa để chạy Claude Code ở chế độ non-interactive. Nó xử lý prompt, in ra stdout, rồi thoát. Các phương án khác hoặc là tính năng không tồn tại hoặc là các cách lách kiểu Unix.

---

## Câu hỏi 11 (Kịch bản: Claude Code for CI)

**Tình huống:** Nhóm muốn giảm chi phí API cho việc phân tích tự động. Hiện Claude phục vụ hai workflow theo thời gian thực: (1) một bước kiểm tra chặn (blocking) trước khi merge, phải hoàn thành trước khi lập trình viên có thể merge một PR, và (2) một báo cáo về tech-debt được tạo qua đêm để xem vào buổi sáng. Một quản lý đề xuất chuyển cả hai sang Message Batches API để tiết kiệm 50%.

**Bạn nên đánh giá đề xuất này như thế nào?**

- A) Chỉ dùng xử lý batch cho các báo cáo tech-debt; giữ các lệnh gọi thời gian thực cho bước kiểm tra trước khi merge **[ĐÁP ÁN ĐÚNG]**
- B) Chuyển cả hai workflow sang xử lý batch và poll để chờ hoàn thành
- C) Giữ các lệnh gọi thời gian thực cho cả hai để tránh các vấn đề về thứ tự trong kết quả batch
- D) Chuyển cả hai sang xử lý batch với một fallback về thời gian thực nếu một batch mất quá lâu

**Vì sao A:** Message Batches API tiết kiệm 50%, nhưng thời gian xử lý có thể lên tới 24 giờ mà không có SLA về độ trễ được đảm bảo. Điều đó khiến nó không phù hợp cho các bước kiểm tra chặn trước khi merge nơi lập trình viên đang chờ, nhưng lý tưởng cho các tải batch chạy qua đêm như báo cáo tech-debt.

---

## Câu hỏi 12 (Kịch bản: Multi-file Code Review)

**Tình huống:** Một pull request thay đổi 14 file trong một module theo dõi kho hàng. Việc review tất cả các file trong một lượt duy nhất cho ra kết quả không nhất quán: nhận xét chi tiết cho một số file nhưng hời hợt cho các file khác, bỏ sót những lỗi hiển nhiên, và phản hồi mâu thuẫn (một pattern bị gắn cờ là có vấn đề ở một file nhưng lại được chấp thuận với code y hệt ở một file khác).

**Bạn nên tái cấu trúc việc review như thế nào?**

- A) Chia thành các lượt tập trung: phân tích từng file riêng lẻ cho các vấn đề cục bộ, rồi chạy một lượt tích hợp riêng cho các luồng dữ liệu liên file **[ĐÁP ÁN ĐÚNG]**
- B) Yêu cầu lập trình viên chia các PR lớn thành các lần nộp gồm 3–4 file
- C) Chuyển sang một mô hình cấp cao hơn với context window lớn hơn để review tất cả 14 file trong một lượt
- D) Chạy ba lượt review toàn bộ PR độc lập và chỉ báo cáo những vấn đề được tìm thấy ở ít nhất hai lượt

**Vì sao A:** Các lượt tập trung giải quyết trực tiếp nguyên nhân gốc rễ—sự pha loãng chú ý khi xử lý nhiều file cùng lúc. Phân tích theo từng file đảm bảo độ sâu nhất quán, và một lượt tích hợp riêng bắt được các vấn đề liên file. B đẩy gánh nặng sang lập trình viên mà không cải thiện hệ thống. C là một quan niệm sai lầm: context lớn hơn không khắc phục chất lượng chú ý. D che lấp các lỗi thật vì đòi hỏi sự đồng thuận trên các phát hiện vốn không nhất quán.

---

# Bài kiểm tra thực hành

> 60 câu hỏi trải dài trên 4 kịch bản. Định dạng và độ khó khớp với bài thi thật.
>
> Ngoài ra, bạn có thể luyện tập các câu hỏi này trong một file HTML giống như bài thi: Practical Test (EN)

## Kịch bản: Hệ thống Nghiên cứu Đa tác nhân

---

## Câu hỏi 1 (Kịch bản: Hệ thống Nghiên cứu Đa tác nhân)

**Tình huống:** Một agent phân tích tài liệu phát hiện rằng hai nguồn đáng tin cậy chứa những số liệu thống kê mâu thuẫn trực tiếp cho một chỉ số quan trọng: một báo cáo của chính phủ nêu mức tăng trưởng 40%, trong khi một phân tích ngành nêu 12%. Cả hai nguồn đều trông đáng tin cậy, và sự chênh lệch này có thể ảnh hưởng đáng kể đến các kết luận nghiên cứu. Agent phân tích tài liệu nên xử lý tình huống này hiệu quả nhất như thế nào?

**Cách tiếp cận nào là hiệu quả nhất?**

- A) Áp dụng các quy tắc kinh nghiệm về độ tin cậy để chọn ra con số có khả năng đúng nhất, hoàn thành phân tích với giá trị đó, và thêm một chú thích đề cập đến sự chênh lệch.
- B) Đưa cả hai con số vào kết quả phân tích mà không đánh dấu chúng là mâu thuẫn, để agent tổng hợp quyết định dùng con số nào dựa trên bối cảnh rộng hơn.
- C) Dừng phân tích và lập tức escalation lên coordinator, yêu cầu nó quyết định nguồn nào có thẩm quyền hơn trước khi tiếp tục.
- D) Hoàn thành phân tích với cả hai con số, chú thích rõ ràng sự mâu thuẫn kèm theo nguồn trích dẫn, và để coordinator quyết định cách dung hòa dữ liệu trước khi chuyển sang tổng hợp. **[ĐÁP ÁN ĐÚNG]**

**Vì sao D:** Cách tiếp cận này bảo toàn sự phân tách trách nhiệm: agent phân tích hoàn thành công việc cốt lõi của nó mà không bị chặn, giữ lại cả hai giá trị mâu thuẫn kèm trích dẫn rõ ràng, và chuyển đúng việc dung hòa cho coordinator, vốn có bối cảnh rộng hơn.

---

## Câu hỏi 2 (Kịch bản: Hệ thống Nghiên cứu Đa tác nhân)

**Tình huống:** Các agent web-search và document-analysis đã hoàn thành nhiệm vụ của chúng và trả kết quả về cho coordinator. Bước tiếp theo để tạo một báo cáo nghiên cứu tích hợp là gì?

**Bước tiếp theo nào là phù hợp nhất?**

- A) Mỗi agent gửi kết quả của nó trực tiếp đến agent viết báo cáo, bỏ qua coordinator.
- B) Agent phân tích tài liệu yêu cầu kết quả web-search và hợp nhất chúng trong nội bộ.
- C) Coordinator chuyển cả hai bộ kết quả cho agent tổng hợp để tích hợp thống nhất. **[ĐÁP ÁN ĐÚNG]**
- D) Coordinator nối các kết quả thô từ cả hai agent và trả về như kết quả cuối cùng.

**Vì sao C:** Trong kiến trúc coordinator–subagent, coordinator chuyển cả hai bộ kết quả cho agent tổng hợp để tích hợp tập trung, bảo toàn quyền kiểm soát và đảm bảo việc hợp nhất chất lượng cao.

---

## Câu hỏi 3 (Kịch bản: Hệ thống Nghiên cứu Đa tác nhân)

**Tình huống:** Một subagent phân tích tài liệu thường xuyên thất bại khi xử lý các file PDF: một số file có các phần bị hỏng gây ra ngoại lệ phân tích cú pháp, một số khác được bảo vệ bằng mật khẩu, và đôi khi thư viện phân tích cú pháp bị treo trên các file lớn. Hiện tại, bất kỳ ngoại lệ nào cũng ngay lập tức kết thúc subagent và trả về lỗi cho coordinator, vốn phải quyết định nên retry, bỏ qua, hay để cả nhiệm vụ thất bại. Điều này khiến coordinator phải tham gia quá mức vào việc xử lý lỗi thông thường. Cải tiến kiến trúc nào là hiệu quả nhất?

**Cải tiến nào là hiệu quả nhất?**

- A) Tạo một agent xử lý lỗi chuyên dụng giám sát mọi thất bại qua một hàng đợi dùng chung và quyết định các hành động khôi phục, gửi lệnh khởi động lại trực tiếp đến các subagent.
- B) Cấu hình subagent để luôn trả về kết quả một phần với trạng thái thành công, nhúng chi tiết lỗi vào metadata; coordinator xem mọi phản hồi là thành công.
- C) Buộc coordinator xác thực tất cả tài liệu trước khi gửi chúng đến subagent, từ chối các tài liệu có thể gây thất bại.
- D) Triển khai khôi phục cục bộ trong subagent cho các thất bại tạm thời và chỉ escalation lên coordinator những lỗi mà nó không thể tự giải quyết, kèm theo các bước đã thử và kết quả một phần. **[ĐÁP ÁN ĐÚNG]**

**Vì sao D:** Hãy xử lý lỗi ở cấp thấp nhất có khả năng giải quyết chúng. Khôi phục cục bộ làm giảm khối lượng công việc của coordinator trong khi vẫn escalation những vấn đề thực sự không thể khôi phục kèm theo đầy đủ bối cảnh và tiến độ một phần.

---

## Câu hỏi 4 (Kịch bản: Hệ thống Nghiên cứu Đa tác nhân)

**Tình huống:** Sau khi chạy hệ thống trên chủ đề “tác động của AI lên các ngành công nghiệp sáng tạo,” bạn quan sát thấy mọi subagent đều hoàn thành thành công: agent web-search tìm thấy các bài viết liên quan, agent phân tích tài liệu tóm tắt chúng chính xác, và agent tổng hợp tạo ra văn bản mạch lạc. Tuy nhiên, các báo cáo cuối cùng chỉ bao phủ nghệ thuật thị giác và hoàn toàn bỏ sót âm nhạc, văn học, và điện ảnh. Trong log của coordinator, bạn thấy nó đã phân rã chủ đề thành ba subtask: “AI trong nghệ thuật số,” “AI trong thiết kế đồ họa,” và “AI trong nhiếp ảnh.” Nguyên nhân gốc rễ có khả năng nhất là gì?

**Nguyên nhân gốc rễ có khả năng nhất là gì?**

- A) Agent tổng hợp thiếu hướng dẫn để phát hiện các lỗ hổng về độ bao phủ.
- B) Agent phân tích tài liệu lọc bỏ các nguồn phi thị giác do tiêu chí liên quan quá nghiêm ngặt.
- C) Việc phân rã nhiệm vụ của coordinator quá hẹp, giao cho các subagent công việc không bao phủ tất cả các lĩnh vực liên quan. **[ĐÁP ÁN ĐÚNG]**
- D) Các truy vấn của agent web-search là không đủ và nên được mở rộng để bao phủ nhiều ngành hơn.

**Vì sao C:** Coordinator chỉ phân rã một chủ đề rộng thành các subtask về nghệ thuật thị giác, bỏ sót hoàn toàn âm nhạc, văn học, và điện ảnh. Vì các subagent đã thực hiện đúng phần việc được giao, việc phân rã quá hẹp là nguyên nhân gốc rễ hiển nhiên.

---

## Câu hỏi 5 (Kịch bản: Hệ thống Nghiên cứu Đa tác nhân)

**Tình huống:** Subagent web-search chỉ trả về kết quả cho 3 trên 5 danh mục nguồn được yêu cầu (các trang web đối thủ và báo cáo ngành thành công, nhưng kho lưu trữ tin tức và các nguồn cấp mạng xã hội bị timeout). Subagent phân tích tài liệu xử lý thành công tất cả các tài liệu được cung cấp. Subagent tổng hợp phải tạo ra một bản tóm tắt từ các đầu vào thượng nguồn có chất lượng hỗn hợp. Chiến lược lan truyền lỗi nào là hiệu quả nhất?

**Chiến lược lan truyền lỗi nào là hiệu quả nhất?**

- A) Tiếp tục tổng hợp chỉ dùng các nguồn thành công và tạo ra một kết quả mà không đề cập đến dữ liệu nào không khả dụng.
- B) Subagent tổng hợp trả về lỗi cho coordinator, kích hoạt một lần retry toàn bộ hoặc làm cho nhiệm vụ thất bại do dữ liệu không đầy đủ.
- C) Subagent tổng hợp yêu cầu coordinator retry các nguồn bị timeout với timeout dài hơn trước khi bắt đầu tổng hợp.
- D) Cấu trúc kết quả tổng hợp với các chú thích về độ bao phủ cho biết những kết luận nào được hỗ trợ tốt và những lỗ hổng tồn tại ở đâu do các nguồn không khả dụng. **[ĐÁP ÁN ĐÚNG]**

**Vì sao D:** Các chú thích về độ bao phủ hiện thực hóa sự suy giảm nhẹ nhàng (graceful degradation) kèm tính minh bạch, bảo toàn giá trị từ phần việc đã hoàn thành trong khi lan truyền sự không chắc chắn để cho phép các quyết định có hiểu biết về độ tin cậy.

---

## Câu hỏi 6 (Kịch bản: Hệ thống Nghiên cứu Đa tác nhân)

**Tình huống:** Subagent phân tích tài liệu gặp một file PDF bị hỏng mà nó không thể phân tích cú pháp. Khi thiết kế việc xử lý lỗi của hệ thống, cách hiệu quả nhất để xử lý thất bại này là gì?

**Cách tiếp cận nào là hiệu quả nhất?**

- A) Trả về lỗi kèm bối cảnh cho agent coordinator, cho phép nó quyết định cách tiến hành. **[ĐÁP ÁN ĐÚNG]**
- B) Âm thầm bỏ qua tài liệu bị hỏng và tiếp tục xử lý các file còn lại để tránh làm gián đoạn quy trình.
- C) Tự động retry phân tích cú pháp tài liệu ba lần với exponential backoff trước khi báo cáo thất bại.
- D) Ném ra một ngoại lệ chấm dứt toàn bộ quy trình nghiên cứu.

**Vì sao A:** Trả về lỗi kèm bối cảnh cho coordinator là cách tiếp cận hiệu quả nhất vì nó cho phép coordinator đưa ra quyết định có hiểu biết—bỏ qua file, thử một phương pháp phân tích cú pháp thay thế, hoặc thông báo cho người dùng—trong khi vẫn duy trì khả năng quan sát thất bại đó.

---

## Câu hỏi 7 (Kịch bản: Hệ thống Nghiên cứu Đa tác nhân)

**Tình huống:** Log sản xuất cho thấy một mẫu hình dai dẳng: các yêu cầu như "phân tích báo cáo quý đã tải lên" được định tuyến đến agent web-search 45% số lần thay vì đến agent phân tích tài liệu. Xem xét các định nghĩa tool, bạn thấy rằng agent web-search có một tool `analyze_content` được mô tả là "phân tích nội dung và trích xuất thông tin then chốt," trong khi agent phân tích tài liệu có một tool `analyze_document` được mô tả là "phân tích tài liệu và trích xuất thông tin then chốt." Bạn nên khắc phục vấn đề định tuyến sai như thế nào?

**Bạn nên khắc phục vấn đề định tuyến sai như thế nào?**

- A) Thêm một bộ phân loại tiền định tuyến phát hiện liệu người dùng đang nhắc đến các file đã tải lên hay nội dung web trước khi coordinator quyết định việc giao việc.
- B) Đổi tên tool web-search thành `extract_web_results` và cập nhật mô tả của nó thành "xử lý và trả về thông tin được truy xuất từ web search và URL." **[ĐÁP ÁN ĐÚNG]**
- C) Thêm các ví dụ few-shot vào prompt của coordinator cho thấy việc định tuyến đúng: "Người dùng tải lên một báo cáo quý → agent phân tích tài liệu" và "Người dùng hỏi về một trang web → agent web-search."
- D) Mở rộng mô tả tool phân tích tài liệu với các ví dụ sử dụng như "Dùng cho các PDF, file Word, và bảng tính đã tải lên," giữ nguyên tool web-search.

**Vì sao B:** Đổi tên tool web-search thành `extract_web_results` và cập nhật mô tả của nó để tham chiếu rõ ràng đến web search và URL sẽ trực tiếp loại bỏ nguyên nhân gốc rễ bằng cách triệt tiêu sự chồng lấn ngữ nghĩa giữa hai tên tool và mô tả. Điều này khiến mục đích của mỗi tool trở nên rõ ràng, cho phép coordinator phân biệt một cách đáng tin cậy giữa phân tích tài liệu và web search.

---

## Câu hỏi 8 (Kịch bản: Hệ thống Nghiên cứu Đa tác nhân)

**Tình huống:** Một đồng nghiệp đề xuất rằng agent phân tích tài liệu nên gửi kết quả của nó trực tiếp đến agent tổng hợp, bỏ qua coordinator. Lợi thế chính của việc giữ coordinator làm trung tâm điều phối cho mọi giao tiếp giữa các subagent là gì?

**Lợi thế chính của việc giữ coordinator làm trung tâm điều phối là gì?**

- A) Coordinator có thể quan sát mọi tương tác, xử lý lỗi một cách thống nhất, và quyết định thông tin nào mỗi subagent nên nhận. **[ĐÁP ÁN ĐÚNG]**
- B) Coordinator gộp nhiều yêu cầu đến các subagent thành batch, giảm tổng số lần gọi API và độ trễ tổng thể.
- C) Định tuyến qua coordinator cho phép logic retry tự động mà các cuộc gọi trực tiếp giữa agent không thể hỗ trợ.
- D) Các subagent dùng bộ nhớ tách biệt, và giao tiếp trực tiếp sẽ đòi hỏi serialization phức tạp mà chỉ coordinator mới thực hiện được.

**Vì sao A:** Mẫu hình coordinator cung cấp khả năng quan sát tập trung vào mọi tương tác, xử lý lỗi thống nhất trên toàn hệ thống, và kiểm soát chi tiết về thông tin nào mỗi subagent nhận được—đây là những lợi thế chính của một cấu trúc giao tiếp hình sao.

---

## Câu hỏi 9 (Kịch bản: Hệ thống Nghiên cứu Đa tác nhân)

**Tình huống:** Subagent web-search bị timeout khi nghiên cứu một chủ đề phức tạp. Bạn cần thiết kế cách thông tin về thất bại này được trả về cho coordinator. Cách tiếp cận lan truyền lỗi nào cho phép khôi phục thông minh tốt nhất?

**Cách tiếp cận lan truyền lỗi nào cho phép khôi phục thông minh tốt nhất?**

- A) Trả về bối cảnh lỗi có cấu trúc cho coordinator bao gồm loại thất bại, truy vấn đã thực thi, mọi kết quả một phần, và các cách tiếp cận thay thế tiềm năng. **[ĐÁP ÁN ĐÚNG]**
- B) Bắt timeout bên trong subagent và trả về một tập kết quả rỗng được đánh dấu là thành công.
- C) Triển khai các lần retry tự động với exponential-backoff bên trong subagent, chỉ trả về một trạng thái chung chung "search unavailable" sau khi đã hết lượt retry.
- D) Lan truyền ngoại lệ timeout trực tiếp đến bộ xử lý cấp cao nhất, chấm dứt toàn bộ quy trình nghiên cứu.

**Vì sao A:** Trả về bối cảnh lỗi có cấu trúc—bao gồm loại thất bại, truy vấn đã thực thi, kết quả một phần, và các cách tiếp cận thay thế—cung cấp cho coordinator mọi thứ cần thiết để đưa ra các quyết định khôi phục thông minh (ví dụ: retry với một truy vấn đã sửa đổi hoặc tiếp tục với kết quả một phần). Nó bảo toàn tối đa bối cảnh cho việc ra quyết định có hiểu biết ở cấp điều phối.

---

## Câu hỏi 10 (Kịch bản: Hệ thống Nghiên cứu Đa tác nhân)

**Tình huống:** Trong thiết kế hệ thống của bạn, bạn đã cấp cho agent phân tích tài liệu quyền truy cập một tool đa năng `fetch_url` để nó có thể tải xuống tài liệu theo URL. Log sản xuất cho thấy agent này giờ thường xuyên tải xuống các trang kết quả của công cụ tìm kiếm để thực hiện web search tùy ý—hành vi lẽ ra phải được định tuyến qua agent web-search—gây ra các kết quả không nhất quán. Cách khắc phục nào là hiệu quả nhất?

**Cách khắc phục nào là hiệu quả nhất?**

- A) Thay thế `fetch_url` bằng một tool `load_document` xác thực rằng các URL trỏ đến các định dạng tài liệu. **[ĐÁP ÁN ĐÚNG]**
- B) Loại bỏ `fetch_url` khỏi agent phân tích tài liệu và định tuyến mọi việc tải URL qua coordinator đến agent web-search.
- C) Triển khai bộ lọc chặn các lệnh gọi `fetch_url` đến các tên miền công cụ tìm kiếm đã biết trong khi cho phép các URL khác.
- D) Thêm hướng dẫn vào prompt của agent phân tích tài liệu rằng `fetch_url` chỉ nên được dùng để tải xuống các URL tài liệu, không phải để tìm kiếm.

**Vì sao A:** Thay thế một tool đa năng bằng một tool chuyên cho tài liệu, vốn xác thực các URL theo các định dạng tài liệu, sẽ khắc phục nguyên nhân gốc rễ bằng cách ràng buộc khả năng ở cấp giao diện. Điều này tuân theo nguyên tắc least privilege, khiến hành vi tìm kiếm không mong muốn trở nên bất khả thi thay vì chỉ bị khuyến cáo.

---

## Câu hỏi 11 (Kịch bản: Hệ thống Nghiên cứu Đa tác nhân)

**Tình huống:** Trong khi nghiên cứu một chủ đề rộng, bạn quan sát thấy rằng agent web-search và agent phân tích tài liệu điều tra cùng các chủ đề con, dẫn đến sự trùng lặp đáng kể trong kết quả của chúng. Mức sử dụng token gần như tăng gấp đôi mà không có sự gia tăng tương ứng về độ rộng hay độ sâu của nghiên cứu. Cách hiệu quả nhất để giải quyết điều này là gì?

**Cách hiệu quả nhất để giải quyết điều này là gì?**

- A) Cho phép cả hai agent hoàn thành song song, sau đó để coordinator khử trùng lặp các kết quả chồng lấn trước khi chuyển chúng cho agent tổng hợp.
- B) Coordinator phân chia rõ ràng không gian nghiên cứu trước khi giao việc, gán cho mỗi agent các chủ đề con hoặc loại nguồn riêng biệt. **[ĐÁP ÁN ĐÚNG]**
- C) Triển khai một cơ chế trạng thái dùng chung trong đó các agent ghi lại lĩnh vực trọng tâm hiện tại của chúng để các agent khác có thể tránh trùng lặp một cách động trong khi thực thi.
- D) Chuyển sang thực thi tuần tự, trong đó phân tích tài liệu chỉ chạy sau khi web search hoàn thành, dùng kết quả web-search làm bối cảnh để tránh trùng lặp.

**Vì sao B:** Việc để coordinator phân chia rõ ràng không gian nghiên cứu trước khi giao việc là hiệu quả nhất vì nó giải quyết nguyên nhân gốc rễ—ranh giới nhiệm vụ không rõ ràng—trước khi bất kỳ công việc nào bắt đầu. Nó bảo toàn tính song song trong khi ngăn ngừa nỗ lực bị trùng lặp và token bị lãng phí.

---

## Câu hỏi 12 (Kịch bản: Hệ thống Nghiên cứu Đa tác nhân)

**Tình huống:** Trong quá trình nghiên cứu, subagent web-search truy vấn ba danh mục nguồn với các kết quả khác nhau: các cơ sở dữ liệu học thuật trả về 15 bài báo liên quan, các báo cáo ngành trả về “0 results,” và các cơ sở dữ liệu bằng sáng chế trả về “Connection timeout.” Khi thiết kế việc lan truyền lỗi đến coordinator, cách tiếp cận nào cho phép các quyết định khôi phục tốt nhất?

**Cách tiếp cận nào cho phép các quyết định khôi phục tốt nhất?**

- A) Tổng hợp các kết quả thành một chỉ số phần-trăm-thành-công duy nhất (ví dụ: “67% độ bao phủ nguồn”) với các log chi tiết có sẵn theo yêu cầu.
- B) Báo cáo cả “timeout” lẫn “0 results” như những thất bại đòi hỏi coordinator can thiệp.
- C) Retry các thất bại tạm thời trong nội bộ và chỉ báo cáo các lỗi dai dẳng.
- D) Phân biệt các thất bại truy cập (timeout) vốn đòi hỏi một quyết định retry với các kết quả rỗng hợp lệ (“0 results”) vốn đại diện cho các truy vấn thành công. **[ĐÁP ÁN ĐÚNG]**

**Vì sao D:** Một timeout (thất bại truy cập) và “0 results” (kết quả rỗng hợp lệ) là những kết cục khác nhau về mặt ngữ nghĩa đòi hỏi các phản ứng khác nhau. Việc phân biệt chúng cho phép coordinator retry cơ sở dữ liệu bằng sáng chế trong khi chấp nhận “0 results” của các báo cáo ngành như một phát hiện hợp lệ, mang tính thông tin.

---

## Câu hỏi 13 (Kịch bản: Hệ thống Nghiên cứu Đa tác nhân)

**Tình huống:** Giám sát sản xuất cho thấy chất lượng tổng hợp không nhất quán. Khi các kết quả tổng hợp là ~75K token, agent tổng hợp trích dẫn một cách đáng tin cậy thông tin từ 15K token đầu tiên (tiêu đề/đoạn trích web-search) và 10K token cuối cùng (kết luận phân tích tài liệu), nhưng thường bỏ sót các phát hiện then chốt ở 50K token ở giữa—ngay cả khi chúng trực tiếp trả lời câu hỏi nghiên cứu. Bạn nên tái cấu trúc đầu vào tổng hợp như thế nào?

**Bạn nên tái cấu trúc đầu vào tổng hợp như thế nào?**

- A) Tóm tắt mọi kết quả của subagent xuống dưới 20K token trước khi tổng hợp để giữ nội dung trong phạm vi xử lý đáng tin cậy của mô hình.
- B) Stream kết quả của subagent đến agent tổng hợp một cách tăng dần, xử lý kết quả web-search trước cho đến hết, sau đó thêm các kết quả phân tích tài liệu.
- C) Đặt một bản tóm tắt các phát hiện then chốt ở đầu đầu vào tổng hợp và tổ chức các kết quả chi tiết với các tiêu đề mục rõ ràng để dễ điều hướng hơn. **[ĐÁP ÁN ĐÚNG]**
- D) Triển khai một cơ chế luân phiên đảo thứ tự xuất hiện đầu tiên của kết quả từ mỗi subagent qua các nhiệm vụ nghiên cứu để đảm bảo cả hai nguồn đều được vị trí đầu trang ngang nhau theo thời gian.

**Vì sao C:** Đặt một bản tóm tắt các phát hiện then chốt ở đầu tận dụng hiệu ứng vị trí đầu (primacy) để thông tin quan trọng nằm ở vị trí được xử lý đáng tin cậy nhất. Việc thêm các tiêu đề mục rõ ràng xuyên suốt giúp mô hình điều hướng và chú ý đến nội dung ở giữa đầu vào, trực tiếp giảm thiểu hiện tượng “lạc giữa chừng” (lost in the middle).

---

## Câu hỏi 14 (Kịch bản: Hệ thống Nghiên cứu Đa tác nhân)

**Tình huống:** Trong quá trình kiểm thử, kết quả kết hợp của agent web-search (85K token bao gồm nội dung trang) và agent phân tích tài liệu (70K token bao gồm các chuỗi suy luận) tổng cộng là 155K token, nhưng agent tổng hợp hoạt động tốt nhất với các đầu vào dưới 50K token. Giải pháp nào là hiệu quả nhất?

**Giải pháp nào là hiệu quả nhất?**

- A) Sửa đổi các agent thượng nguồn để trả về dữ liệu có cấu trúc (các sự kiện then chốt, trích dẫn, điểm số liên quan) thay vì nội dung và lập luận dài dòng. **[ĐÁP ÁN ĐÚNG]**
- B) Thêm một agent tóm tắt trung gian cô đọng các phát hiện trước khi chuyển chúng đến tổng hợp.
- C) Để agent tổng hợp xử lý các phát hiện theo các batch tuần tự, duy trì trạng thái giữa các lần gọi.
- D) Lưu trữ các phát hiện trong một cơ sở dữ liệu vector và cấp cho agent tổng hợp các tool tìm kiếm để truy vấn trong khi nó làm việc.

**Vì sao A:** Sửa đổi các agent thượng nguồn để trả về dữ liệu có cấu trúc khắc phục nguyên nhân gốc rễ bằng cách giảm khối lượng token tại nguồn trong khi bảo toàn thông tin thiết yếu. Nó tránh việc chuyển đi nội dung trang cồng kềnh và các vết suy luận làm phình token mà không cải thiện bước tổng hợp.

---

## Câu hỏi 15 (Kịch bản: Hệ thống Nghiên cứu Đa tác nhân)

**Tình huống:** Trong quá trình kiểm thử, bạn quan sát thấy rằng agent tổng hợp thường cần xác minh các khẳng định cụ thể trong khi hợp nhất kết quả. Hiện tại, khi cần xác minh, agent tổng hợp trả quyền kiểm soát về cho coordinator, vốn gọi agent web-search rồi gọi lại agent tổng hợp với các kết quả. Điều này thêm 2–3 vòng lặp phụ cho mỗi nhiệm vụ và tăng độ trễ thêm 40%. Đánh giá của bạn cho thấy 85% các lần xác minh này là kiểm tra sự kiện đơn giản (ngày tháng, tên, số liệu thống kê) và 15% đòi hỏi nghiên cứu sâu hơn. Cách tiếp cận nào giảm chi phí phụ trội hiệu quả nhất trong khi vẫn bảo toàn độ tin cậy của hệ thống?

**Cách tiếp cận nào là hiệu quả nhất?**

- A) Cấp cho agent tổng hợp quyền truy cập tất cả các tool web-search để nó có thể xử lý bất kỳ nhu cầu xác minh nào trực tiếp mà không cần các vòng lặp qua coordinator.
- B) Để agent tổng hợp tích lũy mọi nhu cầu xác minh và trả chúng về dưới dạng một batch cho coordinator vào cuối, sau đó coordinator gửi tất cả chúng đến agent web-search cùng một lúc.
- C) Để agent web-search chủ động cache thêm bối cảnh xung quanh mỗi nguồn trong quá trình nghiên cứu ban đầu nhằm dự liệu trước nhu cầu xác minh của tổng hợp.
- D) Cấp cho agent tổng hợp một tool `verify_fact` có phạm vi giới hạn cho các kiểm tra đơn giản, trong khi định tuyến các lần xác minh phức tạp qua coordinator đến agent web-search. **[ĐÁP ÁN ĐÚNG]**

**Vì sao D:** Một tool xác minh sự kiện có phạm vi giới hạn cho phép agent tổng hợp xử lý 85% các kiểm tra đơn giản một cách trực tiếp, loại bỏ hầu hết các vòng lặp, trong khi vẫn bảo toàn đường giao việc qua coordinator cho 15% các lần xác minh phức tạp. Điều này áp dụng least privilege trong khi giảm đáng kể độ trễ.

---

# Kịch bản: Claude Code cho Tích hợp Liên tục

---

## Câu hỏi 16 (Kịch bản: Claude Code cho Tích hợp Liên tục)

**Tình huống:** Pipeline CI/CD của bạn chạy Claude Code CLI (ở chế độ `--print`) sử dụng CLAUDE.md để cung cấp bối cảnh dự án cho việc review code, và các lập trình viên nhìn chung thấy các bản review có nội dung thực chất. Tuy nhiên, họ báo cáo rằng việc tích hợp các phát hiện vào quy trình làm việc là khó khăn—Claude xuất ra các đoạn văn dạng tường thuật phải được sao chép thủ công vào các comment của pull request. Đội nhóm muốn tự động đăng mỗi phát hiện như một comment inline riêng biệt của pull request tại đúng vị trí trong code, điều này đòi hỏi dữ liệu có cấu trúc với đường dẫn file, số dòng, mức độ nghiêm trọng, và bản sửa đề xuất. Cách tiếp cận nào là hiệu quả nhất?

**Cách tiếp cận nào là hiệu quả nhất?**

- A) Thêm một mục “Output Format for Review” vào CLAUDE.md với các ví dụ về phát hiện có cấu trúc để Claude học định dạng mong đợi từ bối cảnh dự án.
- B) Dùng các cờ CLI `--output-format json` và `--json-schema` để áp đặt các phát hiện có cấu trúc, sau đó phân tích cú pháp kết quả để đăng các comment inline qua GitHub API. **[ĐÁP ÁN ĐÚNG]**

- C) Bao gồm các hướng dẫn định dạng rõ ràng trong prompt review yêu cầu mỗi phát hiện tuân theo một mẫu có thể phân tích cú pháp như `[FILE:path] [LINE:n] [SEVERITY:level] ...`.
- D) Giữ định dạng review tường thuật nhưng thêm một bước tóm tắt dùng Claude để tạo ra một bản tóm tắt JSON có cấu trúc về các phát hiện.

**Vì sao B:** Sử dụng `--output-format json` với `--json-schema` áp đặt structured output ở cấp CLI, đảm bảo JSON đúng định dạng với các trường bắt buộc (đường dẫn file, số dòng, mức độ nghiêm trọng, bản sửa đề xuất) có thể được phân tích cú pháp một cách đáng tin cậy và đăng như các comment inline của pull request qua GitHub API. Nó tận dụng các khả năng CLI tích hợp sẵn được thiết kế chuyên cho structured output.

---

## Câu hỏi 17 (Kịch bản: Claude Code cho Tích hợp Liên tục)

**Tình huống:** Đội nhóm của bạn dùng Claude Code để tạo các gợi ý code, nhưng bạn nhận thấy một mẫu hình: các vấn đề không hiển nhiên—các tối ưu hóa hiệu năng làm hỏng các edge case, các đợt dọn dẹp bất ngờ thay đổi hành vi—chỉ được phát hiện khi một thành viên khác trong đội review pull request. Lập luận của Claude trong quá trình tạo cho thấy nó đã cân nhắc các trường hợp này nhưng kết luận rằng cách tiếp cận của nó là đúng. Cách tiếp cận nào trực tiếp giải quyết nguyên nhân gốc rễ của hạn chế tự-kiểm-tra này?

**Cách tiếp cận nào trực tiếp giải quyết nguyên nhân gốc rễ?**

- A) Chạy một phiên bản Claude Code độc lập thứ hai để review các thay đổi mà không có quyền truy cập vào lập luận của bộ tạo. **[ĐÁP ÁN ĐÚNG]**
- B) Bật chế độ extended thinking cho giai đoạn tạo để cho phép cân nhắc kỹ lưỡng hơn trước khi đưa ra gợi ý.
- C) Thêm các hướng dẫn tự-review rõ ràng vào prompt tạo yêu cầu Claude phê bình các gợi ý của chính nó trước khi hoàn thiện kết quả.
- D) Bao gồm đầy đủ các file kiểm thử và tài liệu trong bối cảnh prompt để Claude hiểu rõ hơn hành vi mong đợi trong quá trình tạo.

**Vì sao A:** Một phiên bản Claude Code độc lập thứ hai không có quyền truy cập vào lập luận của bộ tạo trực tiếp giải quyết nguyên nhân gốc rễ bằng cách tránh thiên kiến xác nhận (confirmation bias). Góc nhìn "con mắt tươi mới" này phản chiếu việc review đồng cấp của con người, nơi một người review khác phát hiện các vấn đề mà tác giả đã hợp lý hóa.

---

## Câu hỏi 18 (Kịch bản: Claude Code cho Tích hợp Liên tục)

**Tình huống:** Thành phần review code của bạn mang tính lặp: Claude phân tích file đã thay đổi, sau đó có thể yêu cầu các file liên quan (imports, lớp cơ sở, tests) qua các lệnh gọi tool để hiểu bối cảnh trước khi đưa ra phản hồi cuối cùng. Ứng dụng của bạn định nghĩa một tool cho phép Claude yêu cầu nội dung file; Claude gọi tool, nhận kết quả, và tiếp tục phân tích. Bạn đang đánh giá việc xử lý theo batch để giảm chi phí API. Hạn chế kỹ thuật chính khi cân nhắc xử lý batch cho quy trình làm việc này là gì?

**Hạn chế kỹ thuật chính là gì?**

- A) Xử lý batch không bao gồm các correlation ID để ánh xạ kết quả đầu ra trở lại các yêu cầu đầu vào.
- B) Mô hình bất đồng bộ không thể thực thi tool giữa chừng của một yêu cầu và trả kết quả về để Claude tiếp tục phân tích. **[ĐÁP ÁN ĐÚNG]**
- C) Batch API không hỗ trợ các định nghĩa tool trong các tham số yêu cầu.
- D) Độ trễ xử lý batch lên đến 24 giờ là quá chậm cho phản hồi pull request, mặc dù quy trình làm việc nếu không thì vẫn vận hành được.

**Vì sao B:** Một mô hình Batch API bất đồng bộ kiểu "fire-and-forget" không có cơ chế nào để chặn một lệnh gọi tool trong một yêu cầu, thực thi tool, và trả kết quả về để Claude tiếp tục phân tích. Điều này về cơ bản là không tương thích với các quy trình gọi tool mang tính lặp vốn đòi hỏi nhiều vòng yêu cầu/phản hồi tool trong một tương tác logic đơn lẻ.

---

## Câu hỏi 19 (Kịch bản: Claude Code cho Tích hợp Liên tục)

**Tình huống:** Hệ thống CI/CD của bạn chạy ba phân tích dựa trên Claude: (1) các kiểm tra style nhanh trên mỗi pull request chặn việc merge cho đến khi hoàn thành, (2) các đợt audit bảo mật toàn diện hàng tuần trên toàn bộ codebase, và (3) việc tạo test-case hàng đêm cho các module được thay đổi gần đây. Message Batches API cung cấp khoản tiết kiệm 50% nhưng việc xử lý có thể mất đến 24 giờ. Bạn muốn tối ưu hóa chi phí API trong khi duy trì một trải nghiệm lập trình viên chấp nhận được. Sự kết hợp nào ghép đúng mỗi nhiệm vụ với một cách tiếp cận API?

**Sự kết hợp nào là đúng?**

- A) Dùng Message Batches API cho cả ba nhiệm vụ để tối đa hóa khoản tiết kiệm 50%, cấu hình pipeline để poll cho việc hoàn thành batch.
- B) Dùng các lệnh gọi đồng bộ cho các kiểm tra style của pull request; dùng Message Batches API cho các đợt audit bảo mật hàng tuần và việc tạo test hàng đêm. **[ĐÁP ÁN ĐÚNG]**
- C) Dùng các lệnh gọi đồng bộ cho cả ba nhiệm vụ để có thời gian phản hồi nhất quán, dựa vào prompt caching để giảm chi phí trên các khối lượng công việc.
- D) Dùng các lệnh gọi đồng bộ cho các kiểm tra style của pull request và việc tạo test hàng đêm; chỉ dùng Message Batches API cho các đợt audit bảo mật hàng tuần.

**Vì sao B:** Các kiểm tra style của pull request chặn các lập trình viên và đòi hỏi phản hồi tức thì qua các lệnh gọi đồng bộ, trong khi các đợt audit bảo mật hàng tuần và việc tạo test hàng đêm là các nhiệm vụ theo lịch với thời hạn linh hoạt có thể chịu được cửa sổ batch lên đến 24 giờ—nắm bắt khoản tiết kiệm 50% cho cả hai.

---

## Câu hỏi 20 (Kịch bản: Claude Code cho Tích hợp Liên tục)

**Tình huống:** Các bản review tự động của bạn tìm ra các vấn đề thực sự, nhưng các lập trình viên báo cáo rằng phản hồi không thể hành động được. Các phát hiện bao gồm các cụm từ như "logic định tuyến phiếu phức tạp" hoặc "null pointer tiềm ẩn" mà không chỉ rõ chính xác cần thay đổi điều gì. Khi bạn thêm các hướng dẫn chi tiết như "luôn bao gồm các gợi ý sửa cụ thể," mô hình vẫn tạo ra kết quả không nhất quán—đôi khi chi tiết, đôi khi mơ hồ. Kỹ thuật prompting nào tạo ra phản hồi có thể hành động được một cách nhất quán đáng tin cậy nhất?

**Kỹ thuật prompting nào là đáng tin cậy nhất?**

- A) Tinh chỉnh thêm các hướng dẫn với các yêu cầu rõ ràng hơn cho mỗi phần của định dạng phản hồi (vị trí, vấn đề, mức độ nghiêm trọng, bản sửa đề xuất).
- B) Mở rộng context window để bao gồm nhiều codebase xung quanh hơn nhằm cho mô hình đủ thông tin để đề xuất các bản sửa cụ thể.
- C) Triển khai một cách tiếp cận hai lượt trong đó một prompt xác định các vấn đề và một prompt thứ hai tạo các bản sửa, cho phép chuyên môn hóa.
- D) Thêm 3–4 ví dụ few-shot cho thấy chính xác định dạng bắt buộc: vấn đề được xác định, vị trí trong code, gợi ý sửa cụ thể. **[ĐÁP ÁN ĐÚNG]**

**Vì sao D:** Các ví dụ few-shot là kỹ thuật hiệu quả nhất để đạt được định dạng kết quả nhất quán khi chỉ riêng các hướng dẫn tạo ra kết quả biến thiên. Cung cấp 3–4 ví dụ cho thấy chính xác cấu trúc mong muốn (vấn đề, vị trí, bản sửa cụ thể) cho mô hình một mẫu hình cụ thể để noi theo, điều này đáng tin cậy hơn các hướng dẫn trừu tượng.

---

## Câu hỏi 21 (Kịch bản: Claude Code cho Tích hợp Liên tục)

**Tình huống:** Pipeline CI/CD của bạn bao gồm hai chế độ review code dựa trên Claude: một hook pre-merge-commit chặn việc merge pull request cho đến khi hoàn thành, và một "phân tích sâu" chạy qua đêm, poll cho việc hoàn thành batch, và đăng các gợi ý chi tiết lên pull request. Bạn muốn giảm chi phí API bằng cách dùng Message Batches API, vốn cung cấp khoản tiết kiệm 50% nhưng đòi hỏi polling và có thể mất đến 24 giờ. Chế độ nào nên dùng xử lý batch?

**Chế độ nào nên dùng xử lý batch?**

- A) Chỉ hook pre-merge-commit.
- B) Chỉ phân tích sâu. **[ĐÁP ÁN ĐÚNG]**
- C) Cả hai chế độ.
- D) Không chế độ nào.

**Vì sao B:** Phân tích sâu là một ứng viên lý tưởng cho xử lý batch vì nó vốn đã chạy qua đêm, chịu được độ trễ, và dùng một mô hình polling trước khi công bố kết quả—khớp với kiến trúc bất đồng bộ, dựa trên polling của Message Batches API trong khi nắm bắt khoản tiết kiệm 50%.

---

## Câu hỏi 22 (Kịch bản: Claude Code cho Tích hợp Liên tục)

**Tình huống:** Bản review tự động của bạn phân tích các comment và docstring. Prompt hiện tại hướng dẫn Claude "kiểm tra rằng các comment chính xác và cập nhật." Các phát hiện thường gắn cờ các mẫu hình chấp nhận được (các đánh dấu TODO, các mô tả đơn giản) trong khi bỏ sót các comment mô tả hành vi mà code không còn hiện thực nữa. Thay đổi nào giải quyết nguyên nhân gốc rễ của việc phân tích không nhất quán này?

**Thay đổi nào giải quyết nguyên nhân gốc rễ?**

- A) Bao gồm dữ liệu `git blame` để Claude có thể xác định các comment có trước các thay đổi code gần đây.
- B) Thêm các ví dụ few-shot về các comment gây hiểu lầm để giúp mô hình nhận ra các mẫu hình tương tự trong codebase.
- C) Lọc các mẫu hình comment TODO, FIXME, và mô tả trước khi phân tích để giảm nhiễu.
- D) Chỉ định các tiêu chí rõ ràng: chỉ gắn cờ các comment khi hành vi mà chúng tuyên bố mâu thuẫn với hành vi thực tế của code. **[ĐÁP ÁN ĐÚNG]**

**Vì sao D:** Các tiêu chí rõ ràng—chỉ gắn cờ các comment khi hành vi được tuyên bố mâu thuẫn với hành vi code thực tế—trực tiếp giải quyết nguyên nhân gốc rễ bằng cách thay thế một hướng dẫn mơ hồ bằng một định nghĩa chính xác về điều gì cấu thành một vấn đề. Điều này giảm các false positive trên các mẫu hình chấp nhận được và việc bỏ sót các comment thực sự gây hiểu lầm.

---

## Câu hỏi 23 (Kịch bản: Claude Code cho Tích hợp Liên tục)

**Tình huống:** Hệ thống review code tự động của bạn cho thấy các đánh giá mức độ nghiêm trọng không nhất quán—các vấn đề tương tự như rủi ro null pointer được đánh giá là “critical” trong một số pull request nhưng chỉ là “medium” trong các pull request khác. Các khảo sát lập trình viên cho thấy sự mất tin tưởng đang gia tăng—nhiều người bắt đầu gạt bỏ các phát hiện mà không đọc vì “một nửa là sai.” Các danh mục có tỷ lệ false-positive cao làm xói mòn niềm tin vào các danh mục chính xác. Cách tiếp cận nào khôi phục niềm tin của lập trình viên tốt nhất trong khi cải thiện hệ thống?

**Cách tiếp cận nào khôi phục niềm tin của lập trình viên tốt nhất?**

- A) Tạm thời vô hiệu hóa các danh mục có tỷ lệ false-positive cao (style, naming, documentation) và chỉ giữ các danh mục có độ chính xác cao trong khi cải thiện các prompt. **[ĐÁP ÁN ĐÚNG]**
- B) Giữ tất cả các danh mục được bật nhưng hiển thị điểm tin cậy với mỗi phát hiện để các lập trình viên có thể quyết định điều gì cần điều tra.
- C) Giữ tất cả các danh mục được bật và thêm các ví dụ few-shot để cải thiện độ chính xác cho mỗi danh mục trong vài tuần tới.
- D) Áp dụng một mức giảm độ nghiêm ngặt đồng đều trên tất cả các danh mục để đưa tỷ lệ false-positive tổng thể xuống.

**Vì sao A:** Tạm thời vô hiệu hóa các danh mục có tỷ lệ false-positive cao ngay lập tức chặn sự xói mòn niềm tin bằng cách loại bỏ các phát hiện gây nhiễu khiến các lập trình viên gạt bỏ mọi thứ, trong khi bảo toàn giá trị từ các danh mục có độ chính xác cao như bảo mật và tính đúng đắn. Nó cũng tạo không gian để cải thiện các prompt cho các danh mục có vấn đề trước khi bật lại chúng.

---

## Câu hỏi 24 (Kịch bản: Claude Code cho Tích hợp Liên tục)

**Tình huống:** Bản review tự động của bạn tạo các gợi ý test-case cho mỗi pull request. Khi review một pull request thêm tính năng theo dõi việc hoàn thành khóa học, Claude gợi ý 10 test case, nhưng phản hồi của lập trình viên cho thấy 6 trong số đó trùng lặp các kịch bản đã được bao phủ bởi bộ test hiện có. Thay đổi nào giảm các gợi ý trùng lặp hiệu quả nhất?

**Thay đổi nào là hiệu quả nhất?**

- A) Bao gồm file test hiện có trong bối cảnh để Claude có thể xác định những kịch bản nào đã được bao phủ. **[ĐÁP ÁN ĐÚNG]**
- B) Giảm số lượng gợi ý được yêu cầu từ 10 xuống 5, giả định rằng Claude ưu tiên các trường hợp có giá trị nhất trước.
- C) Thêm các hướng dẫn chỉ đạo Claude chỉ tập trung vào các edge case và điều kiện lỗi thay vì các đường thành công.
- D) Triển khai hậu xử lý lọc các gợi ý có mô tả khớp với các tên test hiện có qua sự chồng lấn từ khóa.

**Vì sao A:** Bao gồm file test hiện có khắc phục nguyên nhân gốc rễ của sự trùng lặp: Claude chỉ có thể tránh gợi ý các kịch bản đã được bao phủ nếu nó biết những test nào đã tồn tại. Điều này cho Claude thông tin cần thiết để đề xuất các test thực sự mới, có giá trị.

---

## Câu hỏi 25 (Kịch bản: Claude Code cho Tích hợp Liên tục)

**Tình huống:** Sau khi một bản review tự động ban đầu xác định 12 phát hiện, một lập trình viên push các commit mới để giải quyết các vấn đề. Việc chạy lại review tạo ra 8 phát hiện, nhưng các lập trình viên báo cáo rằng 5 trong số đó trùng lặp các comment trước đó trên code đã được sửa trong các commit mới. Cách hiệu quả nhất để loại bỏ phản hồi dư thừa này trong khi duy trì tính kỹ lưỡng là gì?

**Cách hiệu quả nhất để loại bỏ phản hồi dư thừa là gì?**

- A) Chỉ chạy review khi pull request được tạo và ở trạng thái pre-merge cuối cùng, bỏ qua các commit trung gian.
- B) Thêm một bộ lọc hậu xử lý loại bỏ các phát hiện khớp với các phát hiện trước đó theo đường dẫn file và mô tả vấn đề trước khi đăng các comment.
- C) Giới hạn phạm vi review vào các file đã thay đổi trong lần push gần nhất, loại trừ các file từ các commit trước đó.
- D) Bao gồm các phát hiện review trước đó trong bối cảnh và hướng dẫn Claude chỉ báo cáo các vấn đề mới hoặc vẫn chưa được giải quyết. **[ĐÁP ÁN ĐÚNG]**

**Vì sao D:** Bao gồm các phát hiện review trước đó trong bối cảnh cho phép Claude phân biệt các vấn đề mới với những vấn đề đã được giải quyết trong các commit gần đây. Điều này bảo toàn tính kỹ lưỡng của review trong khi dùng lập luận của Claude để tránh phản hồi dư thừa trên code đã được sửa.

---

## Câu hỏi 26 (Kịch bản: Claude Code cho Tích hợp Liên tục)

**Tình huống:** Script pipeline của bạn chạy `claude "Analyze this pull request for security issues"`, nhưng tác vụ bị treo vô thời hạn. Log cho thấy Claude Code đang chờ đầu vào tương tác. Cách tiếp cận đúng để chạy Claude Code trong một pipeline tự động là gì?

**Cách tiếp cận đúng là gì?**

- A) Thêm cờ `--batch`: `claude --batch "Analyze this pull request for security issues"`.

- B) Thêm cờ `-p`: `claude -p "Analyze this pull request for security issues"`. **[ĐÁP ÁN ĐÚNG]**
- C) Chuyển hướng stdin từ `/dev/null`: `claude "Analyze this pull request for security issues" < /dev/null`.
- D) Đặt biến môi trường `CLAUDE_HEADLESS=true` trước khi chạy lệnh.

**Vì sao B:** Cờ `-p` (hay `--print`) là cách được tài liệu hóa để chạy Claude Code không tương tác. Nó xử lý prompt, in kết quả ra stdout, và thoát mà không chờ đầu vào của người dùng—lý tưởng cho các pipeline CI/CD.

---

## Câu hỏi 27 (Kịch bản: Claude Code cho Tích hợp Liên tục)

**Tình huống:** Một pull request thay đổi 14 file trong một module theo dõi tồn kho. Một bản review một-lượt phân tích tất cả các file cùng nhau tạo ra các kết quả không nhất quán: phản hồi chi tiết trên một số file nhưng các comment hời hợt trên các file khác, bỏ sót các bug hiển nhiên, và phản hồi mâu thuẫn (một mẫu hình bị gắn cờ trong một file nhưng code y hệt được duyệt trong một file khác trong cùng pull request). Bạn nên tái cấu trúc bản review như thế nào?

**Bạn nên tái cấu trúc bản review như thế nào?**

- A) Chạy ba lượt review toàn-pull-request độc lập và chỉ gắn cờ các vấn đề xuất hiện trong ít nhất hai trong ba lần chạy.
- B) Chia thành các lượt tập trung: review từng file riêng lẻ cho các vấn đề cục bộ, sau đó chạy một lượt riêng định hướng tích hợp để xem xét các luồng dữ liệu liên file. **[ĐÁP ÁN ĐÚNG]**
- C) Yêu cầu các lập trình viên chia các pull request lớn thành các bản nộp nhỏ hơn gồm 3–4 file trước khi chạy review tự động.
- D) Chuyển sang một mô hình lớn hơn với context window lớn hơn để nó có thể chú ý đầy đủ đến tất cả 14 file trong một lượt.

**Vì sao B:** Các lượt tập trung theo từng file giải quyết nguyên nhân gốc rễ—sự pha loãng chú ý—bằng cách đảm bảo độ sâu nhất quán và phát hiện vấn đề cục bộ đáng tin cậy. Sau đó một lượt riêng định hướng tích hợp bao phủ các mối quan tâm liên file như các tương tác phụ thuộc và luồng dữ liệu.

---

## Câu hỏi 28 (Kịch bản: Claude Code cho Tích hợp Liên tục)

**Tình huống:** Bản review code tự động của bạn trung bình có 15 phát hiện cho mỗi pull request, và các lập trình viên báo cáo tỷ lệ false-positive 40%. Nút thắt cổ chai là thời gian điều tra: các lập trình viên phải nhấp vào từng phát hiện để đọc lý lẽ của Claude trước khi quyết định nên sửa hay gạt bỏ nó. CLAUDE.md của bạn đã chứa các quy tắc toàn diện về các mẫu hình chấp nhận được, và các bên liên quan đã từ chối bất kỳ cách tiếp cận nào lọc các phát hiện trước khi các lập trình viên nhìn thấy chúng. Thay đổi nào giải quyết thời gian điều tra tốt nhất?

**Thay đổi nào giải quyết thời gian điều tra tốt nhất?**

- A) Yêu cầu Claude bao gồm lý lẽ và ước lượng độ tin cậy của nó trực tiếp trong mỗi phát hiện. **[ĐÁP ÁN ĐÚNG]**

- B) Thêm một bộ hậu xử lý phân tích các mẫu hình phát hiện và tự động ức chế những phát hiện khớp với các chữ ký false-positive trong lịch sử.
- C) Phân loại các phát hiện thành “các vấn đề chặn” so với “các gợi ý,” với các yêu cầu review khác nhau theo cấp độ.
- D) Cấu hình Claude chỉ hiển thị các phát hiện có độ tin cậy cao, lọc các cờ không chắc chắn trước khi các lập trình viên nhìn thấy chúng.

**Vì sao A:** Bao gồm lý lẽ và độ tin cậy trực tiếp trong mỗi phát hiện giảm thời gian điều tra bằng cách cho phép các lập trình viên phân loại nhanh mà không cần mở từng phát hiện. Nó thỏa mãn ràng buộc “không lọc” vì tất cả các phát hiện vẫn hiển thị trong khi tăng tốc việc ra quyết định của lập trình viên.

---

## Câu hỏi 29 (Kịch bản: Claude Code cho Tích hợp Liên tục)

**Tình huống:** Phân tích bản review code tự động của bạn cho thấy sự khác biệt lớn về tỷ lệ false-positive theo danh mục phát hiện: các phát hiện về bảo mật/tính đúng đắn có 8% false positive, các phát hiện về hiệu năng 18%, các phát hiện về style/naming 52%, và các phát hiện về documentation 48%. Các khảo sát lập trình viên cho thấy sự mất tin tưởng đang gia tăng—nhiều người bắt đầu gạt bỏ các phát hiện mà không đọc vì “một nửa là sai.” Các danh mục có tỷ lệ false-positive cao làm xói mòn niềm tin vào các danh mục chính xác. Cách tiếp cận nào khôi phục niềm tin của lập trình viên tốt nhất trong khi cải thiện hệ thống?

**Cách tiếp cận nào khôi phục niềm tin của lập trình viên tốt nhất?**

- A) Tạm thời vô hiệu hóa các danh mục có tỷ lệ false-positive cao (style, naming, documentation) và chỉ giữ các danh mục có độ chính xác cao trong khi cải thiện các prompt. **[ĐÁP ÁN ĐÚNG]**
- B) Giữ tất cả các danh mục được bật nhưng hiển thị điểm tin cậy với mỗi phát hiện để các lập trình viên có thể quyết định điều gì cần điều tra.
- C) Giữ tất cả các danh mục được bật và thêm các ví dụ few-shot để cải thiện độ chính xác cho mỗi danh mục trong vài tuần tới.
- D) Áp dụng một mức giảm độ nghiêm ngặt đồng đều trên tất cả các danh mục để đưa tỷ lệ false-positive tổng thể xuống.

**Vì sao A:** Tạm thời vô hiệu hóa các danh mục có tỷ lệ false-positive cao ngay lập tức chặn sự xói mòn niềm tin bằng cách loại bỏ các phát hiện gây nhiễu khiến các lập trình viên gạt bỏ mọi thứ, trong khi bảo toàn giá trị từ các danh mục có độ chính xác cao như bảo mật và tính đúng đắn. Nó cũng tạo không gian để cải thiện các prompt cho các danh mục có vấn đề trước khi bật lại chúng.

---

## Câu hỏi 30 (Kịch bản: Claude Code cho Tích hợp Liên tục)

**Tình huống:** Đội nhóm của bạn muốn giảm chi phí API cho việc phân tích tự động. Hiện tại, các lệnh gọi Claude đồng bộ hỗ trợ hai quy trình làm việc: (1) một kiểm tra pre-merge mang tính chặn vốn phải hoàn thành trước khi các lập trình viên có thể merge, và (2) một báo cáo nợ kỹ thuật được tạo qua đêm để review vào sáng hôm sau. Quản lý của bạn đề xuất chuyển cả hai sang Message Batches API để tiết kiệm 50%. Bạn nên đánh giá đề xuất này như thế nào?

**Bạn nên đánh giá đề xuất này như thế nào?**

- A) Chuyển cả hai sang xử lý batch với phương án dự phòng là các lệnh gọi đồng bộ nếu các batch mất quá lâu.
- B) Chuyển cả hai quy trình làm việc sang xử lý batch với việc poll trạng thái để xác minh sự hoàn thành.
- C) Chỉ dùng xử lý batch cho các báo cáo nợ kỹ thuật; giữ các lệnh gọi đồng bộ cho các kiểm tra pre-merge. **[ĐÁP ÁN ĐÚNG]**
- D) Giữ các lệnh gọi đồng bộ cho cả hai quy trình làm việc để tránh các vấn đề với việc sắp xếp thứ tự kết quả batch.

**Vì sao C:** Việc xử lý của Message Batches API có thể mất đến 24 giờ mà không có SLA về độ trễ, điều này chấp nhận được cho các báo cáo nợ kỹ thuật qua đêm nhưng không chấp nhận được cho các kiểm tra pre-merge mang tính chặn nơi các lập trình viên phải chờ. Điều này ghép mỗi quy trình làm việc với đúng API dựa trên các yêu cầu về độ trễ.

---

## Kịch bản: Sinh mã với Claude Code

---

## Câu hỏi 31 (Kịch bản: Sinh mã với Claude Code)

**Tình huống:** Bạn yêu cầu Claude Code triển khai một hàm chuyển đổi các phản hồi API thành định dạng chuẩn hóa nội bộ. Sau hai vòng lặp, cấu trúc đầu ra vẫn không khớp với kỳ vọng—một số trường được lồng nhau theo cách khác và các dấu thời gian được định dạng sai. Bạn đã mô tả yêu cầu bằng văn xuôi, nhưng mỗi lần Claude lại diễn giải khác nhau.

**Cách tiếp cận nào hiệu quả nhất cho vòng lặp tiếp theo?**

- A) Viết một JSON schema mô tả cấu trúc đầu ra mong đợi và kiểm tra đầu ra của Claude dựa trên schema đó sau mỗi vòng lặp.
- B) Cung cấp 2–3 ví dụ đầu vào-đầu ra cụ thể minh họa phép chuyển đổi mong đợi cho các phản hồi API tiêu biểu. **[ĐÁP ÁN ĐÚNG]**
- C) Viết lại yêu cầu với độ chính xác kỹ thuật cao hơn, chỉ định chính xác cách ánh xạ trường, quy tắc lồng nhau và chuỗi định dạng dấu thời gian.
- D) Yêu cầu Claude giải thích cách hiểu hiện tại của nó về các yêu cầu để xác định nơi các diễn giải bị lệch nhau.

**Vì sao B:** Các ví dụ đầu vào-đầu ra cụ thể loại bỏ sự mơ hồ vốn có trong các mô tả bằng văn xuôi bằng cách cho Claude thấy chính xác kết quả chuyển đổi mong đợi. Điều này giải quyết trực tiếp nguyên nhân gốc rễ—việc diễn giải sai yêu cầu dạng văn bản—bằng cách cung cấp các mẫu rõ ràng cho việc lồng trường và định dạng dấu thời gian.

---

## Câu hỏi 32 (Kịch bản: Sinh mã với Claude Code)

**Tình huống:** Bạn cần thêm Slack làm một kênh thông báo mới. Codebase hiện có các mẫu rõ ràng, đã được thiết lập cho các kênh email, SMS và push. Tuy nhiên, API của Slack cung cấp những cách tích hợp khác biệt căn bản—incoming webhooks (đơn giản, một chiều), bot tokens (hỗ trợ xác nhận gửi và điều khiển theo chương trình), hoặc Slack Apps (sự kiện hai chiều, yêu cầu phê duyệt workspace). Nhiệm vụ của bạn ghi "thêm hỗ trợ Slack" mà không chỉ định phương thức tích hợp hay yêu cầu các tính năng nâng cao như theo dõi việc gửi.

**Bạn nên tiếp cận nhiệm vụ này như thế nào?**

- A) Bắt đầu ở chế độ thực thi trực tiếp dùng incoming webhooks để khớp với mẫu thông báo một chiều hiện có.
- B) Chuyển sang planning mode để khám phá các phương án tích hợp và hệ quả về mặt kiến trúc, sau đó đưa ra khuyến nghị trước khi triển khai. **[ĐÁP ÁN ĐÚNG]**
- C) Bắt đầu ở chế độ thực thi trực tiếp bằng cách dựng khung lớp kênh Slack dùng các mẫu hiện có, hoãn quyết định về phương thức tích hợp.
- D) Bắt đầu ở chế độ thực thi trực tiếp dùng cách tiếp cận bot-token để đảm bảo có thể xác nhận việc gửi.

**Vì sao B:** Việc tích hợp Slack có nhiều cách tiếp cận hợp lệ với hệ quả kiến trúc khác biệt đáng kể, và các yêu cầu thì mơ hồ. Planning mode cho phép bạn đánh giá các trade-off giữa webhooks, bot tokens và Slack Apps rồi thống nhất về một cách tiếp cận trước khi triển khai.

---

## Câu hỏi 33 (Kịch bản: Sinh mã với Claude Code)

**Tình huống:** Tệp CLAUDE.md của bạn đã phình lên hơn 400 dòng, chứa các chuẩn lập trình, quy ước kiểm thử, một checklist review PR chi tiết, hướng dẫn triển khai và quy trình migration cơ sở dữ liệu. Bạn muốn Claude luôn tuân theo các chuẩn lập trình và quy ước kiểm thử, nhưng chỉ áp dụng hướng dẫn review PR, deploy và migration khi đang làm các tác vụ đó.

**Cách tái cấu trúc nào hiệu quả nhất?**

- A) Chuyển tất cả hướng dẫn vào các tệp Skills riêng được tổ chức theo loại quy trình làm việc, chỉ để lại một mô tả dự án ngắn gọn trong CLAUDE.md.
- B) Giữ mọi thứ trong CLAUDE.md nhưng dùng cú pháp `@import` để tổ chức thành các tệp được bảo trì riêng theo từng danh mục.
- C) Tách CLAUDE.md thành các tệp dưới `.claude/rules/` với các mẫu glob ràng buộc theo đường dẫn để mỗi quy tắc chỉ được tải cho các loại tệp liên quan.
- D) Giữ các chuẩn phổ quát trong CLAUDE.md và tạo các Skills cho hướng dẫn theo từng quy trình cụ thể (review PR, deploy, migration) kèm các từ khóa kích hoạt. **[ĐÁP ÁN ĐÚNG]**

**Vì sao D:** Nội dung CLAUDE.md được tải trong mọi session, đảm bảo các chuẩn lập trình và quy ước kiểm thử luôn được áp dụng, trong khi các Skills được gọi theo nhu cầu khi Claude phát hiện các từ khóa kích hoạt—lý tưởng cho hướng dẫn theo quy trình cụ thể như review PR, deploy và migration.

---

## Câu hỏi 34 (Kịch bản: Sinh mã với Claude Code)

**Tình huống:** Bạn được giao nhiệm vụ tái cấu trúc ứng dụng monolithic của nhóm thành microservices. Việc này tác động đến các thay đổi trên hàng chục tệp và đòi hỏi các quyết định về ranh giới dịch vụ và phụ thuộc giữa các module.

**Bạn nên chọn cách tiếp cận nào?**

- A) Chuyển sang planning mode để khám phá codebase, hiểu các phụ thuộc và thiết kế cách tiếp cận triển khai trước khi thực hiện thay đổi. **[ĐÁP ÁN ĐÚNG]**
- B) Bắt đầu ở chế độ thực thi trực tiếp và chỉ chuyển sang planning sau khi gặp phải độ phức tạp bất ngờ trong quá trình triển khai.
- C) Bắt đầu ở chế độ thực thi trực tiếp và thực hiện các thay đổi tăng dần, để việc triển khai tự bộc lộ các ranh giới dịch vụ tự nhiên.
- D) Dùng thực thi trực tiếp với các hướng dẫn chi tiết định trước, chỉ định cấu trúc của từng dịch vụ.

**Vì sao A:** Planning mode là chiến lược đúng đắn cho việc tái cấu trúc kiến trúc phức tạp như tách một monolith: nó cho phép khám phá an toàn và đưa ra quyết định có cơ sở về các ranh giới trước khi cam kết với những thay đổi có thể tốn kém trên nhiều tệp.

---

## Câu hỏi 35 (Kịch bản: Sinh mã với Claude Code)

**Tình huống:** Nhóm của bạn đã tạo một skill `/analyze-codebase` thực hiện phân tích mã sâu—quét phụ thuộc, đếm độ phủ kiểm thử và các chỉ số chất lượng mã. Sau khi chạy lệnh, các thành viên trong nhóm báo cáo rằng Claude trở nên kém phản hồi hơn trong session và mất ngữ cảnh của nhiệm vụ ban đầu.

**Bạn khắc phục điều này hiệu quả nhất bằng cách nào trong khi vẫn giữ đầy đủ khả năng phân tích?**

- A) Thêm `context: fork` trong frontmatter của skill để chạy phân tích trong một context subagent tách biệt. **[ĐÁP ÁN ĐÚNG]**
- B) Thêm `model: haiku` trong frontmatter để dùng một mô hình nhanh hơn, rẻ hơn cho việc phân tích.
- C) Tách skill thành ba skill nhỏ hơn, mỗi skill tạo ra ít đầu ra hơn.
- D) Thêm hướng dẫn vào skill để nén tất cả kết quả thành một bản tóm tắt ngắn trước khi hiển thị.

**Vì sao A:** `context: fork` chạy phân tích trong một context subagent tách biệt nên đầu ra lớn không làm ô nhiễm context window của session chính và Claude không bị mất dấu nhiệm vụ ban đầu. Nó bảo toàn đầy đủ khả năng phân tích trong khi giữ cho session chính luôn phản hồi tốt.

---

## Câu hỏi 36 (Kịch bản: Sinh mã với Claude Code)

**Tình huống:** Nhóm của bạn dùng một skill `/commit` trong `.claude/skills/commit/SKILL.md`. Một lập trình viên muốn tùy chỉnh nó cho quy trình làm việc cá nhân của họ (định dạng commit message khác, thêm các kiểm tra) mà không ảnh hưởng đến đồng đội.

**Bạn khuyến nghị điều gì?**

- A) Tạo một phiên bản cá nhân dưới `~/.claude/skills/` với tên khác, ví dụ `/my-commit`. **[ĐÁP ÁN ĐÚNG]**
- B) Thêm logic điều kiện dựa trên username trong frontmatter của skill dự án.
- C) Tạo một phiên bản cá nhân tại `~/.claude/skills/commit/SKILL.md` với cùng tên.
- D) Đặt `override: true` trong frontmatter của skill cá nhân để ưu tiên nó hơn phiên bản dự án.

**Vì sao A:** Các skill cá nhân được ưu tiên hơn các skill dự án có cùng tên, vì vậy việc dùng lại tên `commit` sẽ âm thầm che khuất skill của nhóm chỉ đối với riêng lập trình viên này — họ sẽ ngừng nhận được các cập nhật mỗi khi nhóm cải tiến `/commit`, và phải tự nhớ rằng mình đang chạy một skill khác dưới cùng một lệnh. Đặt tên phiên bản cá nhân là `/my-commit` giúp tránh hoàn toàn xung đột này: lập trình viên vẫn tiếp tục dùng `/commit` do nhóm duy trì, đồng thời có một skill riêng, được đặt tên rõ ràng cho quy trình làm việc cá nhân, không có nguy cơ nhầm lẫn giữa hai skill hay bỏ lỡ các cập nhật của nhóm.

---

## Câu hỏi 37 (Kịch bản: Sinh mã với Claude Code)

**Tình huống:** Nhóm của bạn đã dùng Claude Code trong nhiều tháng. Gần đây, ba lập trình viên báo cáo rằng Claude tuân theo hướng dẫn "luôn bao gồm xử lý lỗi toàn diện," nhưng một lập trình viên thứ tư vừa gia nhập lại nói Claude không tuân theo. Cả bốn người làm việc trong cùng một repo và có mã được cập nhật mới nhất.

**Nguyên nhân và cách khắc phục khả dĩ nhất là gì?**

- A) Hướng dẫn nằm trong các tệp `~/.claude/CLAUDE.md` ở cấp người dùng của những lập trình viên ban đầu, không nằm trong `.claude/CLAUDE.md` của dự án. Hãy chuyển hướng dẫn sang tệp cấp dự án để tất cả thành viên trong nhóm đều nhận được. **[ĐÁP ÁN ĐÚNG]**
- B) Tệp `~/.claude/CLAUDE.md` của lập trình viên mới chứa các hướng dẫn xung đột ghi đè lên cài đặt dự án; họ nên xóa phần xung đột đó.
- C) Claude Code học các tùy chọn theo từng người dùng theo thời gian; lập trình viên mới phải lặp lại yêu cầu cho đến khi Claude "nhớ" nó.
- D) Claude Code cache CLAUDE.md sau lần đọc đầu tiên; các lập trình viên ban đầu dùng phiên bản đã cache. Mọi người nên xóa cache của Claude Code.

**Vì sao A:** Nếu hướng dẫn chỉ được thêm vào các tệp cấu hình cấp người dùng của những lập trình viên ban đầu mà không thêm vào `.claude/CLAUDE.md` cấp dự án, các thành viên mới sẽ không nhận được. Việc chuyển nó sang cấu hình cấp dự án đảm bảo tất cả thành viên hiện tại và tương lai đều tự động nhận được hướng dẫn.

---

## Câu hỏi 38 (Kịch bản: Sinh mã với Claude Code)

**Tình huống:** Bạn nhận thấy việc đưa 2–3 ví dụ triển khai endpoint đầy đủ làm ngữ cảnh giúp cải thiện đáng kể tính nhất quán khi sinh các endpoint API mới. Tuy nhiên, ngữ cảnh này chỉ hữu ích khi tạo endpoint mới—không hữu ích khi debug, review mã, hoặc làm các việc khác trong thư mục API.

**Cách tiếp cận cấu hình nào hiệu quả nhất?**

- A) Thêm các ví dụ endpoint và tài liệu về mẫu vào CLAUDE.md của dự án để chúng luôn sẵn có.
- B) Tham chiếu thủ công các ví dụ endpoint trong mỗi yêu cầu sinh mã bằng cách sao chép mã vào prompt.
- C) Cấu hình các quy tắc theo đường dẫn trong `.claude/rules/api/` bao gồm các ví dụ endpoint và kích hoạt khi làm việc trong thư mục API.
- D) Tạo một skill tham chiếu các ví dụ endpoint và chứa hướng dẫn tuân theo mẫu, được gọi theo nhu cầu qua một slash command. **[ĐÁP ÁN ĐÚNG]**

**Vì sao D:** Một skill được gọi theo nhu cầu chỉ tải ngữ cảnh ví dụ khi sinh các endpoint mới, chứ không phải trong các tác vụ không liên quan như debug hay review. Điều này giữ cho context chính sạch sẽ trong khi vẫn bảo toàn việc sinh mã chất lượng cao khi cần.

---

## Câu hỏi 39 (Kịch bản: Sinh mã với Claude Code)

**Tình huống:** Nhóm của bạn đã tạo một skill `/migration` sinh các tệp migration cơ sở dữ liệu. Nó nhận tên migration qua `$ARGUMENTS`. Trong môi trường production bạn quan sát thấy ba vấn đề: (1) các lập trình viên thường chạy skill mà không có đối số, gây ra các tệp bị đặt tên kém, (2) skill đôi khi dùng chi tiết schema cơ sở dữ liệu từ các cuộc trò chuyện trước không liên quan, và (3) một lập trình viên vô tình chạy việc dọn dẹp thử nghiệm có tính phá hủy khi skill có quyền truy cập tool quá rộng.

**Cách tiếp cận cấu hình nào khắc phục được cả ba vấn đề?**

- A) Dùng tham số vị trí `$1` và `$2` thay cho `$ARGUMENTS` để bắt buộc các đầu vào cụ thể, đưa vào các tham chiếu tệp schema rõ ràng qua cú pháp `@` để kiểm soát ngữ cảnh, và thêm một mô tả frontmatter cảnh báo về các thao tác phá hủy.
- B) Thêm `argument-hint` trong frontmatter để yêu cầu các tham số bắt buộc, dùng `context: fork` để cô lập việc thực thi, và giới hạn `allowed-tools` chỉ cho các thao tác ghi tệp. **[ĐÁP ÁN ĐÚNG]**
- C) Tách thành các skill `/migration-create` và `/migration-apply`, thêm hướng dẫn validation để yêu cầu tên migration nếu thiếu, và dùng các phạm vi `allowed-tools` khác nhau cho mỗi skill.
- D) Thêm hướng dẫn validation trong SKILL.md của skill để đảm bảo `$ARGUMENTS` là một tên hợp lệ, thêm các prompt để bỏ qua ngữ cảnh cuộc trò chuyện trước, và liệt kê các thao tác bị cấm cần tránh.

**Vì sao B:** Cách này dùng ba tính năng cấu hình riêng biệt để giải quyết từng vấn đề: `argument-hint` cải thiện việc nhập đối số và giảm tình trạng thiếu đối số, `context: fork` ngăn rò rỉ ngữ cảnh từ các cuộc trò chuyện trước, và `allowed-tools` giới hạn skill chỉ ở các thao tác ghi tệp an toàn, ngăn các hành động phá hủy.

---

## Câu hỏi 40 (Kịch bản: Sinh mã với Claude Code)

**Tình huống:** Codebase của bạn chứa các khu vực có quy ước lập trình khác nhau: các component React dùng phong cách functional với hooks, các API handler dùng async/await với cách xử lý lỗi cụ thể, và các model cơ sở dữ liệu tuân theo mẫu repository. Các tệp test được phân tán khắp codebase ngay cạnh mã đang được kiểm thử (ví dụ `Button.test.tsx` cạnh `Button.tsx`), và bạn muốn mọi test đều tuân theo cùng các quy ước bất kể vị trí.

**Cách được hỗ trợ tốt nhất để đảm bảo Claude tự động áp dụng đúng các quy ước khi sinh mã là gì?**

- A) Đặt tất cả các quy ước trong CLAUDE.md gốc dưới các tiêu đề cho mỗi khu vực và dựa vào Claude để suy ra phần nào áp dụng.
- B) Tạo các skill trong `.claude/skills/` cho mỗi loại mã, nhúng các quy ước vào từng SKILL.md.
- C) Đặt một tệp CLAUDE.md riêng trong mỗi thư mục con chứa các quy ước cho khu vực đó.
- D) Tạo các tệp quy tắc dưới `.claude/rules/` với YAML frontmatter chỉ định các mẫu glob để áp dụng quy ước một cách có điều kiện dựa trên đường dẫn tệp. **[ĐÁP ÁN ĐÚNG]**

**Vì sao D:** Các tệp `.claude/rules/` với YAML frontmatter và các mẫu glob (ví dụ `**/*.test.tsx`, `src/api/**/*.ts`) cho phép áp dụng quy ước một cách deterministic, theo đường dẫn, bất kể cấu trúc thư mục. Đây là cách được hỗ trợ tốt nhất cho các mẫu cắt ngang như các tệp test phân tán.

---

## Câu hỏi 41 (Kịch bản: Sinh mã với Claude Code)

**Tình huống:** Bạn muốn tạo một slash command tùy chỉnh `/review` chạy checklist review mã chuẩn của nhóm. Nó phải sẵn có cho mọi lập trình viên khi họ clone hoặc cập nhật repository.

**Bạn nên tạo tệp lệnh ở đâu?**

- A) Trong `~/.claude/commands/` ở thư mục home của mỗi lập trình viên.
- B) Trong repository dự án dưới `.claude/commands/`. **[ĐÁP ÁN ĐÚNG]**
- C) Trong `.claude/config.json` dưới dạng một mảng các lệnh.
- D) Trong CLAUDE.md gốc của dự án.

**Vì sao B:** Đặt các slash command tùy chỉnh dưới `.claude/commands/` bên trong repository dự án đảm bảo chúng được quản lý phiên bản và tự động sẵn có cho mọi lập trình viên clone hoặc cập nhật repo. Đây là vị trí dự định cho các lệnh tùy chỉnh cấp dự án trong Claude Code.

---

## Câu hỏi 42 (Kịch bản: Sinh mã với Claude Code)

**Tình huống:** Tệp CLAUDE.md của nhóm bạn đã vượt quá 500 dòng, trộn lẫn các quy ước TypeScript, hướng dẫn kiểm thử, các mẫu API và quy trình triển khai. Các lập trình viên thấy khó tìm và cập nhật đúng phần cần thiết.

**Claude Code hỗ trợ cách tiếp cận nào để tổ chức các hướng dẫn cấp dự án thành các module chủ đề tập trung?**

- A) Định nghĩa một tệp ánh xạ `.claude/config.yaml` gắn các mẫu tệp với các phần cụ thể bên trong CLAUDE.md.
- B) Tạo các tệp Markdown riêng trong `.claude/rules/`, mỗi tệp bao quát một chủ đề (ví dụ `testing.md`, `api-conventions.md`). **[ĐÁP ÁN ĐÚNG]**
- C) Tách các hướng dẫn thành các tệp README.md trong các thư mục con liên quan mà Claude tự động tải làm hướng dẫn.

- D) Tạo nhiều tệp tên CLAUDE.md ở các cấp khác nhau của cây thư mục, mỗi tệp ghi đè các hướng dẫn của tệp cha.

**Vì sao B:** Claude Code hỗ trợ một thư mục `.claude/rules/` nơi bạn có thể tạo các tệp Markdown riêng cho hướng dẫn theo chủ đề (ví dụ `testing.md`, `api-conventions.md`), cho phép các nhóm tổ chức những bộ hướng dẫn lớn thành các module tập trung, dễ bảo trì.

---

## Câu hỏi 43 (Kịch bản: Sinh mã với Claude Code)

**Tình huống:** Bạn tạo một skill tùy chỉnh `/explore-alternatives` mà nhóm bạn dùng để brainstorm và đánh giá các cách tiếp cận triển khai trước khi chọn một cách. Các lập trình viên báo cáo rằng sau khi chạy skill, các phản hồi tiếp theo của Claude bị ảnh hưởng bởi cuộc thảo luận về các phương án—đôi khi tham chiếu đến các cách tiếp cận đã bị loại bỏ hoặc giữ lại ngữ cảnh khám phá gây cản trở việc triển khai thực tế.

**Bạn nên cấu hình skill này hiệu quả nhất như thế nào?**

- A) Dùng tiền tố `!` trong skill để chạy logic khám phá như một tiến trình con bash.
- B) Thêm `context: fork` trong frontmatter của skill. **[ĐÁP ÁN ĐÚNG]**
- C) Tách thành hai skill—`/explore-start` và `/explore-end`—để đánh dấu ranh giới khi nào ngữ cảnh khám phá nên được loại bỏ.
- D) Tạo skill trong `~/.claude/skills/` thay vì `.claude/skills/`.

**Vì sao B:** `context: fork` chạy skill trong một context subagent tách biệt nên các cuộc thảo luận khám phá không làm ô nhiễm lịch sử cuộc trò chuyện chính. Điều này ngăn các cách tiếp cận đã bị loại bỏ và ngữ cảnh brainstorm ảnh hưởng đến công việc triển khai tiếp theo.

---

## Câu hỏi 44 (Kịch bản: Sinh mã với Claude Code)

**Tình huống:** Nhóm của bạn muốn thêm một GitHub MCP server để tìm kiếm PR và kiểm tra trạng thái CI qua Claude Code. Mỗi người trong sáu lập trình viên đều có token truy cập GitHub cá nhân riêng. Bạn muốn có bộ công cụ nhất quán trên toàn nhóm mà không commit thông tin xác thực vào hệ thống quản lý phiên bản.

**Cách tiếp cận cấu hình nào hiệu quả nhất?**

- A) Để mỗi lập trình viên tự thêm server ở phạm vi người dùng qua `claude mcp add --scope user`.
- B) Tạo một wrapper MCP server đọc token từ một tệp `.env` và làm proxy cho các lệnh gọi GitHub API, rồi thêm wrapper vào `.mcp.json` của dự án.
- C) Thêm server vào `.mcp.json` của dự án dùng thay thế biến môi trường (`${GITHUB_TOKEN}`) cho việc xác thực và ghi rõ biến môi trường bắt buộc trong README của dự án. **[ĐÁP ÁN ĐÚNG]**
- D) Cấu hình server ở phạm vi dự án với một token giữ chỗ, rồi bảo các lập trình viên ghi đè nó trong cấu hình cục bộ của họ.

**Vì sao C:** Một `.mcp.json` cấp dự án với thay thế biến môi trường là cách thông dụng: nó cung cấp một nguồn sự thật duy nhất được quản lý phiên bản cho cấu hình MCP trong khi cho phép mỗi lập trình viên

cung cấp thông tin xác thực qua các biến môi trường. Việc ghi tài liệu cho biến này giúp việc onboarding dễ dàng mà không commit bí mật.

---

## Câu hỏi 45 (Kịch bản: Sinh mã với Claude Code)

**Tình huống:** Bạn đang thêm các wrapper xử lý lỗi quanh các lệnh gọi API bên ngoài trên một codebase 120 tệp. Công việc có ba giai đoạn: (1) khám phá tất cả các điểm gọi và mẫu, (2) cùng nhau thiết kế cách tiếp cận xử lý lỗi, và (3) triển khai các wrapper một cách nhất quán. Trong Giai đoạn 1, Claude sinh ra đầu ra lớn liệt kê hàng trăm điểm gọi kèm ngữ cảnh, nhanh chóng lấp đầy context window trước khi việc khám phá hoàn tất.

**Cách tiếp cận nào hiệu quả nhất để hoàn thành nhiệm vụ trong khi vẫn duy trì tính nhất quán của việc triển khai?**

- A) Dùng một Explore subagent cho Giai đoạn 1 để cô lập đầu ra khám phá dài dòng và trả về một bản tóm tắt, sau đó tiếp tục Giai đoạn 2–3 trong cuộc trò chuyện chính. **[ĐÁP ÁN ĐÚNG]**
- B) Thực hiện tất cả các giai đoạn trong cuộc trò chuyện chính, định kỳ dùng `/compact` để giảm mức sử dụng ngữ cảnh khi đi qua các tệp.
- C) Chuyển sang chế độ headless với `--continue`, truyền các bản tóm tắt ngữ cảnh rõ ràng giữa các lệnh gọi batch để duy trì tính liên tục.
- D) Định nghĩa mẫu xử lý lỗi trong CLAUDE.md, rồi xử lý các tệp theo từng batch qua nhiều session, dựa vào tệp bộ nhớ chung để đảm bảo tính nhất quán.

**Vì sao A:** Một Explore subagent cô lập đầu ra khám phá dài dòng trong một context riêng và chỉ trả về một bản tóm tắt ngắn gọn cho cuộc trò chuyện chính. Điều này bảo toàn context window chính cho các giai đoạn thiết kế cộng tác và triển khai nhất quán, nơi ngữ cảnh được giữ lại có giá trị nhất.

---

## Kịch bản: Customer Support Agent

---

## Câu hỏi 46 (Kịch bản: Customer Support Agent)

**Tình huống:** Trong quá trình thử nghiệm, bạn nhận thấy agent thường gọi `get_customer` khi người dùng hỏi về trạng thái đơn hàng, mặc dù `lookup_order` mới là lựa chọn phù hợp hơn. Bạn nên kiểm tra điều gì đầu tiên để xử lý vấn đề này?

**Bạn nên kiểm tra điều gì đầu tiên?**

- A) Triển khai một classifier tiền xử lý để phát hiện các yêu cầu liên quan đến đơn hàng và định tuyến chúng trực tiếp đến `lookup_order`.
- B) Giảm số lượng tool khả dụng cho agent để đơn giản hóa việc lựa chọn.
- C) Thêm các ví dụ few-shot vào system prompt bao quát tất cả các mẫu yêu cầu đơn hàng có thể có nhằm cải thiện việc chọn tool.

- D) Kiểm tra mô tả của các tool để đảm bảo chúng phân biệt rõ ràng mục đích của từng tool. **[ĐÁP ÁN ĐÚNG]**

**Vì sao D:** Mô tả tool là đầu vào chính mà mô hình dùng để quyết định gọi tool nào. Khi một agent liên tục chọn sai tool, bước chẩn đoán đầu tiên là xác minh rằng mô tả tool phân tách rõ ràng mục đích và ranh giới sử dụng của từng tool.

---

## Câu hỏi 47 (Kịch bản: Customer Support Agent)

**Tình huống:** Agent của bạn xử lý các yêu cầu một-vấn-đề với độ chính xác 94% (ví dụ: "Tôi cần hoàn tiền cho đơn hàng #1234"). Nhưng khi khách hàng đưa nhiều vấn đề vào một tin nhắn (ví dụ: "Tôi cần hoàn tiền cho đơn hàng #1234 và cũng muốn cập nhật địa chỉ giao hàng cho đơn hàng #5678"), độ chính xác chọn tool giảm xuống còn 58%. Agent thường chỉ giải quyết một vấn đề hoặc trộn lẫn các tham số giữa các yêu cầu. Cách tiếp cận nào cải thiện độ tin cậy hiệu quả nhất cho các yêu cầu nhiều vấn đề?

**Cách tiếp cận nào hiệu quả nhất?**

- A) Triển khai một lớp tiền xử lý dùng một lời gọi mô hình riêng để phân rã các tin nhắn nhiều vấn đề thành các yêu cầu riêng biệt, xử lý độc lập từng yêu cầu và hợp nhất kết quả.
- B) Kết hợp các tool liên quan thành ít tool đa năng hơn.
- C) Thêm các ví dụ few-shot vào prompt minh họa cách suy luận đúng và trình tự gọi tool cho các yêu cầu nhiều vấn đề. **[ĐÁP ÁN ĐÚNG]**
- D) Triển khai validation phản hồi để phát hiện các câu trả lời chưa đầy đủ và tự động nhắc lại agent để giải quyết những vấn đề bị bỏ sót.

**Vì sao C:** Các ví dụ few-shot minh họa cách suy luận đúng và trình tự gọi tool cho các yêu cầu nhiều vấn đề là hiệu quả nhất bởi vì agent đã hoạt động tốt với các vấn đề đơn lẻ—thứ nó cần là hướng dẫn về mẫu để phân rã và định tuyến nhiều vấn đề cũng như giữ các tham số tách biệt.

---

## Câu hỏi 48 (Kịch bản: Customer Support Agent)

**Tình huống:** Production logs cho thấy với các yêu cầu đơn giản như "hoàn tiền cho đơn hàng #1234," agent của bạn giải quyết vấn đề trong 3–4 lần gọi tool với tỷ lệ thành công 91%. Nhưng với các yêu cầu phức tạp như "Tôi bị tính tiền hai lần, chiết khấu của tôi không được áp dụng, và tôi muốn hủy," agent trung bình cần hơn 12 lần gọi tool với tỷ lệ thành công chỉ 54%—thường điều tra các vấn đề một cách tuần tự và lấy dữ liệu khách hàng dư thừa cho mỗi vấn đề. Thay đổi nào cải thiện hiệu quả nhất việc xử lý các yêu cầu phức tạp?

**Thay đổi nào hiệu quả nhất?**

- A) Thêm các điểm kiểm tra xác minh rõ ràng giữa các giai đoạn, yêu cầu agent ghi lại tiến trình sau khi giải quyết mỗi vấn đề trước khi chuyển sang vấn đề tiếp theo.
- B) Giảm số lượng tool bằng cách kết hợp `get_customer`, `lookup_order`, và các tool liên quan đến hóa đơn thành một tool `investigate_issue` duy nhất.
- C) Phân rã yêu cầu thành các vấn đề riêng biệt, sau đó điều tra song song từng vấn đề bằng cách dùng chung context khách hàng trước khi tổng hợp một giải pháp cuối cùng. **[ĐÁP ÁN ĐÚNG]**

- D) Thêm các ví dụ few-shot vào system prompt minh họa các trình tự gọi tool lý tưởng cho nhiều kịch bản hóa đơn đa diện.

**Vì sao C:** Phân rã thành các vấn đề riêng biệt và điều tra song song với context khách hàng dùng chung khắc phục cả hai vấn đề then chốt: nó loại bỏ việc truy xuất dữ liệu dư thừa bằng cách tái sử dụng context dùng chung giữa các vấn đề và giảm tổng số vòng lặp gọi tool bằng cách điều tra song song trước khi tổng hợp một giải pháp duy nhất.

---

## Câu hỏi 49 (Kịch bản: Customer Support Agent)

**Tình huống:** Agent của bạn đạt tỷ lệ giải quyết ngay trong lần liên hệ đầu tiên là 55%, thấp hơn nhiều so với mục tiêu 80%. Logs cho thấy nó escalation các trường hợp đơn giản (thay thế tiêu chuẩn cho hàng bị hư hỏng có bằng chứng ảnh) trong khi lại cố gắng tự xử lý các tình huống phức tạp đòi hỏi ngoại lệ chính sách. Cách hiệu quả nhất để cải thiện việc hiệu chỉnh escalation là gì?

**Cách hiệu quả nhất để cải thiện việc hiệu chỉnh escalation là gì?**

- A) Yêu cầu agent tự đánh giá độ tin cậy trên thang điểm 1–10 trước mỗi phản hồi và tự động định tuyến đến con người khi độ tin cậy giảm dưới một ngưỡng.
- B) Triển khai một mô hình classifier riêng được huấn luyện trên các ticket lịch sử để dự đoán những yêu cầu nào cần escalation trước khi agent chính bắt đầu xử lý.
- C) Thêm các tiêu chí escalation rõ ràng vào system prompt cùng với các ví dụ few-shot cho thấy khi nào nên escalation so với khi nào tự giải quyết. **[ĐÁP ÁN ĐÚNG]**
- D) Triển khai phân tích cảm xúc để xác định mức độ bức xúc của khách hàng và tự động escalation khi vượt quá một ngưỡng cảm xúc tiêu cực.

**Vì sao C:** Các tiêu chí escalation rõ ràng kèm ví dụ few-shot trực tiếp giải quyết nguyên nhân gốc rễ—ranh giới quyết định không rõ ràng giữa các trường hợp đơn giản và phức tạp. Đây là can thiệp đầu tiên cân xứng và hiệu quả nhất, dạy cho agent khi nào nên escalation và khi nào tự giải quyết mà không cần thêm hạ tầng.

---

## Câu hỏi 50 (Kịch bản: Customer Support Agent)

**Tình huống:** Sau khi gọi `get_customer` và `lookup_order`, agent đã có tất cả dữ liệu hệ thống khả dụng nhưng vẫn đối mặt với sự không chắc chắn. Tình huống nào là yếu tố kích hoạt việc gọi `escalate_to_human` hợp lý nhất?

**Tình huống nào hợp lý nhất để escalation?**

- A) Một khách hàng muốn hủy một đơn hàng đã giao đi hôm qua và sẽ đến vào ngày mai. Agent nên escalation vì khách hàng có thể đổi ý sau khi nhận được gói hàng.
- B) Một khách hàng khẳng định họ không nhận được đơn hàng, nhưng thông tin theo dõi cho thấy nó đã được giao và có chữ ký nhận tại địa chỉ của họ ba ngày trước. Agent nên escalation vì việc đưa ra bằng chứng mâu thuẫn có thể làm tổn hại mối quan hệ với khách hàng.

- C) Một khách hàng yêu cầu so khớp giá với đối thủ cạnh tranh. Các chính sách của bạn cho phép điều chỉnh giá khi có giảm giá trên chính website của bạn trong vòng 14 ngày, nhưng không đề cập gì đến giá của đối thủ. Agent nên escalation để diễn giải chính sách. **[ĐÁP ÁN ĐÚNG]**
- D) Một tin nhắn của khách hàng chứa cả một câu hỏi về hóa đơn lẫn một yêu cầu trả lại sản phẩm. Agent nên escalation để một con người có thể phối hợp cả hai vấn đề trong một lần tương tác.

**Vì sao C:** Đây là một lỗ hổng chính sách thực sự: các quy tắc của công ty bao quát việc giảm giá trên chính website của bạn nhưng không đề cập đến việc so khớp giá với đối thủ. Agent không được tự đặt ra chính sách và nên escalation để con người phán xét về cách diễn giải hoặc mở rộng các quy tắc hiện có.

---

## Câu hỏi 51 (Kịch bản: Customer Support Agent)

**Tình huống:** Production logs cho thấy trong 12% các trường hợp, agent của bạn bỏ qua `get_customer` và gọi trực tiếp `lookup_order` chỉ dùng tên do khách hàng cung cấp, đôi khi dẫn đến việc nhận diện sai tài khoản và hoàn tiền sai. Thay đổi nào khắc phục vấn đề độ tin cậy này hiệu quả nhất?

**Thay đổi nào hiệu quả nhất?**

- A) Thêm các ví dụ few-shot cho thấy agent luôn gọi `get_customer` trước, ngay cả khi khách hàng tự nguyện cung cấp chi tiết đơn hàng.
- B) Triển khai một classifier định tuyến phân tích mỗi yêu cầu và chỉ kích hoạt một tập con các tool phù hợp với loại yêu cầu đó.
- C) Thêm một điều kiện tiên quyết theo lập trình chặn `lookup_order` và `process_refund` cho đến khi `get_customer` trả về một định danh khách hàng đã được xác minh. **[ĐÁP ÁN ĐÚNG]**
- D) Củng cố system prompt khẳng định rằng việc xác minh khách hàng qua `get_customer` là bắt buộc trước bất kỳ thao tác đơn hàng nào.

**Vì sao C:** Một điều kiện tiên quyết theo lập trình cung cấp sự bảo đảm deterministic rằng trình tự yêu cầu được tuân thủ. Đây là cách tiếp cận hiệu quả nhất vì nó loại bỏ khả năng bỏ qua bước xác minh, bất kể hành vi của LLM ra sao.

---

## Câu hỏi 52 (Kịch bản: Customer Support Agent)

**Tình huống:** Các chỉ số production cho thấy khi giải quyết các tranh chấp hóa đơn phức tạp hoặc các trường hợp trả lại nhiều đơn hàng, điểm hài lòng của khách hàng thấp hơn 15% so với các trường hợp đơn giản—ngay cả khi giải pháp về mặt kỹ thuật là đúng. Phân tích nguyên nhân gốc rễ cho thấy agent đưa ra giải pháp chính xác nhưng giải thích lý do không nhất quán: đôi khi bỏ sót các chi tiết chính sách liên quan, đôi khi thiếu thông tin về mốc thời gian hoặc các bước tiếp theo. Các khoảng trống context cụ thể thay đổi theo từng trường hợp. Bạn muốn cải thiện chất lượng giải pháp mà không thêm sự giám sát của con người. Cách tiếp cận nào hiệu quả nhất?

**Cách tiếp cận nào hiệu quả nhất?**

- A) Thêm một giai đoạn tự phê bình, trong đó agent đánh giá bản nháp phản hồi về tính đầy đủ—đảm bảo nó giải quyết vấn đề của khách hàng, bao gồm context liên quan, và dự đoán trước các câu hỏi tiếp theo. **[ĐÁP ÁN ĐÚNG]**

- B) Thêm một giai đoạn xác nhận, trong đó agent hỏi "Điều này có giải quyết hoàn toàn vấn đề của bạn không?" trước khi đóng, cho phép khách hàng yêu cầu thêm thông tin nếu cần.
- C) Nâng cấp mô hình từ Haiku lên Sonnet cho các trường hợp phức tạp, định tuyến dựa trên một chỉ số độ phức tạp được định nghĩa.
- D) Triển khai các ví dụ few-shot trong system prompt cho thấy các giải thích đầy đủ cho năm loại trường hợp phức tạp phổ biến, minh họa cách đưa vào context chính sách, mốc thời gian, và các bước tiếp theo.

**Vì sao A:** Một giai đoạn tự phê bình (mẫu evaluator-optimizer) trực tiếp giải quyết sự không nhất quán về tính đầy đủ của phần giải thích bằng cách buộc agent đánh giá bản nháp của chính nó dựa trên các tiêu chí cụ thể—như context chính sách, mốc thời gian, và các bước tiếp theo—trước khi trình bày nó. Điều này phát hiện các khoảng trống đặc thù theo từng trường hợp mà không cần sự giám sát của con người.

---

## Câu hỏi 53 (Kịch bản: Customer Support Agent)

**Tình huống:** Các chỉ số production cho thấy agent của bạn trung bình hơn 4 vòng lặp API cho mỗi lần giải quyết. Phân tích cho thấy Claude thường yêu cầu `get_customer` và `lookup_order` trong các lượt tuần tự riêng biệt ngay cả khi cả hai đều cần ngay từ đầu. Cách hiệu quả nhất để giảm số vòng lặp là gì?

**Cách hiệu quả nhất để giảm số vòng lặp là gì?**

- A) Triển khai thực thi suy đoán tự động gọi các tool có khả năng cần dùng song song với bất kỳ tool nào được yêu cầu và trả về tất cả kết quả bất kể đã yêu cầu gì.
- B) Tăng `max_tokens` để cho Claude nhiều không gian hơn để lập kế hoạch và kết hợp các yêu cầu tool một cách tự nhiên.
- C) Tạo các tool tổng hợp như `get_customer_with_orders` gói các tổ hợp tra cứu phổ biến vào các lời gọi đơn lẻ.
- D) Hướng dẫn Claude trong prompt gói các yêu cầu tool vào một lượt và trả về tất cả kết quả cùng nhau trước lời gọi API tiếp theo. **[ĐÁP ÁN ĐÚNG]**

**Vì sao D:** Việc nhắc Claude gói các yêu cầu tool liên quan vào một lượt duy nhất tận dụng khả năng vốn có của nó để yêu cầu nhiều tool cùng lúc. Nó trực tiếp khắc phục mẫu gọi tuần tự với thay đổi kiến trúc tối thiểu.

---

## Câu hỏi 54 (Kịch bản: Customer Support Agent)

**Tình huống:** Production logs cho thấy một mẫu: khách hàng tham chiếu đến các con số cụ thể (ví dụ: "khoản chiết khấu 15% mà tôi đã đề cập"), nhưng agent phản hồi với các giá trị không chính xác. Điều tra cho thấy các chi tiết này đã được đề cập từ hơn 20 lượt trước và bị cô đọng thành các bản tóm tắt mơ hồ như "đã thảo luận về giá khuyến mãi." Cách khắc phục nào hiệu quả nhất?

**Cách khắc phục nào hiệu quả nhất?**

- A) Tăng ngưỡng tóm tắt từ 70% lên 85% để các cuộc hội thoại có nhiều không gian hơn trước khi việc tóm tắt được kích hoạt.

- B) Lưu trữ toàn bộ lịch sử hội thoại trong bộ lưu trữ bên ngoài và triển khai truy xuất khi agent phát hiện các tham chiếu như "như tôi đã đề cập."
- C) Trích xuất các dữ kiện giao dịch (số tiền, ngày tháng, mã đơn hàng) vào một khối "case facts" bền vững được đưa vào mọi prompt nằm ngoài lịch sử đã tóm tắt. **[ĐÁP ÁN ĐÚNG]**
- D) Sửa lại prompt tóm tắt để giữ nguyên văn một cách rõ ràng tất cả các con số, tỷ lệ phần trăm, ngày tháng, và các kỳ vọng do khách hàng nêu ra.

**Vì sao C:** Việc tóm tắt vốn dĩ làm mất đi các chi tiết chính xác. Trích xuất các dữ kiện giao dịch vào một khối "case facts" có cấu trúc nằm ngoài lịch sử đã tóm tắt giữ lại thông tin quan trọng để nó luôn sẵn có một cách đáng tin cậy trong mọi prompt bất kể đã tóm tắt bao nhiêu lượt.

---

## Câu hỏi 55 (Kịch bản: Customer Support Agent)

**Tình huống:** Tool `get_customer` của bạn trả về tất cả các kết quả khớp khi tìm kiếm theo tên. Hiện tại, khi có nhiều kết quả, Claude chọn khách hàng có đơn hàng gần đây nhất, nhưng dữ liệu production cho thấy điều này chọn sai tài khoản 15% số lần đối với các kết quả khớp mơ hồ. Bạn nên xử lý điều này như thế nào?

**Bạn nên xử lý điều này như thế nào?**

- A) Triển khai một hệ thống chấm điểm độ tin cậy hành động tự động khi độ tin cậy trên 85% và yêu cầu làm rõ khi dưới ngưỡng.
- B) Hướng dẫn Claude yêu cầu một định danh bổ sung (email, số điện thoại, hoặc mã đơn hàng) khi `get_customer` trả về nhiều kết quả khớp, trước khi thực hiện bất kỳ hành động nào dành riêng cho khách hàng. **[ĐÁP ÁN ĐÚNG]**
- C) Sửa đổi `get_customer` để chỉ trả về một kết quả khớp có khả năng cao nhất dựa trên một thuật toán xếp hạng, loại bỏ sự mơ hồ.
- D) Thêm các ví dụ few-shot vào prompt minh họa cách suy luận đúng và trình tự gọi tool cho các kết quả khớp mơ hồ.

**Vì sao B:** Yêu cầu người dùng cung cấp một định danh bổ sung là cách đáng tin cậy nhất để giải quyết sự mơ hồ bởi vì người dùng có kiến thức xác định về danh tính của chính họ. Một lượt hội thoại thêm là cái giá nhỏ để đổi lấy việc loại bỏ tỷ lệ lỗi 15% do chọn nhầm tài khoản.

---

## Câu hỏi 56 (Kịch bản: Customer Support Agent)

**Tình huống:** Production logs cho thấy một mẫu nhất quán: khi khách hàng đưa từ "account" vào tin nhắn của họ (ví dụ: "Tôi muốn kiểm tra account của mình cho một đơn hàng tôi đặt hôm qua"), agent gọi `get_customer` trước 78% số lần. Khi khách hàng diễn đạt các yêu cầu tương tự mà không có "account" (ví dụ: "Tôi muốn kiểm tra một đơn hàng tôi đặt hôm qua"), nó gọi `lookup_order` trước 93% số lần. Mô tả tool đã rõ ràng và không mơ hồ. Nguyên nhân gốc rễ có khả năng nhất của sự khác biệt này là gì?

**Nguyên nhân gốc rễ có khả năng nhất là gì?**

- A) System prompt chứa các hướng dẫn nhạy cảm với từ khóa điều hướng hành vi dựa trên các từ như "account," tạo ra các mẫu chọn tool ngoài ý muốn. **[ĐÁP ÁN ĐÚNG]**

- B) Quá trình huấn luyện nền tảng của mô hình tạo ra các liên kết giữa thuật ngữ "account" và các thao tác liên quan đến khách hàng, lấn át mô tả tool.
- C) Mô hình cần thêm dữ liệu huấn luyện về các tin nhắn đa khái niệm và nên được fine-tune trên các ví dụ chứa cả thuật ngữ account lẫn order.
- D) Mô tả tool cần thêm các ví dụ phủ định chỉ rõ khi nào KHÔNG dùng từng tool để ngăn sự nhầm lẫn do từ khóa gây ra này.

**Vì sao A:** Mẫu hệ thống do từ khóa dẫn dắt (78% so với 93%) cho thấy mạnh mẽ rằng có logic định tuyến tường minh trong system prompt phản ứng với từ "account" và điều hướng agent về các tool liên quan đến khách hàng. Vì mô tả tool đã rõ ràng, sự khác biệt này chỉ ra rằng các hướng dẫn ở cấp prompt đang tạo ra sự điều hướng hành vi ngoài ý muốn.

---

## Câu hỏi 57 (Kịch bản: Customer Support Agent)

**Tình huống:** Production logs cho thấy agent thường gọi `get_customer` khi người dùng hỏi về đơn hàng (ví dụ: "kiểm tra đơn hàng #12345 của tôi") thay vì gọi `lookup_order`. Cả hai tool đều có mô tả tối thiểu ("Gets customer information" / "Gets order details") và chấp nhận các định dạng định danh trông tương tự nhau. Bước đầu tiên hiệu quả nhất để cải thiện độ tin cậy của việc chọn tool là gì?

**Bước đầu tiên hiệu quả nhất là gì?**

- A) Triển khai một lớp định tuyến phân tích đầu vào của người dùng trước mỗi lượt và chọn trước tool đúng dựa trên các từ khóa được phát hiện và mẫu ID.
- B) Kết hợp cả hai tool thành một tool `lookup_entity` duy nhất chấp nhận bất kỳ định danh nào và tự quyết định bên trong nên truy vấn backend nào.
- C) Thêm các ví dụ few-shot vào system prompt minh họa các mẫu chọn tool đúng, với 5–8 ví dụ định tuyến các truy vấn liên quan đến đơn hàng đến `lookup_order`.
- D) Mở rộng mô tả của từng tool để bao gồm định dạng đầu vào, ví dụ truy vấn, edge case, và ranh giới giải thích khi nào dùng nó so với các tool tương tự. **[ĐÁP ÁN ĐÚNG]**

**Vì sao D:** Mở rộng mô tả tool với định dạng đầu vào, ví dụ truy vấn, edge case, và ranh giới rõ ràng trực tiếp khắc phục nguyên nhân gốc rễ—các mô tả tối thiểu không cung cấp cho LLM đủ thông tin để phân biệt các tool tương tự. Đây là bước đầu tiên ít công sức, tác động cao, giúp cải thiện cơ chế chính mà LLM dùng để chọn tool.

---

## Câu hỏi 58 (Kịch bản: Customer Support Agent)

**Tình huống:** Bạn đang triển khai agentic loop cho support agent của mình. Sau mỗi lời gọi Claude API, bạn phải quyết định có tiếp tục vòng lặp hay không (chạy các tool được yêu cầu và gọi Claude lần nữa) hoặc dừng lại (trình bày câu trả lời cuối cùng cho khách hàng). Điều gì quyết định quyết định này?

**Điều gì quyết định quyết định này?**

- A) Kiểm tra trường `stop_reason` trong phản hồi của Claude—tiếp tục nếu nó là `tool_use` và dừng nếu nó là `end_turn`. **[ĐÁP ÁN ĐÚNG]**

- B) Phân tích văn bản của Claude để tìm các cụm từ như "Tôi đã xong" hoặc "Tôi có thể giúp gì thêm không?"—các tín hiệu ngôn ngữ tự nhiên cho thấy nhiệm vụ đã hoàn thành.
- C) Đặt một số lần lặp tối đa (ví dụ: 10 lời gọi) và dừng khi đạt đến, bất kể Claude có cho thấy cần thêm việc hay không.
- D) Kiểm tra xem phản hồi có chứa nội dung văn bản của assistant hay không—nếu Claude tạo ra văn bản giải thích, vòng lặp nên kết thúc.

**Vì sao A:** `stop_reason` là tín hiệu có cấu trúc tường minh của Claude để điều khiển vòng lặp: `tool_use` cho thấy Claude muốn chạy một tool và nhận lại kết quả, trong khi `end_turn` cho thấy Claude đã hoàn thành phản hồi của mình và vòng lặp nên kết thúc.

---

## Câu hỏi 59 (Kịch bản: Customer Support Agent)

**Tình huống:** Production logs cho thấy agent diễn giải sai các đầu ra từ các MCP tool của bạn: Unix timestamps từ `get_customer`, ngày tháng ISO 8601 từ `lookup_order`, và các mã trạng thái dạng số (1=đang chờ, 2=đã giao). Một số tool là các MCP server của bên thứ ba mà bạn không thể chỉnh sửa. Cách tiếp cận chuẩn hóa định dạng dữ liệu nào dễ bảo trì nhất?

**Cách tiếp cận nào dễ bảo trì nhất?**

- A) Dùng một PostToolUse hook để chặn các đầu ra của tool và áp dụng các phép biến đổi định dạng trước khi agent xử lý chúng. **[ĐÁP ÁN ĐÚNG]**
- B) Chỉnh sửa các tool mà bạn kiểm soát để trả về các định dạng dễ đọc cho con người và tạo các wrapper cho các tool của bên thứ ba.
- C) Tạo một tool `normalize_data` mà agent gọi sau mỗi lần truy xuất dữ liệu để biến đổi các giá trị.
- D) Thêm tài liệu định dạng chi tiết vào system prompt giải thích các quy ước dữ liệu của từng tool.

**Vì sao A:** Một PostToolUse hook cung cấp một điểm tập trung, deterministic để chặn và chuẩn hóa tất cả các đầu ra của tool—bao gồm cả dữ liệu từ MCP server của bên thứ ba—trước khi agent xử lý chúng. Nó dễ bảo trì hơn vì các phép biến đổi nằm trong code và áp dụng đồng nhất, thay vì dựa vào sự diễn giải của LLM.

---

## Câu hỏi 60 (Kịch bản: Customer Support Agent)

**Tình huống:** Production logs cho thấy agent đôi khi chọn `get_customer` khi `lookup_order` mới phù hợp hơn, đặc biệt với các truy vấn mơ hồ như "Tôi cần trợ giúp với lần mua hàng gần đây của tôi." Bạn quyết định thêm các ví dụ few-shot vào system prompt để cải thiện việc chọn tool. Cách tiếp cận nào giải quyết vấn đề hiệu quả nhất?

**Cách tiếp cận nào hiệu quả nhất?**

- A) Thêm hướng dẫn "use when" và "don't use when" rõ ràng trong mỗi mô tả tool bao quát các trường hợp mơ hồ.
- B) Thêm các ví dụ được nhóm theo tool—tất cả các kịch bản `get_customer` cùng nhau, rồi đến tất cả các kịch bản `lookup_order`.

- C) Thêm 4–6 ví dụ nhắm vào các kịch bản mơ hồ, mỗi ví dụ kèm lý do tại sao chọn một tool thay vì các lựa chọn thay thế hợp lý khác. **[ĐÁP ÁN ĐÚNG]**
- D) Thêm 10–15 ví dụ về các yêu cầu rõ ràng, không mơ hồ minh họa việc chọn tool đúng cho các kịch bản điển hình của từng tool.

**Vì sao C:** Nhắm các ví dụ few-shot vào chính những kịch bản mơ hồ nơi xảy ra lỗi, kèm lý do tường minh tại sao một tool tốt hơn các lựa chọn thay thế, dạy cho mô hình quá trình ra quyết định so sánh cần thiết cho các edge case. Điều này hiệu quả hơn các ví dụ chung chung hoặc các quy tắc mang tính khai báo.

---

## Câu hỏi 61 (Kịch bản: Các mẫu kiến trúc AI hội thoại)

**Tình huống:** Tool `remove_team_member` của bạn dùng một tham số `dry_run: boolean` để xem trước tác động trước khi thực thi. Giám sát production cho thấy agent bỏ qua bước xem trước bằng cách gọi trực tiếp với `dry_run=false`. Bạn cần đảm bảo mọi thao tác xóa đều phải đi trước bởi một bước xem trước mà người dùng xác nhận tường minh.

**Cách tiếp cận đáng tin cậy nhất là gì?**

- A) Thêm validation phía server chỉ cho phép `dry_run=false` khi có một lời gọi `dry_run=true` với các tham số giống hệt diễn ra trong vòng 60 giây vừa qua.
- B) Chú thích tool là yêu cầu xác nhận và cấu hình lớp orchestration để nhắc người dùng phê duyệt trước khi chuyển tiếp bất kỳ lời gọi nào đến các tool đã được chú thích.
- C) Thêm hướng dẫn chi tiết và ví dụ few-shot vào mô tả tool, yêu cầu agent luôn gọi với `dry_run=true` trước và chờ người dùng xác nhận trước khi gọi lại.
- D) Thay bằng hai tool: `preview_remove_member` trả về chi tiết tác động và một confirmation token dùng một lần; `execute_remove_member` yêu cầu token đó, ràng buộc việc thực thi với bước xem trước. **[ĐÁP ÁN ĐÚNG]**

**Vì sao D:** Cách tiếp cận hai tool ràng buộc bằng token khiến việc thực thi mà không có bước xem trước trở nên bất khả thi về mặt kiến trúc—tool execute thực sự yêu cầu một token mà chỉ tool preview mới có thể tạo ra. Đây là cách tiếp cận duy nhất thực thi ràng buộc ở cấp độ code thay vì dựa vào việc LLM tuân thủ các hướng dẫn (C), heuristic về thời gian (A), hay hạ tầng orchestration (B).

---

## Câu hỏi 62 (Kịch bản: Các mẫu kiến trúc AI hội thoại)

**Tình huống:** Giám sát production cho thấy tool `search_catalog` của bạn thất bại 12% số lần: 8% là network timeout sẽ thành công khi retry, và 4% là lỗi cú pháp truy vấn không bao giờ thành công dù retry bao nhiêu lần. Hiện tại cả hai loại lỗi đều được trả về giống hệt nhau, gây lãng phí các lần retry.

**Bạn nên chỉnh sửa cơ chế xử lý lỗi của tool như thế nào?**

- A) Thêm các ví dụ few-shot vào system prompt minh họa cách phân biệt lỗi network với lỗi cú pháp.
- B) Áp dụng logic retry với exponential backoff cho mọi lỗi một cách đồng đều.
- C) Triển khai tự động retry với backoff cho network timeout bên trong tool; trả về lỗi cú pháp ngay lập tức kèm chi tiết validation tham số. **[ĐÁP ÁN ĐÚNG]**

- D) Trả về mọi lỗi kèm một cờ boolean `retryable` và chi tiết loại lỗi.

**Vì sao C:** Xử lý retry ở cấp độ tool đối với các lỗi tạm thời là ranh giới trừu tượng đúng đắn—tool có hiểu biết dứt khoát về loại lỗi và có thể triển khai logic retry deterministic mà không phải dựa vào agent để diễn giải một cờ (D) hay tuân theo các hướng dẫn ở cấp prompt (A). Backoff đồng đều (B) lãng phí thời gian cho các lỗi cú pháp vốn không bao giờ thành công.

---

## Câu hỏi 63 (Kịch bản: Các mẫu kiến trúc AI hội thoại)

**Tình huống:** Qua nhiều lượt trao đổi về chiến lược đầu tư, người dùng đã nói "Tôi có khẩu vị rủi ro rất thấp" và sau đó "Tôi muốn tối đa hóa lợi nhuận." Bây giờ họ hỏi: "Tôi nên đầu tư vào cái gì?"

**Cách tiếp cận nào đảm bảo tốt nhất rằng khuyến nghị phù hợp với ưu tiên thực sự của người dùng?**

- A) Nêu ra mâu thuẫn và hỏi người dùng để làm rõ điều nào quan trọng hơn. **[ĐÁP ÁN ĐÚNG]**
- B) Cung cấp các khuyến nghị riêng biệt cho cả hai kịch bản.
- C) Tiến hành theo sở thích được nêu gần đây nhất.
- D) Khuyến nghị một danh mục cân bằng mà không giải quyết xung đột.

**Vì sao A:** Khi các sở thích của người dùng mâu thuẫn trực tiếp với nhau, nêu ra xung đột và hỏi để làm rõ là cách duy nhất đảm bảo khuyến nghị phù hợp với ý định thực sự của người dùng. Bất kỳ cách tiếp cận nào khác đều liên quan đến việc đưa ra một giả định có thể sai—tối đa hóa lợi nhuận và khẩu vị rủi ro thấp là những mục tiêu về cơ bản không tương thích, đòi hỏi một quyết định của con người.

---

## Câu hỏi 64 (Kịch bản: Các mẫu kiến trúc AI hội thoại)

**Tình huống:** Người dùng tinh chỉnh sở thích playlist qua nhiều lượt hội thoại. Hai tin nhắn sau khi người dùng nói "Tôi yêu nhạc jazz," Claude lại hỏi "Bạn thích những thể loại nào?"

**Nguyên nhân có khả năng nhất là gì?**

- A) Claude yêu cầu một kết nối vector database để duy trì bộ nhớ hội thoại.
- B) Context window của mô hình đã bị vượt quá.
- C) Claude API yêu cầu một tham số `session_id`.
- D) Ứng dụng của bạn không bao gồm các tin nhắn trước đó trong mảng `messages`. **[ĐÁP ÁN ĐÚNG]**

**Vì sao D:** Claude không có bộ nhớ phía server—mỗi lời gọi API đều stateless. Nếu không bao gồm toàn bộ lịch sử hội thoại trong mảng `messages` của mỗi request, Claude không có hiểu biết gì về các lượt trước đó. Vector database (A) và `session_id` (C) không phải là một phần trong kiến trúc của Claude; việc tràn context window (B) là bất khả thi đối với cuộc trao đổi chỉ có hai tin nhắn.

---

## Câu hỏi 65 (Kịch bản: Các mẫu kiến trúc AI hội thoại)

**Tình huống:** Sau một phiên nấu ăn kéo dài 40 phút, cuộc hội thoại đạt 78.000 token. Lịch sử bao gồm các dị ứng, việc điều chỉnh tỷ lệ công thức, các thuật ngữ nấu ăn đã được làm rõ, và thảo luận chung. Bạn phải giảm số token trong khi vẫn bảo toàn thông tin quan trọng.

**Cách tiếp cận nào cân bằng tốt nhất giữa việc bảo toàn và việc giảm token?**

- A) Tóm tắt toàn bộ lịch sử hội thoại.
- B) Chỉ giữ lại 20.000 token gần đây nhất.
- C) Trích xuất dữ liệu có cấu trúc trọng yếu (dị ứng, số lượng, sở thích), tóm tắt phần thảo luận chung, và giữ nguyên văn các cuộc trao đổi gần đây. **[ĐÁP ÁN ĐÚNG]**
- D) Lưu toàn bộ cuộc hội thoại ở bên ngoài và truy xuất các phần liên quan qua semantic search.

**Vì sao C:** Cách tiếp cận lai bảo toàn thông tin có giá trị cao nhất với chi phí thấp nhất. Các sự kiện trọng yếu như dị ứng và số lượng trong công thức được trích xuất thành một khối có cấu trúc gọn gàng (ngăn ngừa sự mất mát độ chính xác xảy ra trong quá trình tóm tắt), phần thảo luận chung được tóm tắt, và các cuộc trao đổi gần đây được giữ nguyên văn để đảm bảo tính mạch lạc của hội thoại. Phương án A và B có nguy cơ làm mất thông tin ăn kiêng trọng yếu; D là quá mức về mặt kiến trúc đối với một phiên nấu ăn đơn lẻ.

---

## Câu hỏi 66 (Kịch bản: Các mẫu kiến trúc AI hội thoại)

**Tình huống:** Người dùng báo cáo rằng trong các cuộc hội thoại kéo dài, trợ lý mất dấu các chủ đề và sở thích trước đó. Triển khai hiện tại của bạn chỉ giữ lại 25 cặp tin nhắn gần nhất.

**Giải pháp hiệu quả nhất là gì?**

- A) Cách tiếp cận lai: tóm tắt các tin nhắn cũ hơn trong khi giữ nguyên văn các tin nhắn gần đây. **[ĐÁP ÁN ĐÚNG]**
- B) Vector similarity search trên toàn bộ lịch sử hội thoại.
- C) Tăng cửa sổ lên 50 cặp tin nhắn.
- D) Tóm tắt các tin nhắn bị loại bỏ ở mỗi lượt và đặt bản tóm tắt đang chạy lên đầu.

**Vì sao A:** Cách tiếp cận lai giải quyết cả hai chiều của vấn đề: giữ lại context gần đây chính xác (then chốt cho tính mạch lạc của hội thoại) trong khi duy trì một biểu diễn nén của các sở thích trước đó (ngăn ngừa mất mát hoàn toàn khi các cặp bị loại bỏ). Tăng cửa sổ (C) chỉ đơn giản là trì hoãn chính vấn đề đó. Vector search (B) có thể bỏ sót context quan trọng vốn không tương đồng ngữ nghĩa với truy vấn hiện tại. Tóm tắt toàn bộ ở mỗi lượt (D) làm tăng chi phí xử lý và tích lũy các lỗi tóm tắt.

---

## Câu hỏi 67 (Kịch bản: Các mẫu kiến trúc AI hội thoại)

**Tình huống:** Người dùng báo cáo rằng latency tăng và chi phí tăng khi các cuộc hội thoại vượt quá 50 lượt.

**Nguyên nhân chính là gì?**

- A) Toàn bộ lịch sử hội thoại được bao gồm trong mỗi request API. **[ĐÁP ÁN ĐÚNG]**
- B) Mô hình tạo ra các phản hồi dài dần lên.
- C) Các thao tác database chậm đi khi lịch sử tăng lên.
- D) Mô hình xây dựng một hồ sơ người dùng nội bộ đòi hỏi nhiều xử lý hơn.

**Vì sao A:** API của Claude hoàn toàn stateless—mỗi request phải bao gồm toàn bộ lịch sử hội thoại trong mảng `messages`. Khi các cuộc hội thoại lớn dần, mỗi request mang theo nhiều token hơn, điều này trực tiếp làm tăng cả latency xử lý lẫn chi phí. Mô hình không duy trì bất kỳ trạng thái nội bộ nào giữa các lời gọi (D là sai), và độ dài phản hồi vốn không gắn liền với độ dài hội thoại (B).

---

## Câu hỏi 68 (Kịch bản: Các mẫu kiến trúc AI hội thoại)

**Tình huống:** Sau ba tháng với các phiên hàng tuần, lịch sử hội thoại tăng lên 85.000 token. Khi người dùng hỏi "Chúng ta đã kết luận gì về chủ đề sự cô lập?", trợ lý đưa ra các câu trả lời chung chung thay vì tham chiếu đến các cuộc thảo luận trước đó.

**Cách tiếp cận hiệu quả nhất là gì?**

- A) Cắt bớt theo cửa sổ trượt.
- B) Tóm tắt lũy tiến nắm bắt các kết luận then chốt.
- C) Semantic embedding cùng với việc truy xuất các cuộc trao đổi liên quan. **[ĐÁP ÁN ĐÚNG]**
- D) Thêm các thẻ XML có cấu trúc đánh dấu các kết luận thảo luận.

**Vì sao C:** Semantic search trên lịch sử hội thoại là cách tiếp cận duy nhất có thể mở rộng cho ba tháng thảo luận trong khi vẫn có thể đưa ra các cuộc trao đổi liên quan cụ thể theo yêu cầu. Cửa sổ trượt (A) sẽ loại bỏ phần lớn lịch sử. Tóm tắt lũy tiến (B) nén các cuộc thảo luận thành những khái niệm trừu tượng làm mất đi các kết luận cụ thể mà người dùng đang hỏi. Thẻ XML (D) đòi hỏi tái cấu trúc toàn bộ nội dung quá khứ và không giải quyết được vấn đề truy xuất ở quy mô này.

---

## Câu hỏi 69 (Kịch bản: Các mẫu kiến trúc AI hội thoại)

**Tình huống:** Trong quá trình kiểm thử QA, Claude tuân theo các hướng dẫn của system prompt trong 10–15 lượt đầu tiên, nhưng các phản hồi về sau lại lệch đi. Cuộc hội thoại vẫn nằm trong giới hạn token.

**Giải pháp tốt nhất là gì?**

- A) Chuyển các hướng dẫn về hành vi vào tin nhắn đầu tiên của người dùng.
- B) Bắt đầu một cuộc hội thoại mới sau 20 lượt.
- C) Chèn các tin nhắn vai trò người dùng nhằm củng cố các hướng dẫn tại những điểm ngắt của cuộc hội thoại. **[ĐÁP ÁN ĐÚNG]**
- D) Dùng validation sau phản hồi để tạo lại các phản hồi không tuân thủ.

**Vì sao C:** Việc chèn định kỳ các lời nhắc về hành vi trực tiếp chống lại hiện tượng trôi hướng dẫn bằng cách tái thiết lập các ràng buộc theo những khoảng đều đặn khi lịch sử hội thoại tích lũy. Chuyển các hướng dẫn vào tin nhắn đầu tiên của người dùng (A) làm giảm thẩm quyền của chúng. Bắt đầu một cuộc

hội thoại mới (B) phá hủy context. Validation sau phản hồi (D) mang tính khắc phục hơn là phòng ngừa và làm tăng latency đáng kể.

---

## Câu hỏi 70 (Kịch bản: Các mẫu kiến trúc AI hội thoại)

**Tình huống:** Gia sư AI của bạn có một system prompt 2.800 token định nghĩa phương pháp giảng dạy và các quy tắc thích ứng. Sau 12 lượt, trợ lý bắt đầu phớt lờ các cấp độ trình độ.

**Cách khắc phục hiệu quả nhất là gì?**

- A) Chèn các lời nhắc mỗi 4–5 lượt.
- B) Thay các quy tắc dài dòng bằng các ví dụ few-shot minh họa việc thích ứng theo cấp độ trình độ. **[ĐÁP ÁN ĐÚNG]**
- C) Đặt các quy tắc trọng yếu ở cuối system prompt.
- D) Đánh giá các phản hồi và tạo lại nếu cấp độ khó không khớp.

**Vì sao B:** Một system prompt 2.800 token với các quy tắc khai báo dễ bị trôi vì các quy tắc trừu tượng đòi hỏi mô hình phải suy luận về chúng ở mỗi lượt. Thay các quy tắc dài dòng bằng các ví dụ few-shot cụ thể minh họa việc thích ứng theo cấp độ trình độ đúng đắn sẽ cung cấp cho mô hình những mẫu hành vi rõ ràng để khớp theo—điều này được tuân theo đáng tin cậy hơn qua nhiều lượt so với các hướng dẫn trừu tượng. Việc chèn lời nhắc (A) có ích nhưng chỉ giải quyết triệu chứng; đặt ở cuối (C) có ích lúc đầu nhưng không xử lý được hiện tượng trôi theo từng lượt; việc tạo lại (D) tốn kém và mang tính khắc phục.

---

## Câu hỏi 71 (Kịch bản: Các mẫu kiến trúc AI hội thoại)

**Tình huống:** Trợ lý của bạn phải duy trì giọng điệu hào hứng, giải thích lập luận của mình, và đặt các câu hỏi làm rõ. Những hướng dẫn về hành vi này nên được định nghĩa ở đâu?

**Những hướng dẫn về hành vi này nên được định nghĩa ở đâu?**

- A) Đặt lên đầu mỗi tin nhắn của người dùng.
- B) Trong system prompt. **[ĐÁP ÁN ĐÚNG]**
- C) Trong tin nhắn assistant đầu tiên.
- D) Trong các biến môi trường.

**Vì sao B:** System prompt được thiết kế chuyên biệt cho các ràng buộc và hướng dẫn về hành vi mang tính bền vững, áp dụng xuyên suốt toàn bộ cuộc hội thoại. Đặt lên đầu mỗi tin nhắn của người dùng (A) là chi phí thừa thãi và dư thừa. Tin nhắn assistant đầu tiên (C) không đáng tin cậy vì mô hình có thể đi chệch khỏi chính các phát biểu trước đó của nó. Các biến môi trường (D) không có tác động gì đến hành vi của mô hình.

---

## Câu hỏi 72 (Kịch bản: Các mẫu kiến trúc AI hội thoại)

**Tình huống:** Người dùng báo cáo các phần mở đầu phản hồi lặp đi lặp lại như "Certainly!" và "I'd be happy to help!"

**Cách tiếp cận hiệu quả nhất là gì?**

- A) Nối thêm một tin nhắn assistant một phần với phần mở đầu trả lời trực tiếp. **[ĐÁP ÁN ĐÚNG]**
- B) Hạ thấp thiết lập temperature.
- C) Hậu xử lý các phản hồi để loại bỏ lời chào.
- D) Thêm hướng dẫn vào system prompt để tránh các cụm từ đó.

**Vì sao A:** Việc prefill phản hồi của assistant bằng phần khởi đầu của một câu trả lời trực tiếp ngăn chặn các mẫu lời chào ngay ở cấp độ sinh—mô hình tiếp tục từ phần prefill thay vì tạo ra các cụm mở đầu mới. Hướng dẫn trong system prompt (D) có thể có ích nhưng kém tin cậy hơn vì mô hình vẫn có thể tạo ra các biến thể. Hậu xử lý (C) là một cách lách dễ vỡ. Temperature (B) kiểm soát tính ngẫu nhiên, chứ không phải các mẫu cụm từ cụ thể.

---

## Câu hỏi 73 (Kịch bản: Các mẫu kiến trúc AI hội thoại)

**Tình huống:** Một webhook thông báo cho hệ thống của bạn rằng kiện hàng của người dùng đã được giao đi trong khi người dùng đang trò chuyện. Bạn muốn trợ lý đưa thông tin này vào phản hồi tiếp theo một cách tự nhiên.

**Cách tiếp cận tốt nhất là gì?**

- A) Thêm trạng thái giao hàng vào system prompt.
- B) Gửi ngay một tin nhắn người dùng tổng hợp.
- C) Buộc trợ lý gọi một status tool ở mỗi lượt.
- D) Nối phần cập nhật trạng thái như một tiền tố cho tin nhắn tiếp theo của người dùng. **[ĐÁP ÁN ĐÚNG]**

**Vì sao D:** Đặt phần cập nhật trạng thái làm tiền tố cho tin nhắn tiếp theo của người dùng sẽ chèn context thời gian thực tại một ranh giới hội thoại tự nhiên mà không làm gián đoạn dòng chảy. Chỉnh sửa system prompt (A) đòi hỏi xây dựng lại session hoặc cổng kênh về mặt kiến trúc. Một tin nhắn người dùng tổng hợp (B) có thể phá vỡ dòng đối thoại tự nhiên và gây nhầm lẫn về nguồn gốc phát ngôn. Buộc gọi một tool ở mỗi lượt (C) là lãng phí khi các sự kiện hiếm khi xảy ra.

---

## Câu hỏi 74 (Kịch bản: Các mẫu kiến trúc AI hội thoại)

**Tình huống:** Người dùng thường xuyên gửi các yêu cầu như "Đặt một địa điểm cho bữa tiệc." Trợ lý đặt từ 4 câu hỏi làm rõ trở lên, gây ra tỷ lệ bỏ cuộc 35%.

**Cách tiếp cận nào cải thiện trade-off tốt nhất?**

- A) Tiến hành với các giá trị mặc định ẩn.

- B) Hỏi tất cả các câu hỏi làm rõ trong một tin nhắn gộp.
- C) Nêu các giả định một cách tường minh và tiến hành trong khi mời người dùng đính chính. **[ĐÁP ÁN ĐÚNG]**
- D) Sử dụng một biểu mẫu thu thập thông tin có cấu trúc.

**Vì sao C:** Nêu các giả định một cách tường minh rồi tiến hành sẽ mang lại cho người dùng một phản hồi tức thì, hữu ích trong khi vẫn giữ được khả năng đính chính các giả định sai của họ. Các giá trị mặc định ẩn (A) khiến người dùng không hề biết điều gì đã được giả định. Một danh sách câu hỏi gộp (B) vẫn đòi hỏi nỗ lực trả lời trước từ người dùng. Một biểu mẫu có cấu trúc (D) tạo thêm ma sát chứ không phải bớt đi—trái với mục tiêu giảm tỷ lệ bỏ cuộc.

---

## Câu hỏi 75 (Kịch bản: Các mẫu kiến trúc AI hội thoại)

**Tình huống:** Trợ lý của bạn dùng một system prompt với persona nhà thầu. Các lượt đầu tuân theo các quy tắc, nhưng đến lượt 7 trợ lý lại đưa ra lời khuyên chung chung. Độ dài hội thoại chỉ mới 2.500 token.

**Nguyên nhân có khả năng nhất là gì?**

- A) System prompt chỉ thiết lập hành vi ban đầu.
- B) Sự chú ý của mô hình suy yếu khi các lượt tích lũy.
- C) Các phản hồi assistant tích lũy làm loãng ảnh hưởng của system prompt. **[ĐÁP ÁN ĐÚNG]**
- D) System prompt chỉ được gửi đi một lần.

**Vì sao C:** Khi các phản hồi assistant tích lũy trong lịch sử hội thoại, tỷ lệ văn bản phản ánh các ràng buộc về hành vi của system prompt giảm đi so với khối lượng ngày càng tăng của nội dung do assistant tạo ra. Mô hình ngày càng khớp mẫu theo các đầu ra trước đó của chính nó thay vì theo system prompt, làm trầm trọng thêm hiện tượng trôi ngay cả ở độ dài token ngắn. System prompt được bao gồm trong mỗi lời gọi API (D là sai nếu xét như một lời giải thích độc lập), và sự suy giảm chú ý của mô hình (B) không xảy ra ở mức 2.500 token.

---

## Câu hỏi 76 (Kịch bản: Các mẫu kiến trúc AI hội thoại)

**Tình huống:** Người dùng đặt các yêu cầu mơ hồ như "Bạn có thể giúp với bản báo cáo không?" Trợ lý phản hồi bằng cách đặt nhiều câu hỏi (báo cáo nào? giúp gì? hạn chót khi nào?), gây ra tỷ lệ bỏ cuộc 40%.

**Giải pháp tốt nhất là gì?**

- A) Đưa ra các giả định hợp lý, nêu chúng một cách tường minh, và đề nghị điều chỉnh. **[ĐÁP ÁN ĐÚNG]**
- B) Phân loại mức độ mơ hồ bằng một mô hình nhỏ hơn trước khi phản hồi.
- C) Sử dụng các cách diễn giải định trước mà không nêu các giả định.
- D) Giới hạn trợ lý ở một câu hỏi làm rõ mỗi lượt.

**Vì sao A:** Tiến hành với các giả định hợp lý đã được nêu rõ sẽ loại bỏ hoàn toàn việc qua lại trong khi vẫn giữ cho người dùng được thông tin và nắm quyền kiểm soát. Các cách diễn giải định trước âm thầm (C)

khiến người dùng bối rối khi phản hồi không khớp với ý định của họ. Một giới hạn một câu hỏi (D) vẫn đòi hỏi nhiều lượt qua lại. Một mô hình phân loại nhỏ hơn (B) làm tăng latency và độ phức tạp hạ tầng mà không giải quyết được vấn đề UX cốt lõi.

---

# Bài tập thực hành

## Bài tập 1: Agent đa công cụ với logic escalation

**Mục tiêu:** Thiết kế một agent loop với tích hợp tool, xử lý lỗi có cấu trúc và escalation.

**Các bước:**

1. Định nghĩa 3–4 MCP tool kèm mô tả chi tiết (bao gồm hai tool tương tự nhau để kiểm tra việc lựa chọn tool)
2. Triển khai một agent loop kiểm tra `stop_reason` (`"tool_use"` / `"end_turn"`)
3. Bổ sung các phản hồi lỗi có cấu trúc: `errorCategory`, `isRetryable`, mô tả
4. Triển khai một interceptor hook chặn các thao tác vượt ngưỡng và định tuyến tới escalation
5. Kiểm thử với các yêu cầu đa khía cạnh

**Lĩnh vực:** 1 (Kiến trúc agent), 2 (Tools và MCP), 5 (Context và độ tin cậy)

---

## Bài tập 2: Cấu hình Claude Code cho phát triển theo nhóm

**Mục tiêu:** Cấu hình CLAUDE.md, custom command, quy tắc theo đường dẫn và MCP server.

**Các bước:**

1. Tạo một CLAUDE.md ở cấp dự án với các tiêu chuẩn áp dụng chung
2. Tạo các file `.claude/rules/` với YAML frontmatter cho những vùng code khác nhau (`paths: ["src/api/**/*"]`, `paths: ["**/*.test.*"]`)
3. Tạo một project skill trong `.claude/skills/` với `context: fork` và `allowed-tools`
4. Cấu hình một MCP server trong `.mcp.json` với biến môi trường + một override cá nhân trong `~/.claude.json`
5. Kiểm thử planning mode so với thực thi trực tiếp trên các tác vụ có độ phức tạp khác nhau

**Lĩnh vực:** 3 (Cấu hình Claude Code), 2 (Tools và MCP)

---

## Bài tập 3: Pipeline trích xuất dữ liệu có cấu trúc

**Mục tiêu:** JSON schema, `tool_use` cho structured output, các vòng lặp validation/retry, batch processing.

**Các bước:**

1. Định nghĩa một extraction tool với một JSON schema (trường bắt buộc/tùy chọn, enum kèm "other", trường nullable)
2. Xây dựng một vòng lặp validation: khi gặp lỗi, retry với tài liệu, kết quả trích xuất sai và lỗi validation cụ thể
3. Bổ sung các ví dụ few-shot cho những tài liệu có cấu trúc khác nhau
4. Sử dụng batch processing thông qua Message Batches API: 100 tài liệu, xử lý các lần thất bại qua `custom_id`
5. Định tuyến tới con người: điểm độ tin cậy ở cấp trường, phân tích loại tài liệu

**Lĩnh vực:** 4 (Prompt engineering), 5 (Context và độ tin cậy)

---

## Bài tập 4: Thiết kế và debug một pipeline nghiên cứu đa agent

**Mục tiêu:** Điều phối subagent, truyền context, lan truyền lỗi, tổng hợp kèm theo dõi nguồn.

**Các bước:**

1. Một coordinator với 2+ subagent (`allowedTools` bao gồm `"Task"`, context được truyền tường minh trong prompt)
2. Chạy các subagent song song qua nhiều lời gọi `Task` trong một response duy nhất
3. Yêu cầu subagent xuất kết quả có cấu trúc: nhận định, trích dẫn, source URL, ngày xuất bản
4. Mô phỏng một subagent timeout: trả về context lỗi có cấu trúc cho coordinator và tiếp tục với kết quả từng phần
5. Kiểm thử với dữ liệu mâu thuẫn: giữ lại cả hai giá trị kèm ghi nguồn; tách biệt các phát hiện đã xác nhận và đang tranh cãi

**Lĩnh vực:** 1 (Kiến trúc agent), 2 (Tools và MCP), 5 (Context và độ tin cậy)

---

# Phụ lục: Công nghệ và khái niệm

| Công nghệ | Khía cạnh chính |
|---|---|
| **Claude Agent SDK** | AgentDefinition, agent loop, `stop_reason`, hook (PostToolUse), tạo subagent qua Task, `allowedTools` |
| **Model Context Protocol (MCP)** | MCP server, tool, resource, `isError`, mô tả tool, `.mcp.json`, biến môi trường |
| **Claude Code** | Phân cấp CLAUDE.md, `.claude/rules/` với mẫu glob, `.claude/commands/`, `.claude/skills/` với SKILL.md, planning mode, `/compact`, `--resume`, `fork_session` |
| **Claude Code CLI** | `-p` / `--print` cho chế độ non-interactive, `--output-format json`, `--json-schema` |

| | |
|---|---|
| **Claude API** | `tool_use` với JSON schema, `tool_choice` ("auto"/"any"/forced), `stop_reason`, `max_tokens`, system prompt |
| **Message Batches API** | tiết kiệm 50%, cửa sổ tối đa 24 giờ, `custom_id`, không hỗ trợ gọi tool nhiều lượt |
| **JSON Schema** | Bắt buộc so với tùy chọn, trường nullable, kiểu enum, "other" + chi tiết, strict mode |
| **Pydantic** | Validation schema, lỗi ngữ nghĩa, vòng lặp validation/retry |
| **Built-in tool** | Read, Write, Edit, Bash, Grep, Glob — mục đích và tiêu chí lựa chọn |
| **Few-shot prompting** | Ví dụ có chủ đích cho các tình huống mơ hồ, khái quát hóa sang mẫu mới |
| **Prompt chaining** | Phân rã tuần tự thành các lượt tập trung |
| **Context window** | Ngân sách token, tóm tắt lũy tiến, "lost in the middle", file scratchpad |
| **Quản lý session** | Resume, `fork_session`, session đặt tên, cô lập context |
| **Hiệu chỉnh độ tin cậy** | Chấm điểm cấp trường, hiệu chỉnh trên tập có nhãn, lấy mẫu phân tầng |

---

# Các chủ đề ngoài phạm vi

Các chủ đề liên quan sau đây sẽ **KHÔNG** xuất hiện trong bài thi:

- Fine-tuning các model Claude hoặc huấn luyện model tùy chỉnh
- Xác thực Claude API, thanh toán hoặc quản lý tài khoản
- Triển khai chi tiết bằng các ngôn ngữ lập trình hoặc framework cụ thể (vượt ngoài những gì cần thiết để cấu hình tool/schema)
- Triển khai hoặc hosting MCP server (hạ tầng, mạng, điều phối container)
- Kiến trúc nội bộ của Claude, quá trình huấn luyện hoặc model weight
- Constitutional AI, RLHF hoặc các phương pháp huấn luyện an toàn
- Embedding model hoặc chi tiết triển khai vector database
- Computer use (tự động hóa trình duyệt, tương tác desktop)
- Khả năng phân tích hình ảnh (Vision)
- Streaming API hoặc server-sent event
- Rate limiting, hạn ngạch, hoặc tính toán chi tiết chi phí API
- OAuth, xoay vòng API key, hoặc chi tiết giao thức xác thực
- Các cấu hình riêng theo nhà cung cấp cloud (AWS, GCP, Azure)
- Benchmark hiệu năng hoặc chỉ số so sánh model
- Chi tiết triển khai prompt caching (vượt ngoài việc biết nó tồn tại)
- Thuật toán đếm token hoặc các chi tiết tokenization

---

# Khuyến nghị ôn tập

1. **Xây dựng một agent với Claude Agent SDK** — triển khai một agent loop hoàn chỉnh với gọi tool, xử lý lỗi và quản lý session. Thực hành với subagent và truyền context tường minh.
2. **Cấu hình Claude Code cho một dự án thực tế** — sử dụng phân cấp CLAUDE.md, quy tắc theo đường dẫn trong `.claude/rules/`, skill với `context: fork` và `allowed-tools`, và tích hợp MCP server.
3. **Thiết kế và kiểm thử các MCP tool** — viết mô tả phân biệt các tool tương tự nhau, trả về lỗi có cấu trúc kèm phân loại và cờ retry, và kiểm thử với các yêu cầu mơ hồ của người dùng.
4. **Xây dựng một pipeline trích xuất dữ liệu** — sử dụng `tool_use` với JSON schema, vòng lặp validation/retry, trường tùy chọn/nullable, và batch processing qua Message Batches API.
5. **Thực hành prompt engineering** — bổ sung ví dụ few-shot cho các kịch bản mơ hồ, tiêu chí đánh giá tường minh, và kiến trúc nhiều lượt cho việc review code lớn.
6. **Nghiên cứu các mẫu quản lý context** — trích xuất dữ kiện từ output dài dòng, dùng file scratchpad, và ủy thác việc khám phá cho subagent để xử lý giới hạn context.
7. **Hiểu về escalation và human-in-the-loop** — khi nào nên escalate (lỗ hổng chính sách, yêu cầu tường minh từ người dùng, không thể tiến triển) và các quy trình định tuyến dựa trên độ tin cậy.
8. **Làm một bài thi thử** trước khi thi thật. Nó dùng cùng các kịch bản và định dạng.