

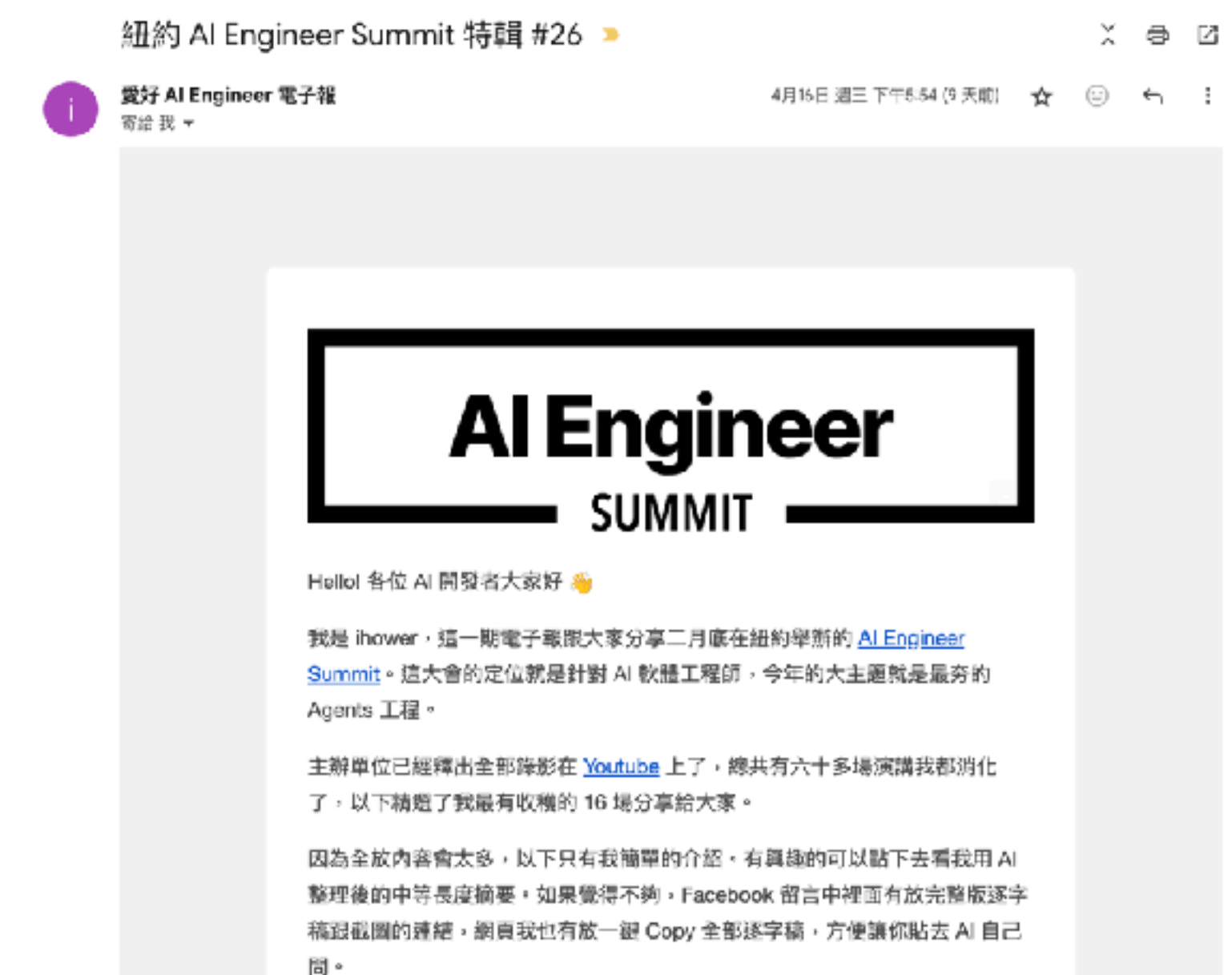
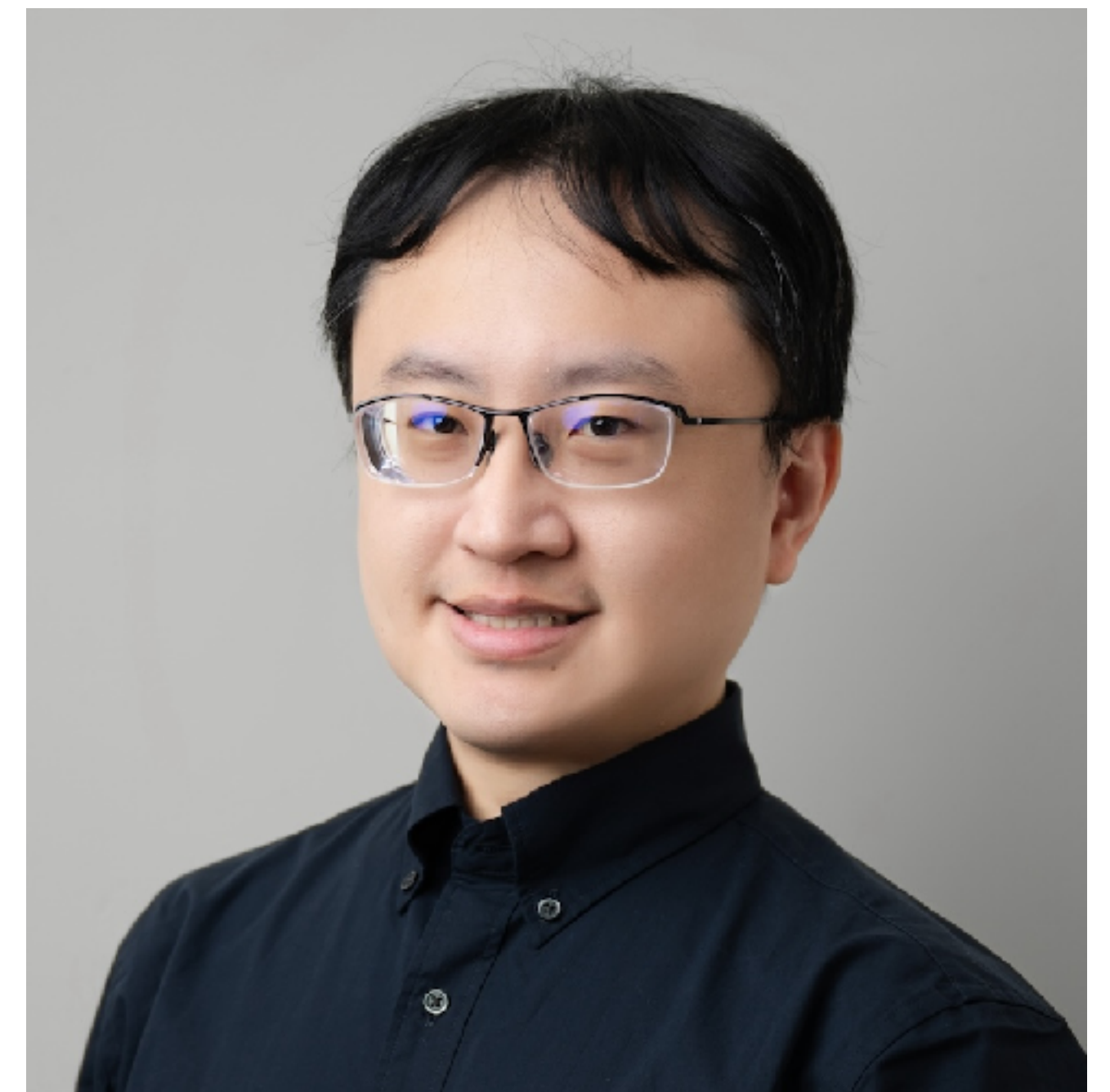
實戰 AI Agents 應用開發

從 Web 後端到前端的整合

ihower@WebConf Taiwan 2025/12/13

我是誰

- 張文鈺，網路暱稱 ihower
- 2002 年開始從事 Web 軟體開發，擅長 Ruby on Rails 技術
- 2005 年 個人部落格 <https://ihower.tw>
- 2018 年 自行開業 愛好資訊科技 <https://aihao.tw>
- 2023 年 開始經營 AI Engineer 電子報
- OpenAI Agents SDK 是我在積極貢獻的開源專案
- 財報狗的 AI Agent 開發顧問 🐶



main 28 Branches 44 Tags Go to file t + <> Code

[📖 README](#)
[📄 MIT license](#)
[✎](#)
[☰](#)

+ 178 contributors

去年 WebConf 談了 Agent 原理和核心

<https://ihower.tw/blog/12586-llm-based-ai-agents>



愛好資訊科技有限公司 創辦人

張文鈺 iHower



共筆文件 

投影片 

議程資訊

講者介紹

淺談 LLM-based AI Agents 應用開發

🕒 12/27 (Fri.) 13:30~14:15 📍 M棟

AI Agent ! Multi-Agents ! Agentic Workflow! 這些充斥業界的 buzzword 被吹捧得如夢似幻，這場演講將深入淺出探討這項技術，用樸實的技術實作說明其核心原理，揭示其真實的運作、能力範疇與侷限。希望能讓您遇到這些 buzzword 時，不再只是被表面所迷惑，而是能夠看穿其技術本質，洞悉浮華背後的真相。

#產品設計

目標會眾

適合對 LLM API 和 Prompt Engineering 有基礎知識的開發者參加

分享資訊 

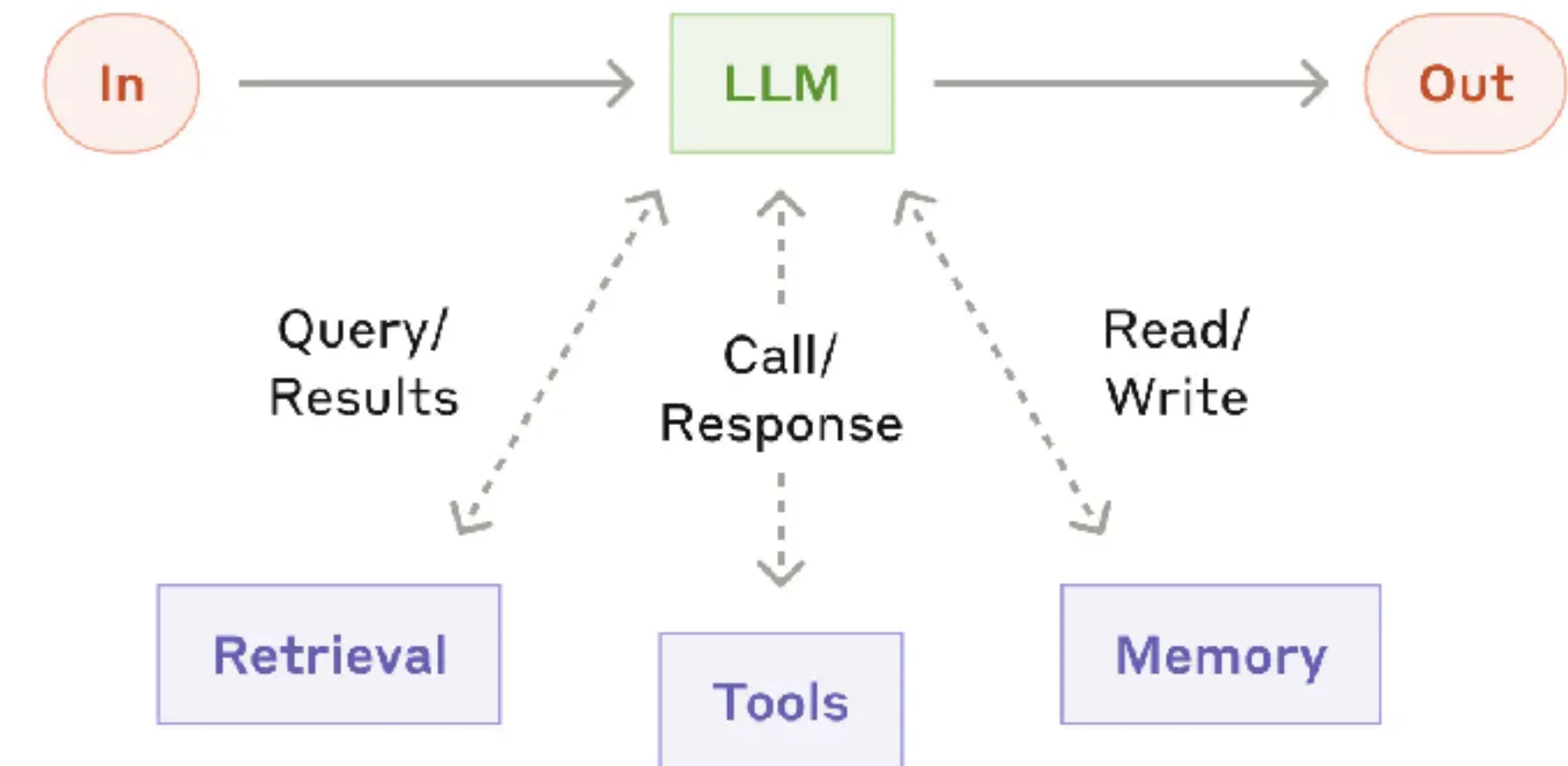
Agenda

- 架構選擇: OpenAI Agents SDK + FastAPI
- 前後端整合的發展
- TTFT 指標和改進
- Prompt Caching
- 上下文工程

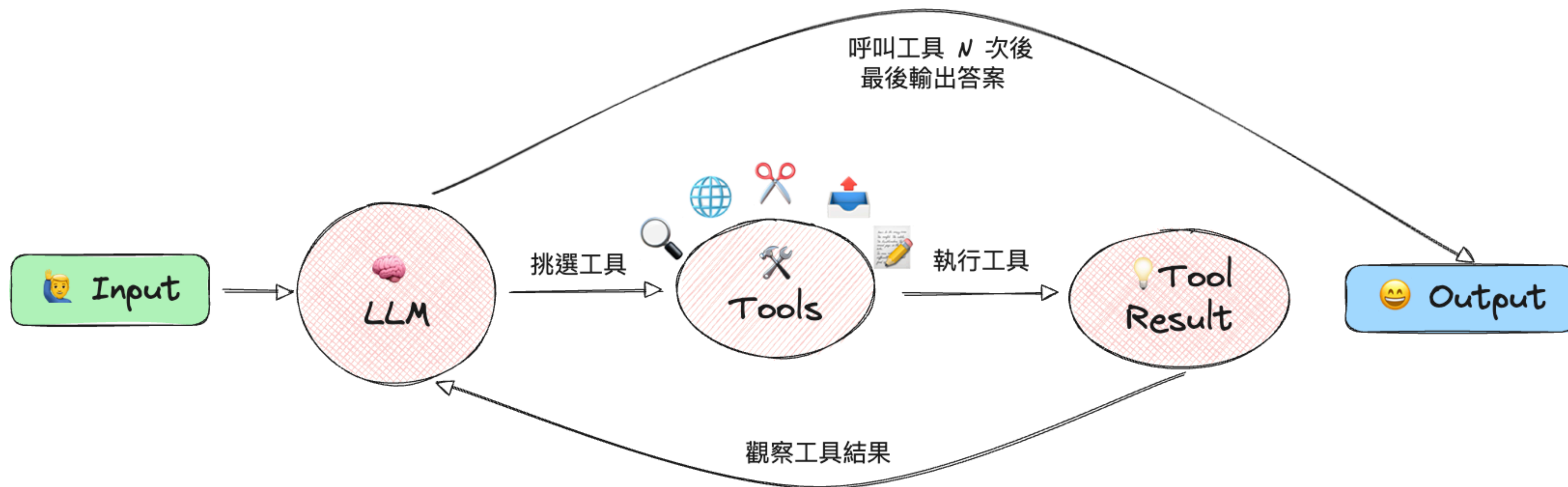
前情提要: LLM-based Agent

一個使用 LLM 大模型的 AI 元件: 它可以根據目標指令，決定步驟、不斷呼叫工具，直到完成任務

- 🧠 LLM 就像一位被關在房間裡的天才
- 🔒 無法主動學習或操作世界，也沒辦法存取即時資訊或使用電腦
- 🛠️ 需要透過工具，由外部系統幫它執行，或是存取長期記憶
- 🗣️ LLM 會輸出一段“工具執行指令”，由外部系統執行後把結果送回來，讓它不斷決定下一步，直到完成任務



Agent is LLM using tools in a loop



Agent is LLM using tools in a loop

```
instructions = "目標任務描述"
tools = [工具1, 工具2, 工具3....]
messages = [] # 記錄目前 messages 狀態

while True:
    response = model.run(instructions, tools, messages) # 呼叫 LLM 大模型，請問下一步

    if response.tool_calls:
        tool_result = tools.run(response.tool_calls) # 執行工具
        messages.append(tool_result) # 更新狀態
    else:
        print(response.final_answer) # 不需要執行工具，回傳最後答案
        break
```

可參考我之前的
淺談 LLM-based AI Agents 應用開發 分享
<https://ihower.tw/blog/archives/12586>

你應該自己練習寫一個 Agent

<https://fly.io/blog/everyone-write-an-agent/>



Articles 文章

Security 安全

Infra Log 基礎設施日誌

Customers 客戶

Docs

Community 社群

Status 狀態

Pricing 定價

Sign In 登入

Get Started 開始使用



READING TIME • 15 MIN 閱讀時間 15 分鐘





Thomas Ptacek
@tqbf

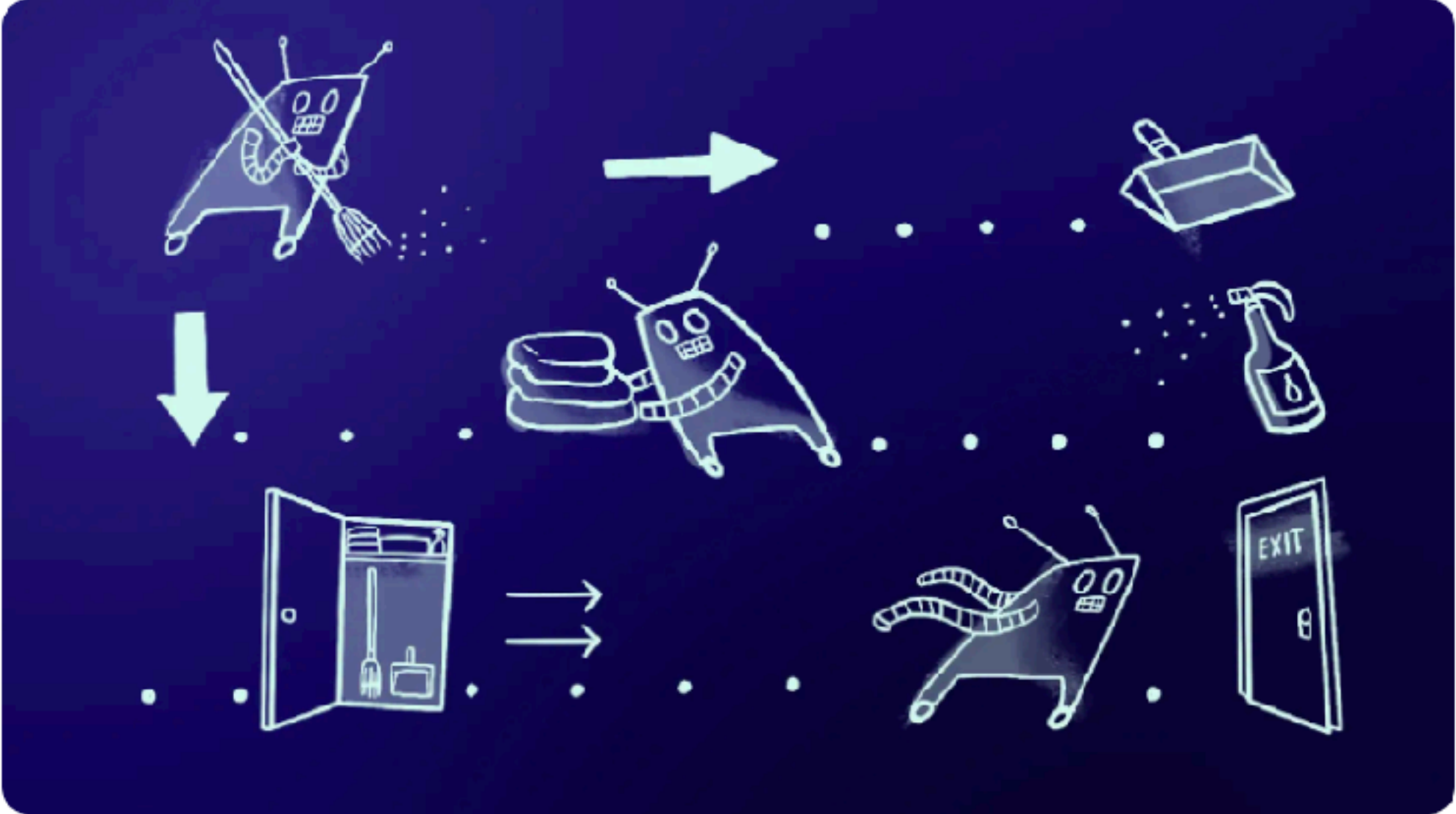
You Should Write An Agent

你應該寫一個代理人

Previous post ↓ 上一篇

↓

[Corrosion 腐蝕](#)



 Annie Rugt

架構選擇

OpenAI Agents SDK+FastAPI

Agents 框架選擇

- OpenAI Agents SDK <https://openai.github.io/openai-agents-python/>
 - 我是原始碼貢獻者 Top 5
 - 相比其他框架，功能比較簡單，但能夠研究透徹掌握、有 bug 能自己修
- PydanticAI <https://ai.pydantic.dev/>
 - 如果你熟後端、Python
- AI SDK <https://ai-sdk.dev/docs/agents/overview>
 - 如果你熟前端、JavaScript / TypeScript 派
- ADK <https://github.com/google/adk-python>
 - 如果你是 Google 粉
- 程式語言建議用 Python 或 JavaScript
 - 各種 Gen AI 相關套件 Python 和 JavaScript 兩種版本是一定有的
 - JavaScript 搭配前端時++
 - Python 做 AI 評估時++
 - 這部分就是在做資料科學、機器學習，AI 生態系完整
 - Python notebook, Google colab 做實驗真的好用



OpenAI Agent SDK 速成

- Demo code: <https://github.com/ihowe/ihowe-llm-workshop/blob/main/oa-agent-basic.py>
- 用 gpt-5.2
- 可調整 reasoning effort: none, low, medium, high, xhigh
- 對話紀錄 chat history 在 Round 2 帶入

```
from agents import Agent, ModelSettings, Runner, function_tool
```

```
@function_tool
```

```
def generate_number() -> int:
```

```
    print("generate_number called")
```

```
    return 42
```

```
agent = Agent(
```

```
    name="foobar",
```

```
    model="gpt-5.1",
```

```
    instructions="you are a helpful assistant",
```

```
    tools=[generate_number],
```

```
    model_settings=ModelSettings( reasoning={ "effort": "high", "summary": "au
```

```
)
```

```
async def main():
```

```
    print("-----Round 1-----")
```

```
    result1 = await Runner.run(agent, "First think and call the `generate_number` tool")
```

```
    print(f"AI: {result1.final_output}")
```

```
    chat_items = result1.to_input_list()
```

```
    print("----")
```

```
    for x in chat_items:
```

```
        print(x)
```

OpenAI Agent SDK 速成(streaming)

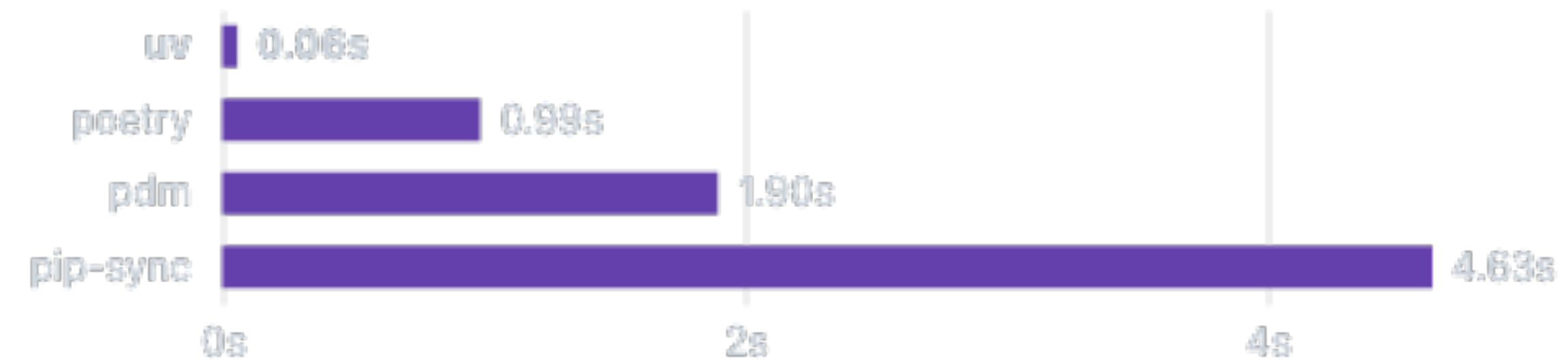
- <https://github.com/ihowe/ihowe-llm-workshop/blob/main/oa-agent-streaming.py>
- 各種串流事件: 打開推理、推理摘要比較看看

```
async for event in result.stream_events():
    if event.type == "raw_response_event": # 這個是 Responses API 的原始 event
        # print(event)

        item = getattr(event.data, "item", None)
        if item:
            print(f"{event.data.type} ItemType: {item.type}")
        elif event.data.type in ["response.output_text.delta", "response.reasoning_summary_text.delta"]:
            print(f"{event.data.type}: {event.data.delta}")
        else:
            print(f"{event.data.type}" )
```

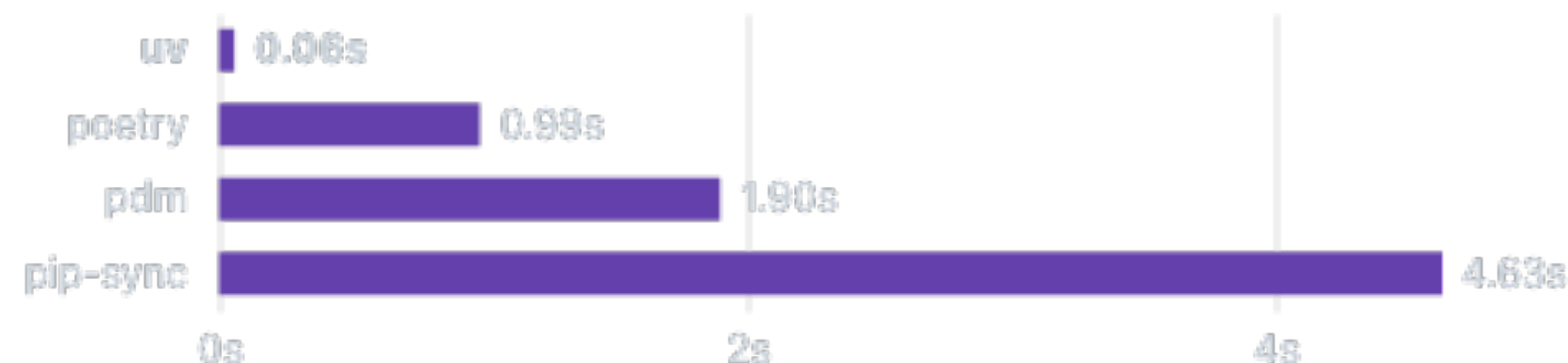
Python 的 API Server 選擇

- FastAPI: 現代高性能 Python Web Framework
 - <https://fastapi.tiangolo.com/>
 - 絕大部分的操作都是 I/O bound 而不是 CPU bound，非常適合 asynchronous programming 架構的 fastapi，和 Node.js 性能同等級
- 我們就做 REST API，做前後端分離架構。這裡的前端可以是：
 - 我們用 Vibe Coding 來刻一個 HTML/JS 前端
 - 另一個 application server 後端，例如 Ruby on Rails
 - 專門的前端，例如 React/Vue.js



Python 生態系: uv 跟 asyncio

- **uv**: 現代高性能 Python package 工具
 - <https://github.com/astral-sh/uv>
- **asyncio**: Python 的 Asynchronous Programming
 - <https://docs.python.org/zh-tw/3.13/library/asyncio.html>
 - 簡介 <https://carrot-cheng.medium.com/python-%E7%9A%84-concurrency-programming-3aeca87073e6>
 - 必學的 Python 程式語言功能
 - FastAPI 基於 asyncio
 - OpenAI Agents SDK 基於 asyncio
 - OpenAI Python SDK 支援 asyncio
 - 很多 Python library 也會同時做兩種版本: non-async 跟 async 版本
 - 其實 Google colab 內也是 asyncio 環境



將 OpenAI Agents SDK 放入 FastAPI

一個簡單的 AI Agent 對話

- 後端 GET /api/v1/agent_simple
 - <https://github.com/ihower/ihower-llm-workshop/blob/main/main.py>
- 前端 HTML/JS
 - https://github.com/ihower/ihower-llm-workshop/blob/main/static/simple_agent.html

前端 Vibe Coding

請製作與AI對話的 HTML 頁面在 static/simple_agent.html，左欄是一個 textarea 的 form，按下 Submit 後

1. 將用戶 query 先插到右欄: Human: query，然後 清空左邊的 form 資料
2. 用 javascript 送出 GET 到 /api/v1/agent_simple 帶有參數 query 和 previous_response_id (預設是 null)
3. 回傳是一個 json，請將 output 累加到右欄，並將 previous_response_id 保存下一次 Sumit帶入

左欄再多一個按鈕是 Reset，按下後清空左欄 textarea 和右欄所有內容，並將 previous_response_id 改回 none

畫面用 bootstrap 裝飾一下(用 CDN 載入 css)

後端程式碼請參考 main.py

改用 Streaming 串流

- 後端 GET /api/v1/agent_simple_stream
 - <https://github.com/ihowe/ihowe-llm-workshop/blob/main/main.py>
 - 關鍵是自行定義後端要轉發給前端的 streaming chunk
 - 哪些 chunk 是 delta 讓前端逐步顯示
 - 哪些 chunk 是 metadata
 - 哪些 chunk 可以前端整個新增/替換顯示
 - 結構化輸出怎麼辦 (請參考 `jiter.from_json` 方法 解析 partial json)
- 前端 HTML/JS
 - https://github.com/ihowe/ihowe-llm-workshop/blob/main/static/simple_agent_stream.html

前端 Vibe Coding

請製作與AI對話的 HTML 頁面在 `static/simple_agent_stream.html`，左欄是一個 `textarea` 的 form，按下 Submit 後

1. 將用戶 query 先插到右欄: Human: query，然後 清空左邊的 form 資料
2. 用 javascript 送出 GET 到 `/api/v1/agent_simple_stream` 帶有參數 `query` 和 `previous_response_id` (預設是 `null`)

3. 回傳是用 SSE 回來，請將串流的資料，顯示在畫面右欄

注意串流資料是可以交加的，包括 `THINK_START`, `THINK_TEXT`, `CALL_TOOL` 以及 `content delta` 資料

例如可能先 `THINK_START` -> `THINK_TEXT` -> `content delta` -> `CALL_TOOL` -> `THINK_START` ->

`THINK_TEXT` -> `CALL_TOOL` -> `content delta` 這樣交錯

這些都陸續累加顯示到右欄，直到收到 `[DONE]` 時，才結束 SSE 串流

並將 `DONE` 訊息中的 `previous_response_id` 保存下一次Submit帶入

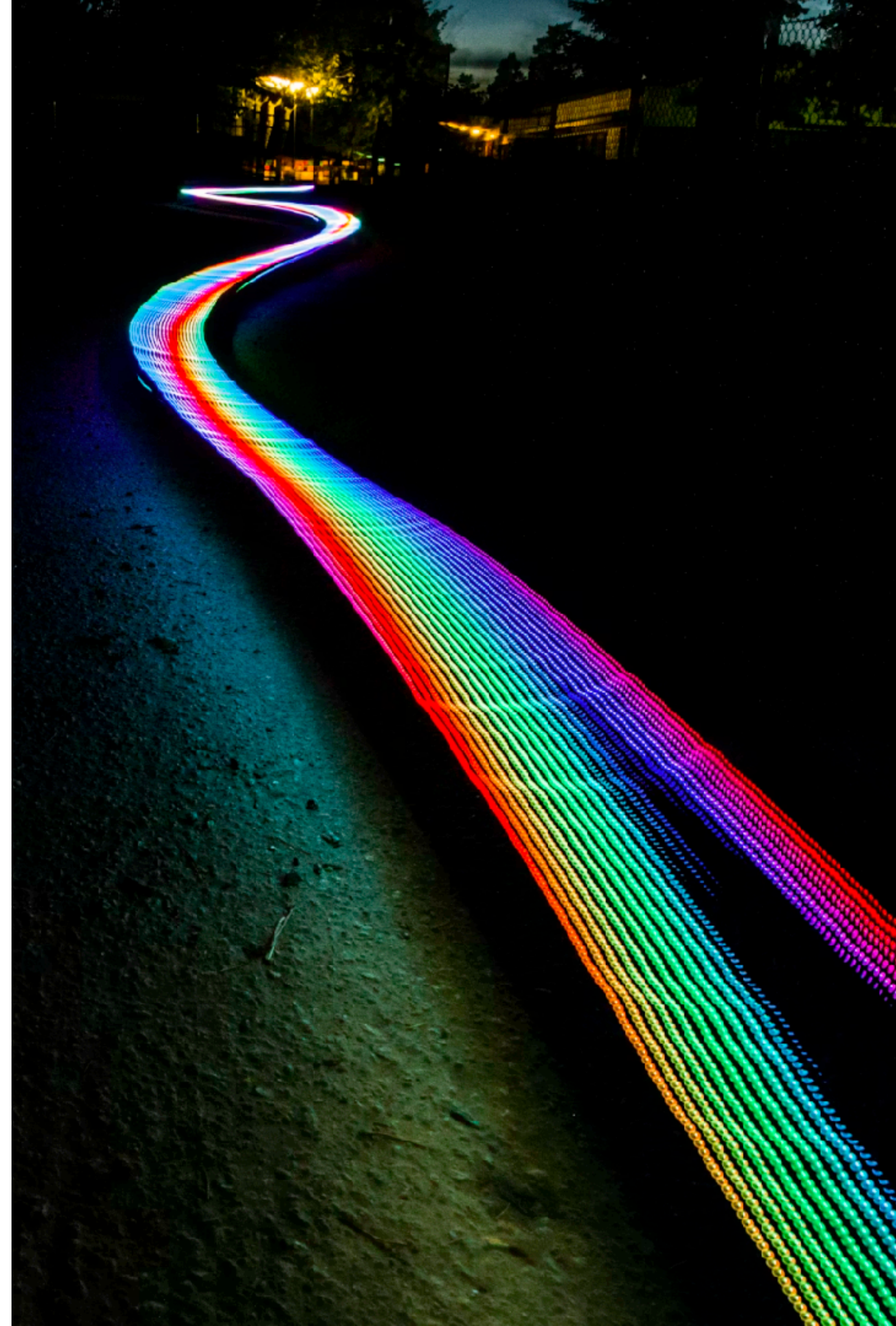
左欄再多一個按鈕是 `Reset`，按下後清空左欄 `textarea` 和右欄所有內容，並將 `previous_response_id` 改回 `none`

畫面用 `bootstrap` 裝飾一下(用 `CDN` 載入 `css`)

後端程式碼請參考 `main.py`

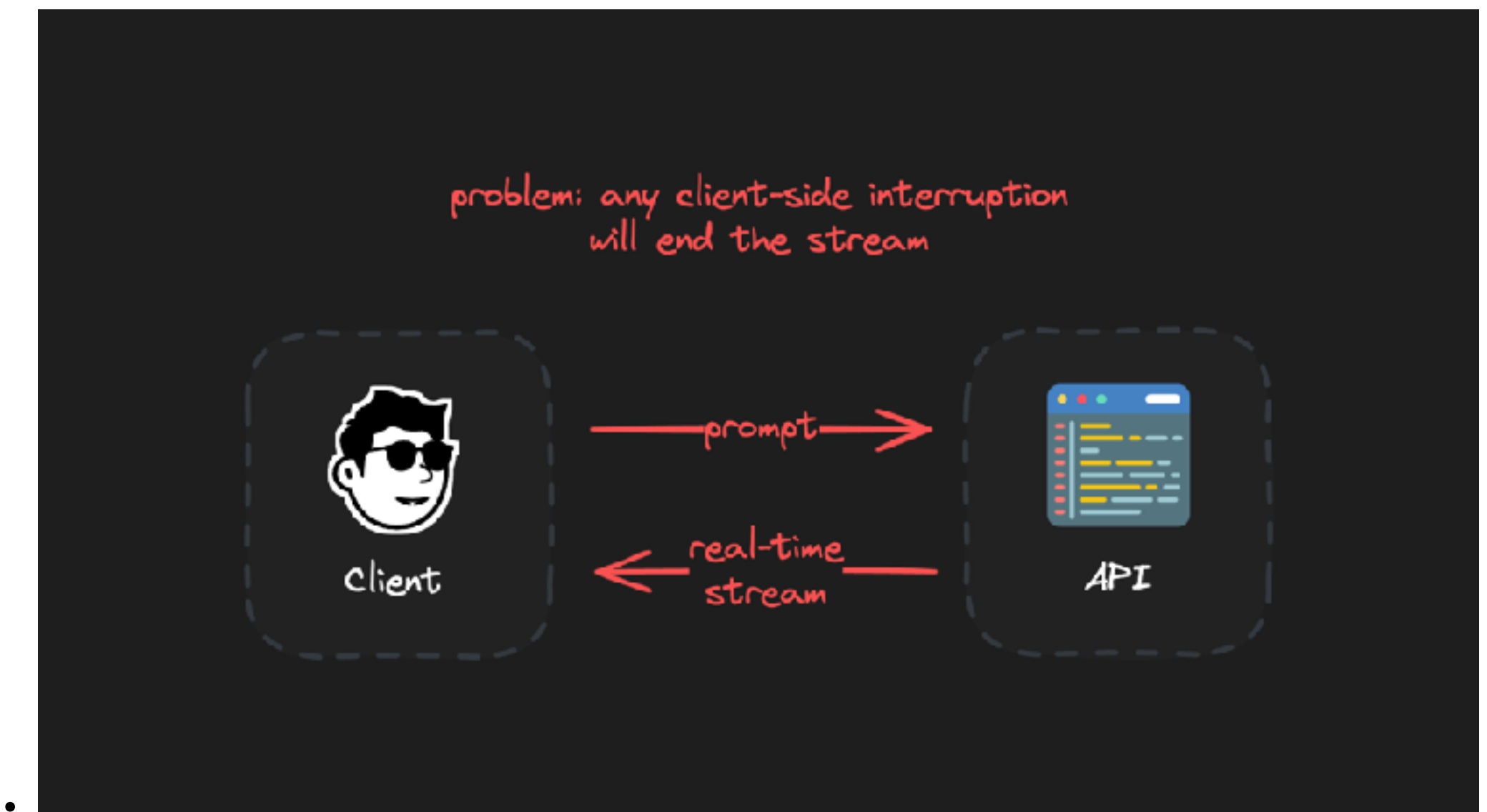
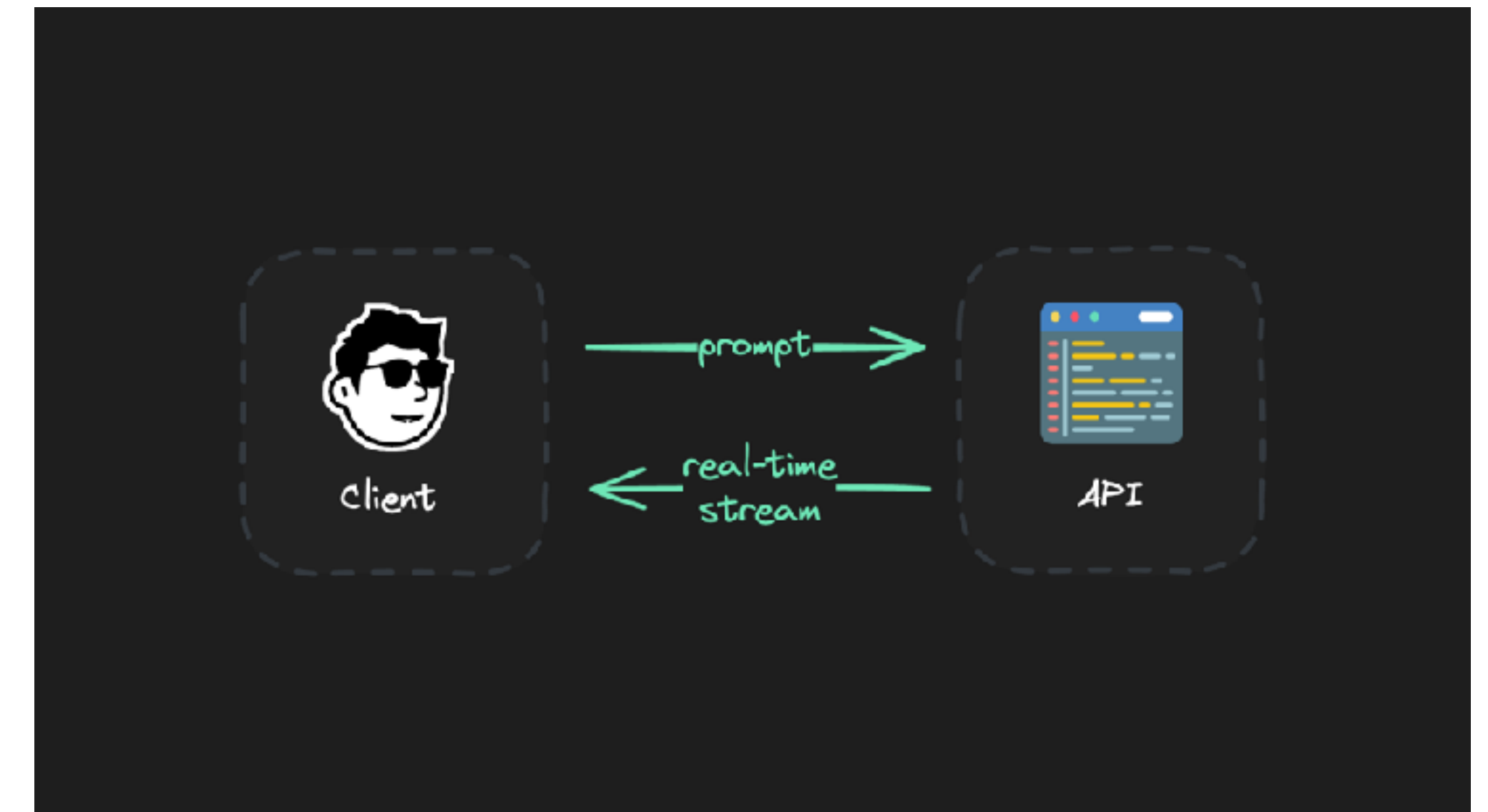
關於 Streaming 串流

- 如果你在做一個用戶對話應用，Streaming 輸出是必要的，對 UX 影響很大
- 前後端的實作處理跟非串流差很多
 - Bug 也不總是相同
- 事後改會很痛苦，大改又要重測試
- 請 Day 1 就做



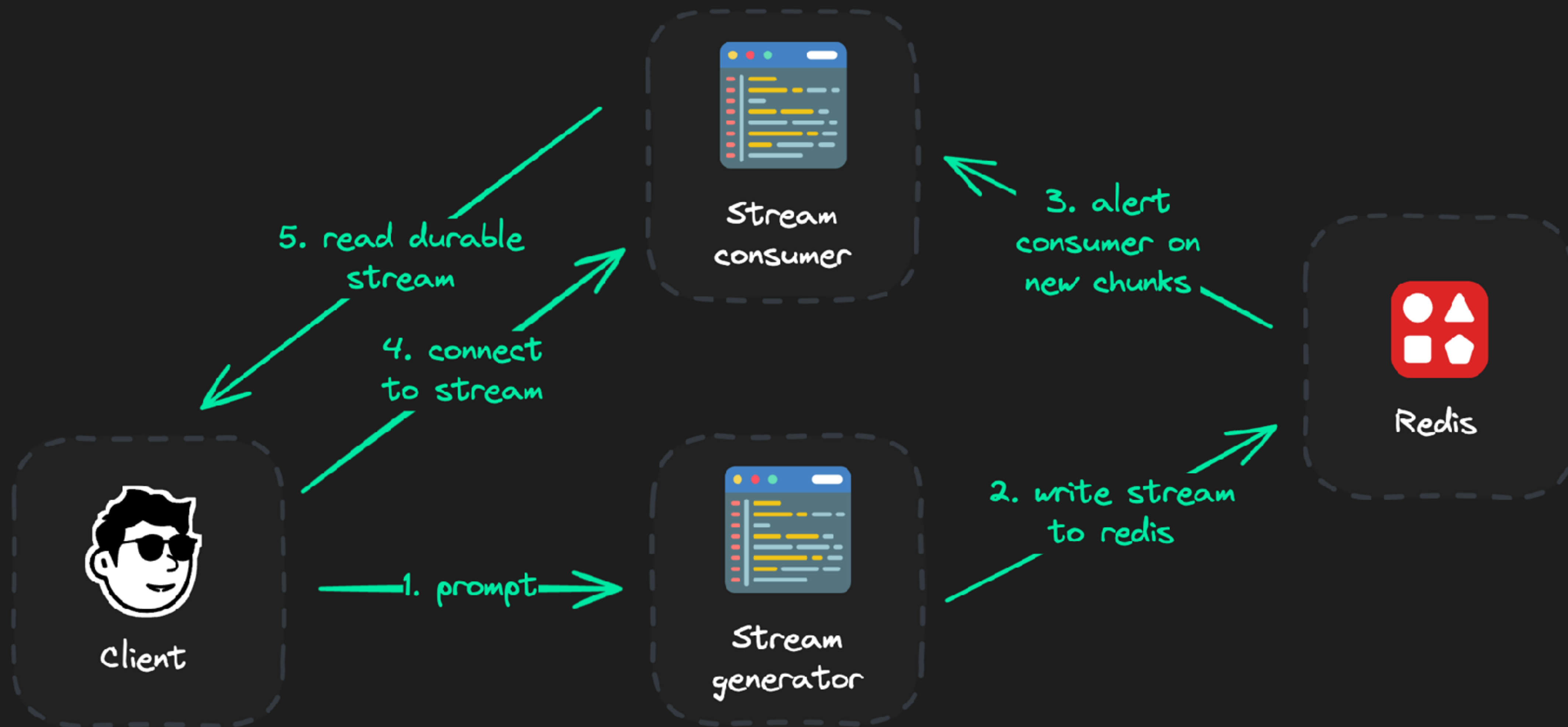
Streamable Resume 架構

- 支援可「斷線續傳」的 Streaming 架構
 - 和 LLM provider 的 SSE 串流一旦斷線、重整頁面，就得全部重來
 - 對長回應與高成本模型（如推理模型）UX 極差
- 做法
 - Generation 與 Client 連線完全解耦
 - LLM 生成不中斷



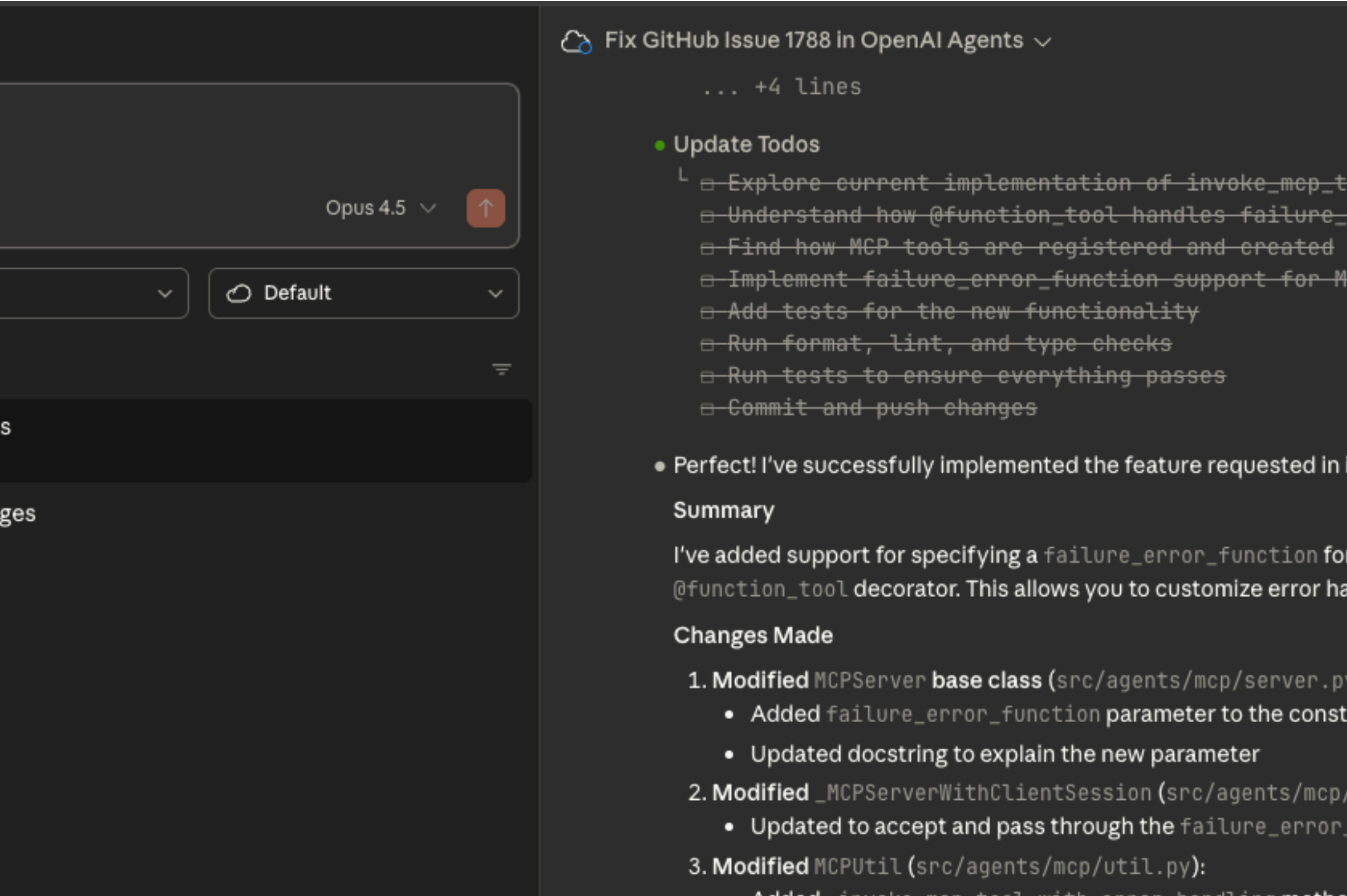
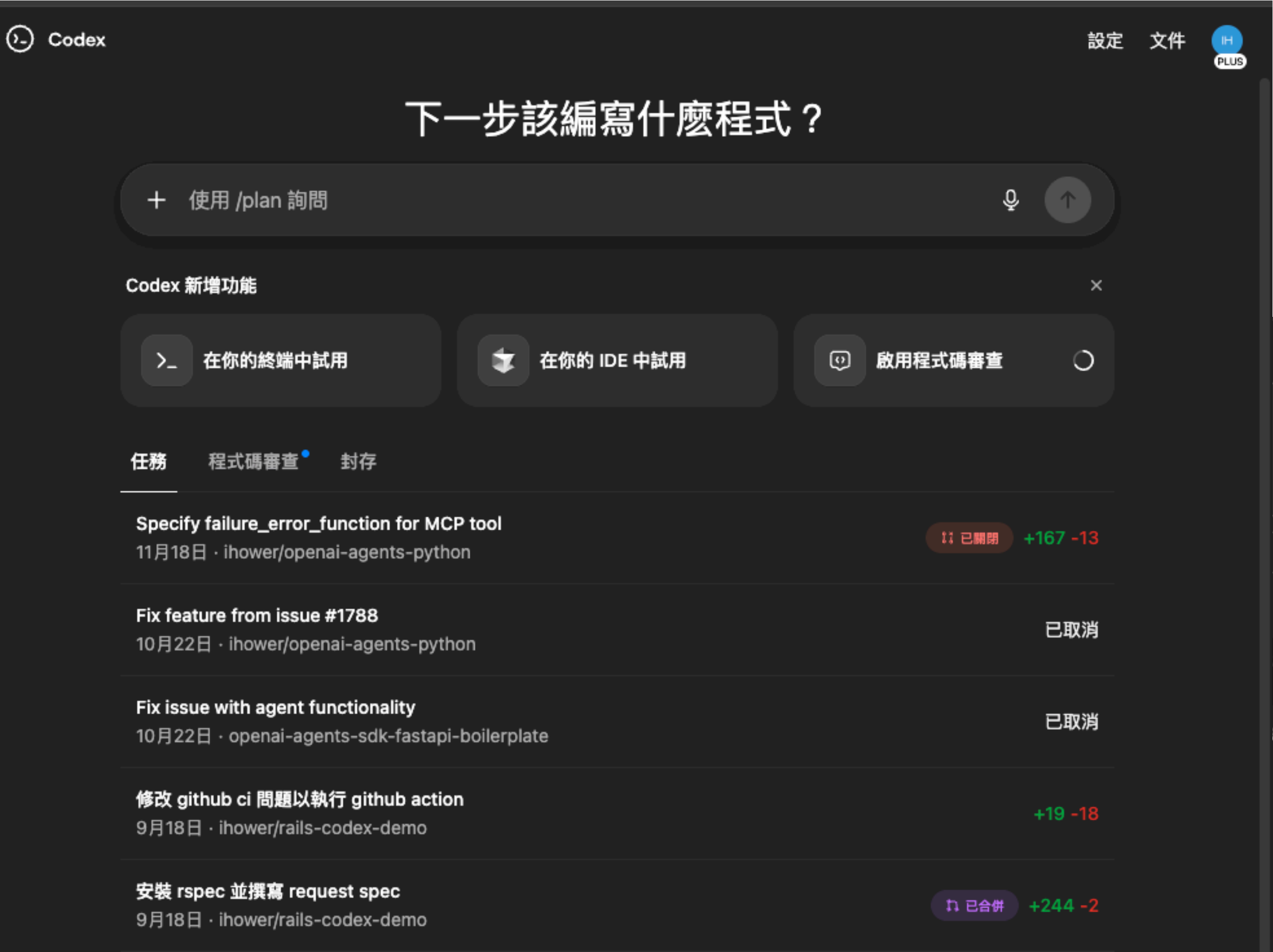
Streamable Resume 架構 做法

- 參考文章: <https://upstash.com/blog/resumable-llm-streams>
- API 要拆成兩個
 - 一個 POST 觸發: 用 background job 寫入 Redis 作為緩衝
 - 一個 GET 訂閱: 接收 stream 訊息
- 好處
 - 允許 client 端可以自由中斷換頁
 - 各種 long-running agent 的基本架構



Coding Agent 的 Web 版也都是這種架構

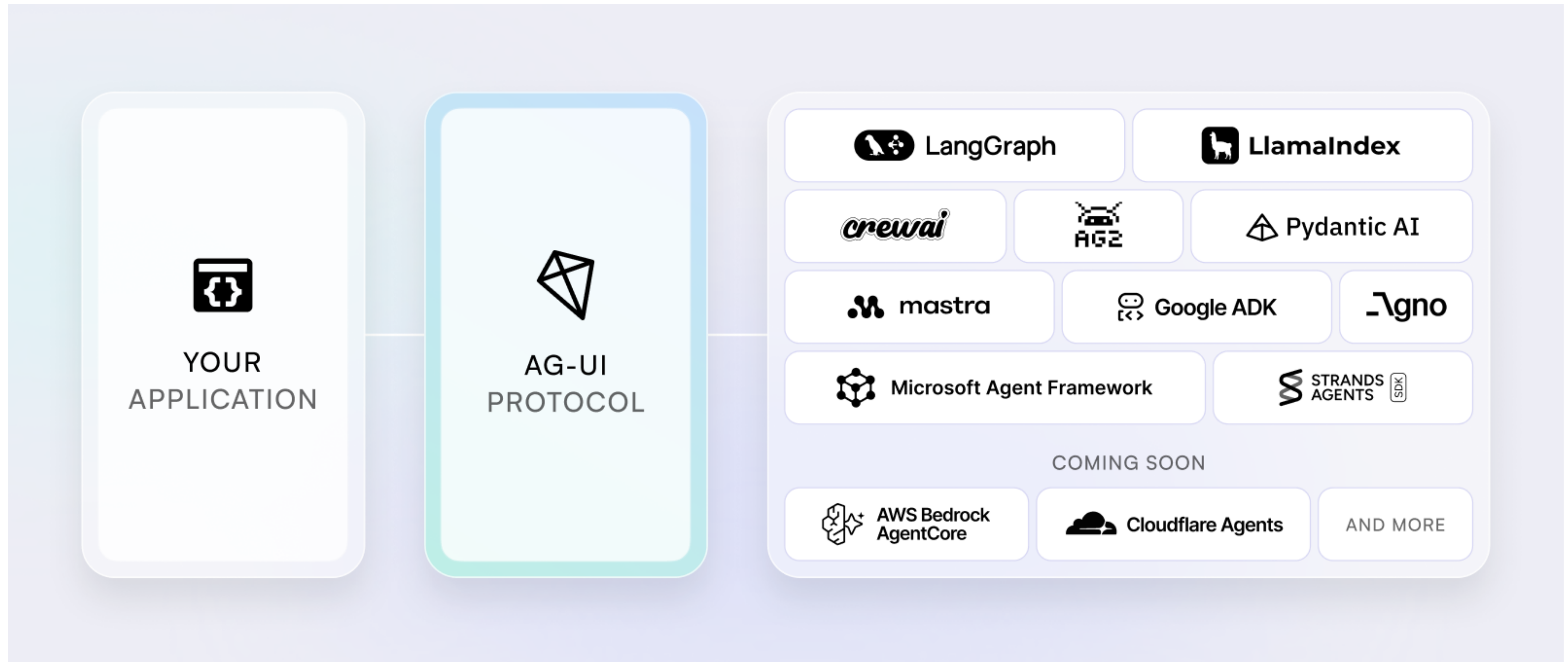
Long-running agent 必備



前後端整合的發展

AG-UI

ag-ui.com 標準化的 UI Streaming chunk 格式

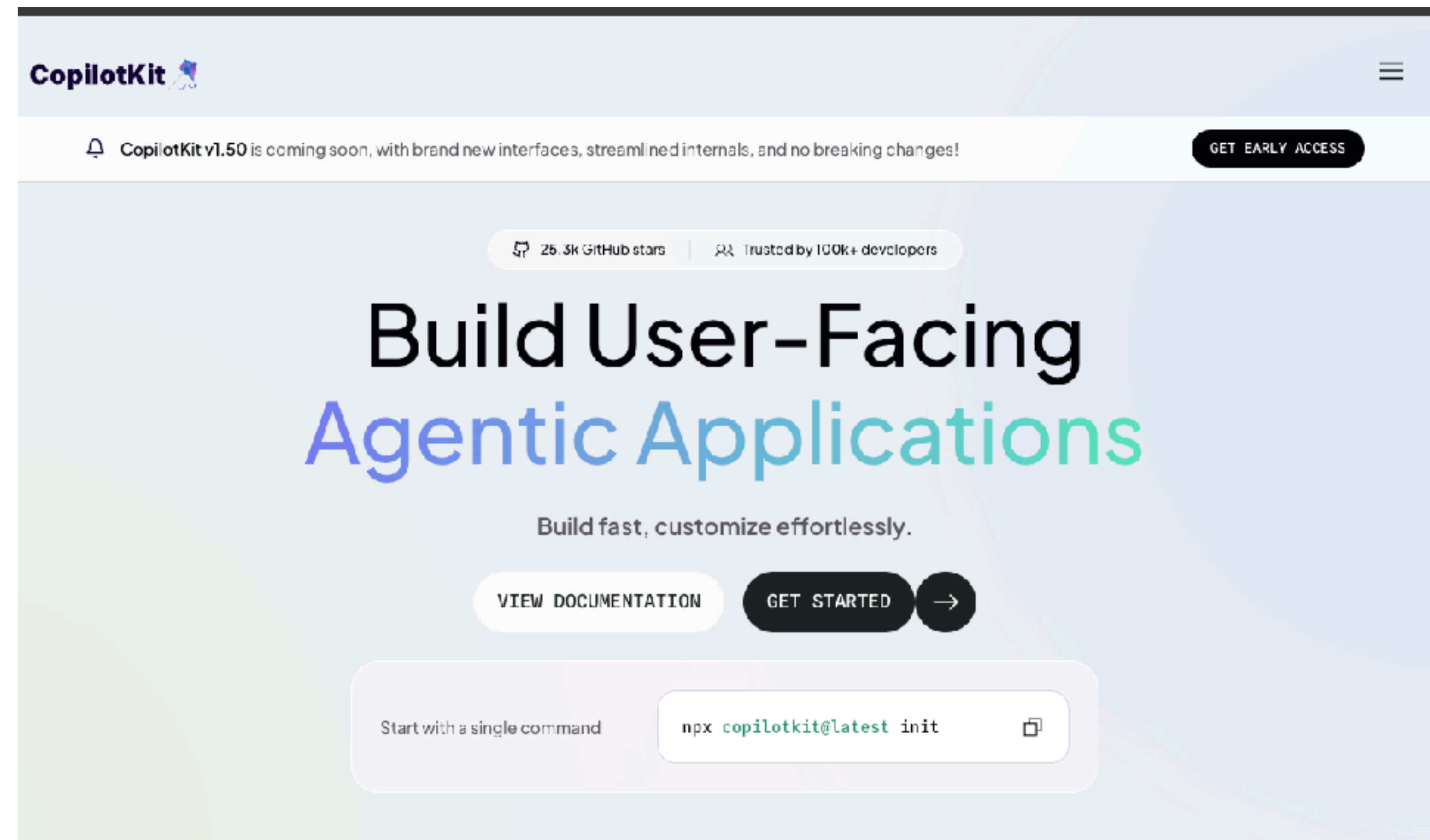


後端改造: 用它定義好的 chunk 格式

- 有 Python 和 TypeScript SDK
 - <https://docs.ag-ui.com/sdk/python/core/overview>
 - <https://docs.ag-ui.com/sdk/js/core/overview>
- 我的 demo
 - <https://github.com/ihower/ihower-llm-workshop/blob/main/main.py>
 - 看 POST /api/ag-ui 這個端點

前端部分: CopilotKit 是一套符合這個標準的前端元件庫

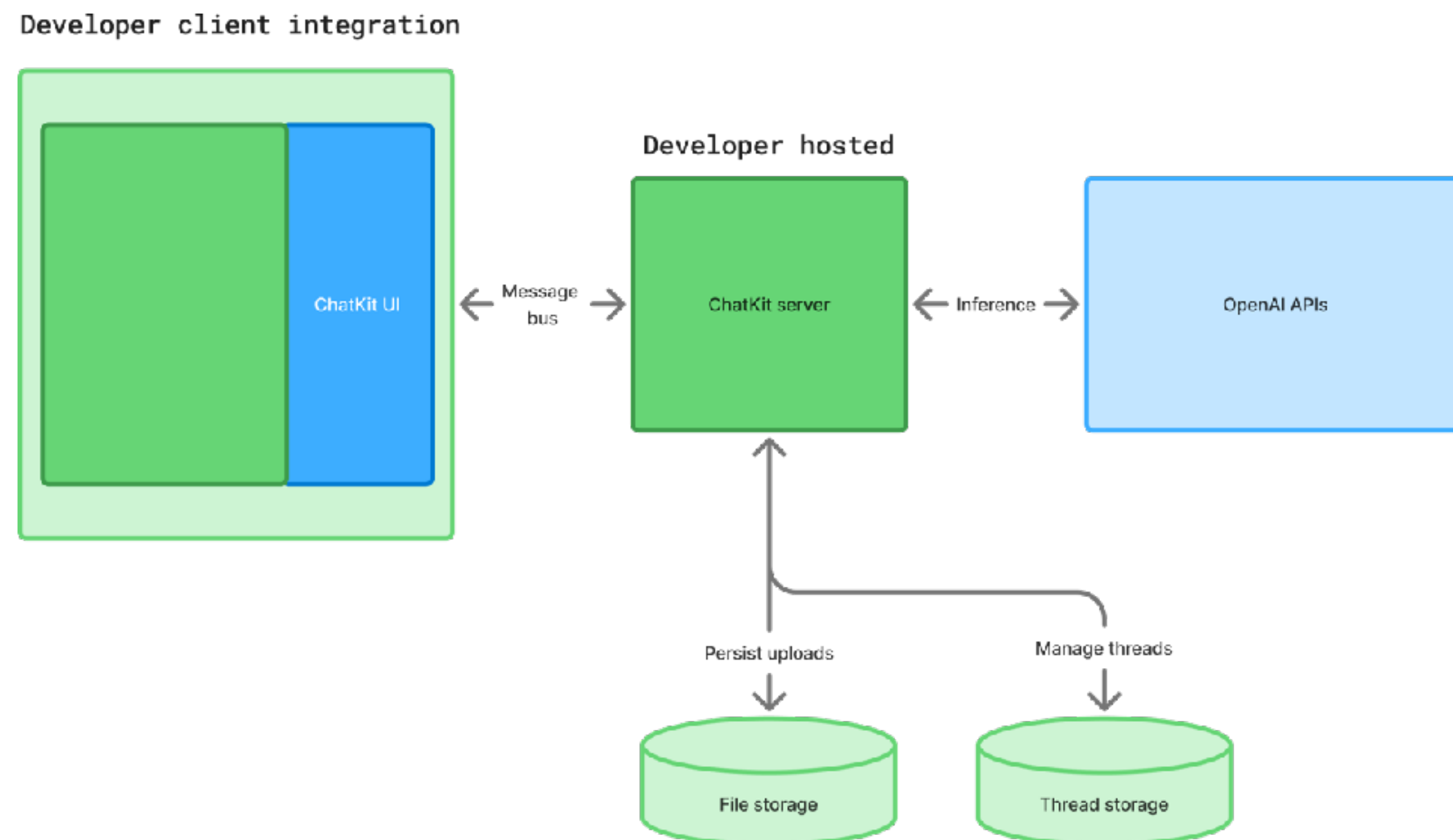
- <https://www.copilotkit.ai/>
- React
- Interactive Demo <https://cloud.copilotkit.ai/getting-started>
- 前端 Demo
- <https://github.com/ihowe/ihowe-llm-workshop/tree/main/copilotkit-frontend>



OpenAI ChatKit

也是類似的思路，標準化 UI streaming chunk

- <https://platform.openai.com/docs/guides/chatkit>
- 後端 Python SDK: <https://github.com/openai/chatkit-python>
- 前端 JS <https://github.com/openai/chatkit-js>



🎉 MCP-UI is now standardized into MCP Apps! [Learn more →](#)

MCP-UI -> MCP Apps

Interactive UI Components over MCP

Open protocol to build rich, dynamic interfaces for your agentic apps with SDKs that bring UI to AI interactions.

[Get Started](#)[GitHub](#)[Live Demo](#)[About](#)

Model Context Protocol

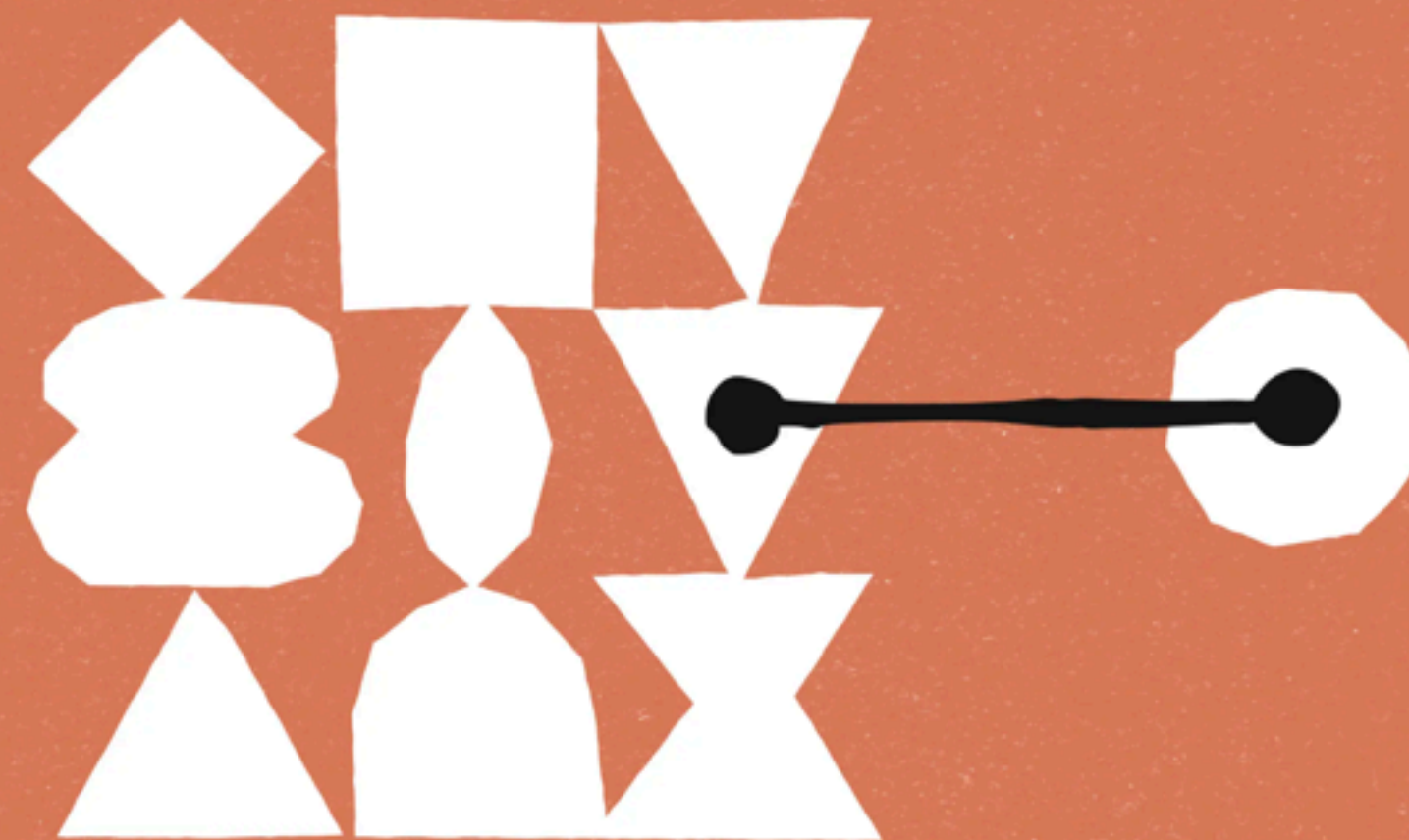
模型如何取得 Context 的標準化協議
包括 Tools、Resources 和 Prompts

<https://modelcontextprotocol.io>

Announcements

Introducing the Model Context Protocol

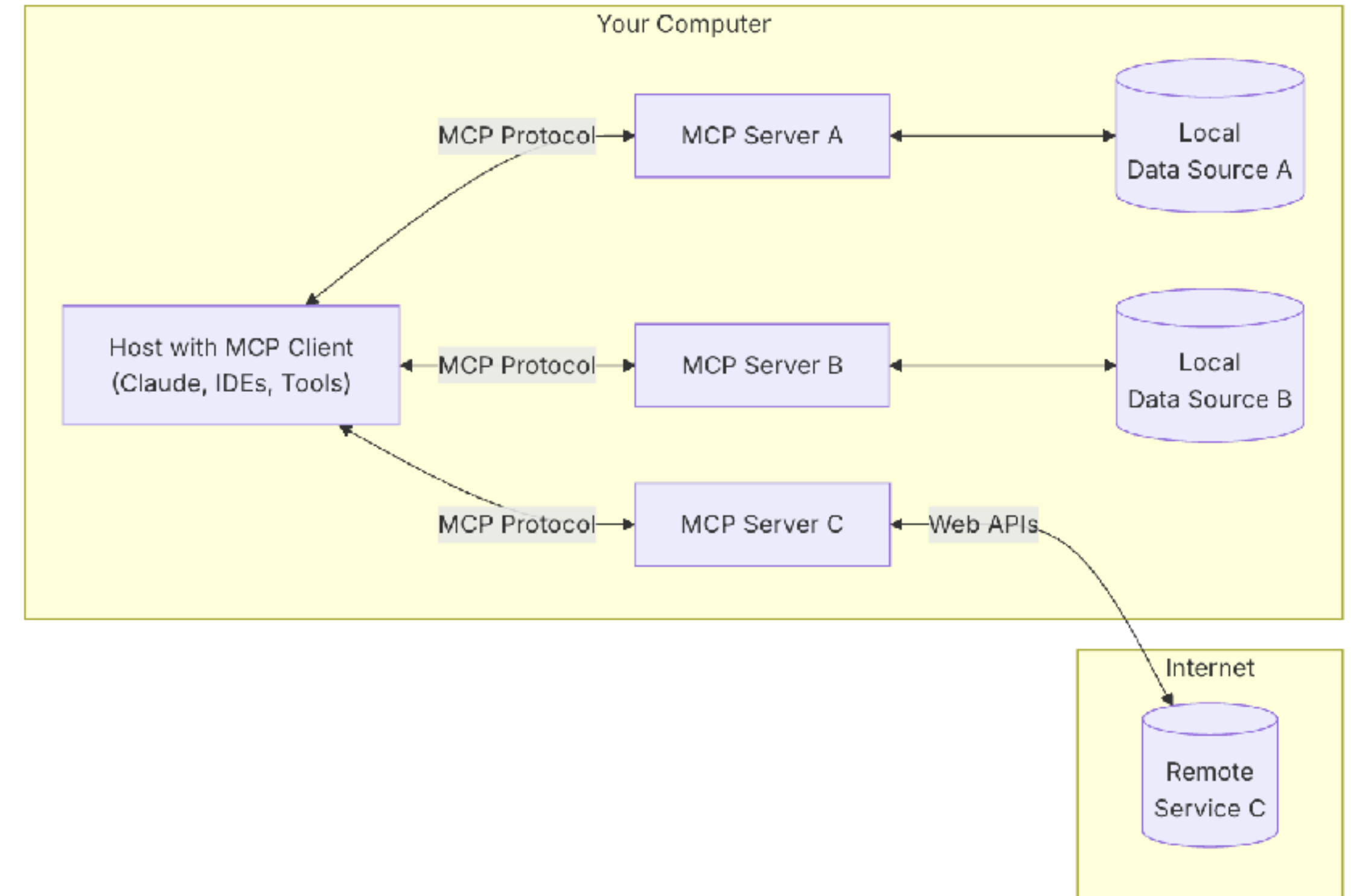
2024年11月25日 • 3 min read



常見使用場景: 讓 App 能夠更方便安裝第三方的工具擴充

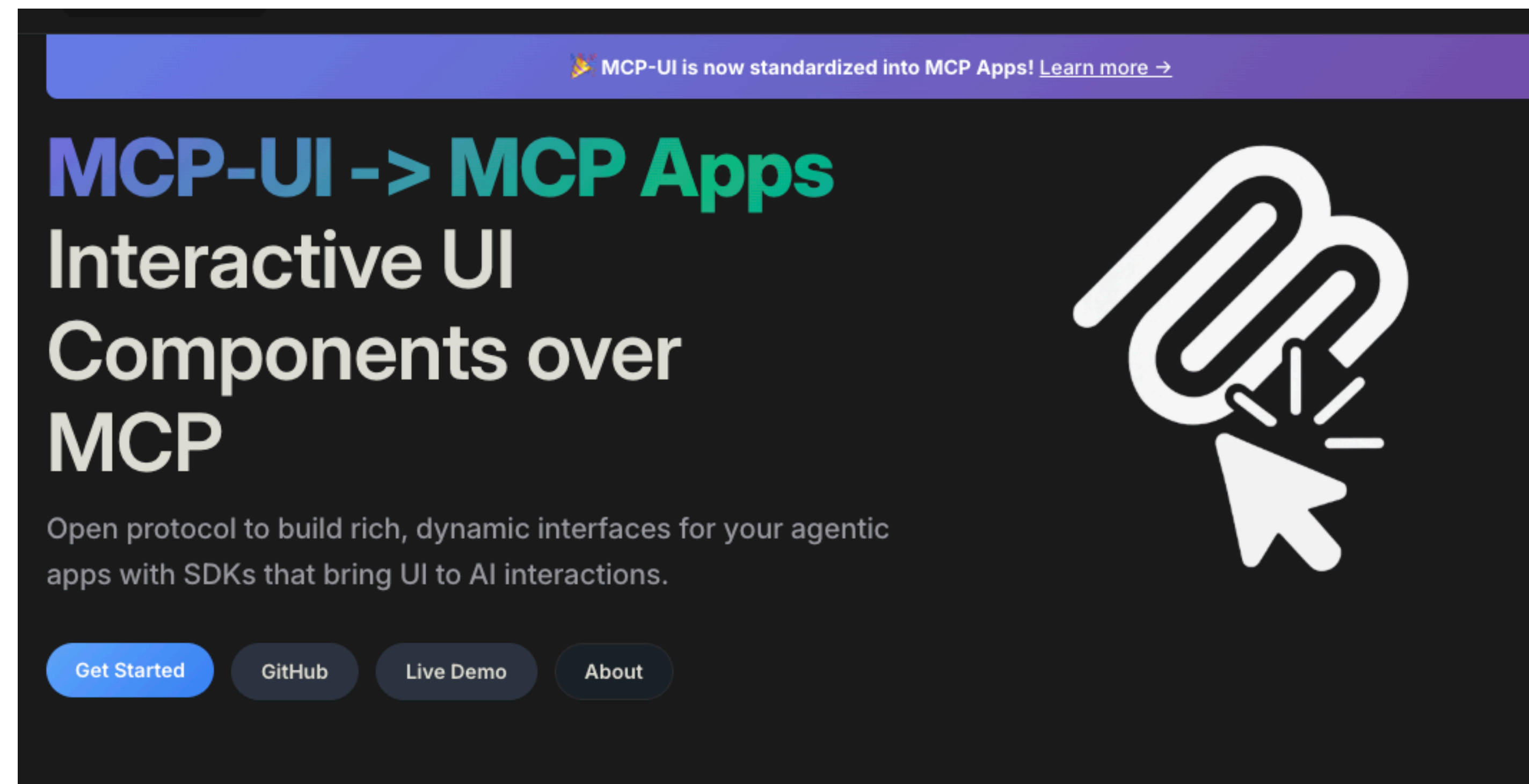
-  Client app 包括:
 - 本機 Claude Desktop App、ChatWise、Cursor、Windsurf Editor 等等 GUI 應用和 編輯器
 - 雲端 Claude Web、Dify 等等服務
 - 你自行開發的 Agent 程式
 - More: <https://modelcontextprotocol.io/clients>
-  搭配安裝不同 MCP servers 擴充
 - 一組 MCP server 就包含很多 functions 工具，是以場景和使用案例為中心，而非只安裝一個 function 工具
 - 例如安裝 Slack MCP server，就會一次安裝八個 functions 在你的 client app 上:

slack_list_channels, slack_post_message, slack_reply_to_thread,
slack_add_reaction, slack_get_channel_history, slack_get_thread_replies,
slack_get_users, slack_get_user_profile



MCP-UI 讓工具輸出標準化的 UI 組件

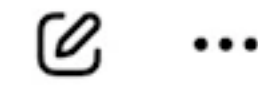
- MCP-UI <https://mcpui.dev/>
- 正在制定 MCP 標準規格
 - <https://blog.modelcontextprotocol.io/posts/2025-11-21-mcp-apps/>
 - Anthropic, OpenAI, Shopify 都有加入
- 使用案例
 - Shopify <https://shopify.engineering/mcp-ui-breaking-the-text-wall>
 - OpenAI Apps SDK <https://developers.openai.com/apps-sdk/>



11:30



≡ ChatGPT 5 >



Booking.com find me a hotel
in Paris for two adults between
11/21-11/24 with parking

B. Booking.com

Nov 21 - Nov 24 · 2 travelers



★★★★★

Zoku Paris

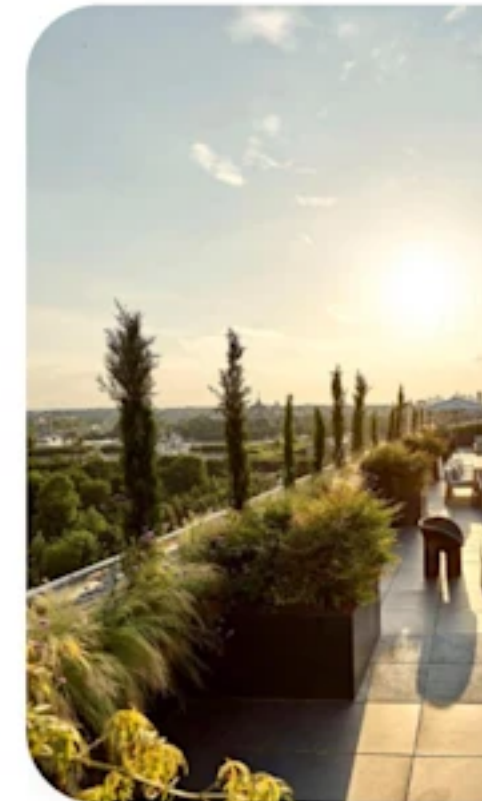
Fabulous · 2,493 reviews

Wifi Parking Pool Pets Res

From

US\$776 / 3 nights

[View on Booking.com](#)



★★★★★

Le Meurice – Dorch

Fabulous · 212 reviews

Wifi Parking Pool

From

US\$687 / 3 nights

[View on Booking.com](#)



Chats



Search



Manage subscription



Updates & FAQ



Settings



Show me my cart please!

You have 3 items from 2 shops in your cart. Checkout with each shop separately.



KOTN
2 items



Checkout

CAD \$188.00



Fire Belly Tea
1 item



Checkout

CAD \$127.35

Let me know if you'd like to change your items or see some product reviews!

← Unisex French Terry Hoodie



KOTN

Unisex French Terry Hoodie

\$188.00

MCP-UI vs. AG-UI

- AG-UI 定義 後端 app 傳給前端的 streaming chunk 格式
- MCP-UI 定義 function calling 工具回傳的格式，要用在前端 UI 組件上
- 不衝突，可以都用 XD
- MCP-UI 比較常見的場景，是你做為 client 去配合平台方
 - 這個平台，例如 ChatGPT 和 Shopify
 - 你是 MCP server 作者，你做了一個 App 工具，想在 ChatGPT/Shopify 上架在他們的 Agent 裡面
 - 因此你的回傳值，就需要符合這個標準，才能順利在他們的 app 內顯示 UI
 - 如果你沒有要做 擴充app 的話，我覺得 MCP-UI 價值就不高了，除非你也想做一個平台讓別人可以擴充 app

WHAT
DO YOU
MEAN
?

TTFT (Time-to-First Token)

TTFT

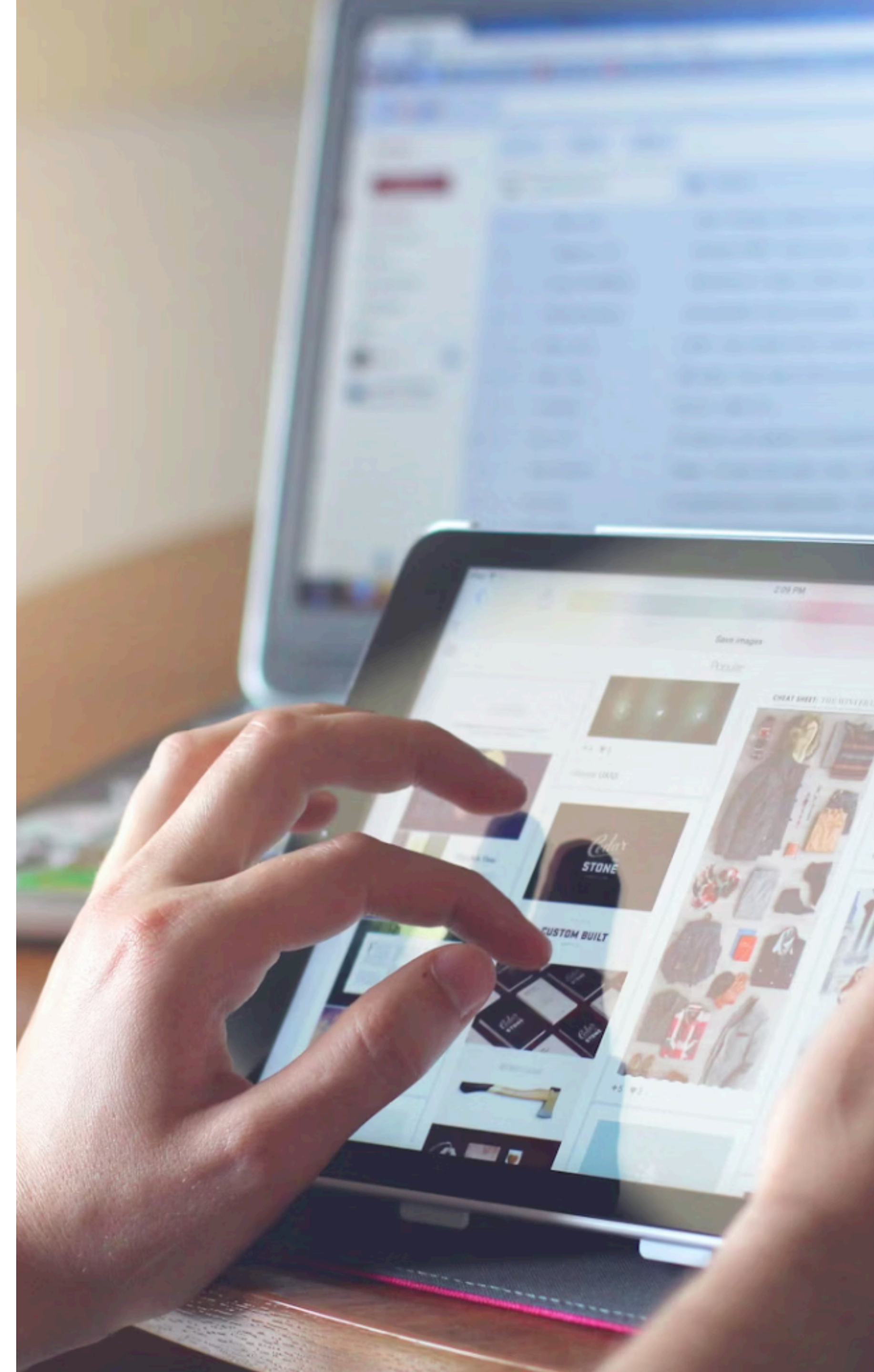
Time to First Token，用戶首個看到 Token 的時間

- 從使用者發送請求，到第一個內容開始呈現的延遲
- 非串流模式時
 - TTFT 會很久，你得等到 LLM 全部輸出完才一次看到
- 串流模型時
 - 第一個內容 delta 文字
 - 或是你可以想辦法提前告知用戶進度



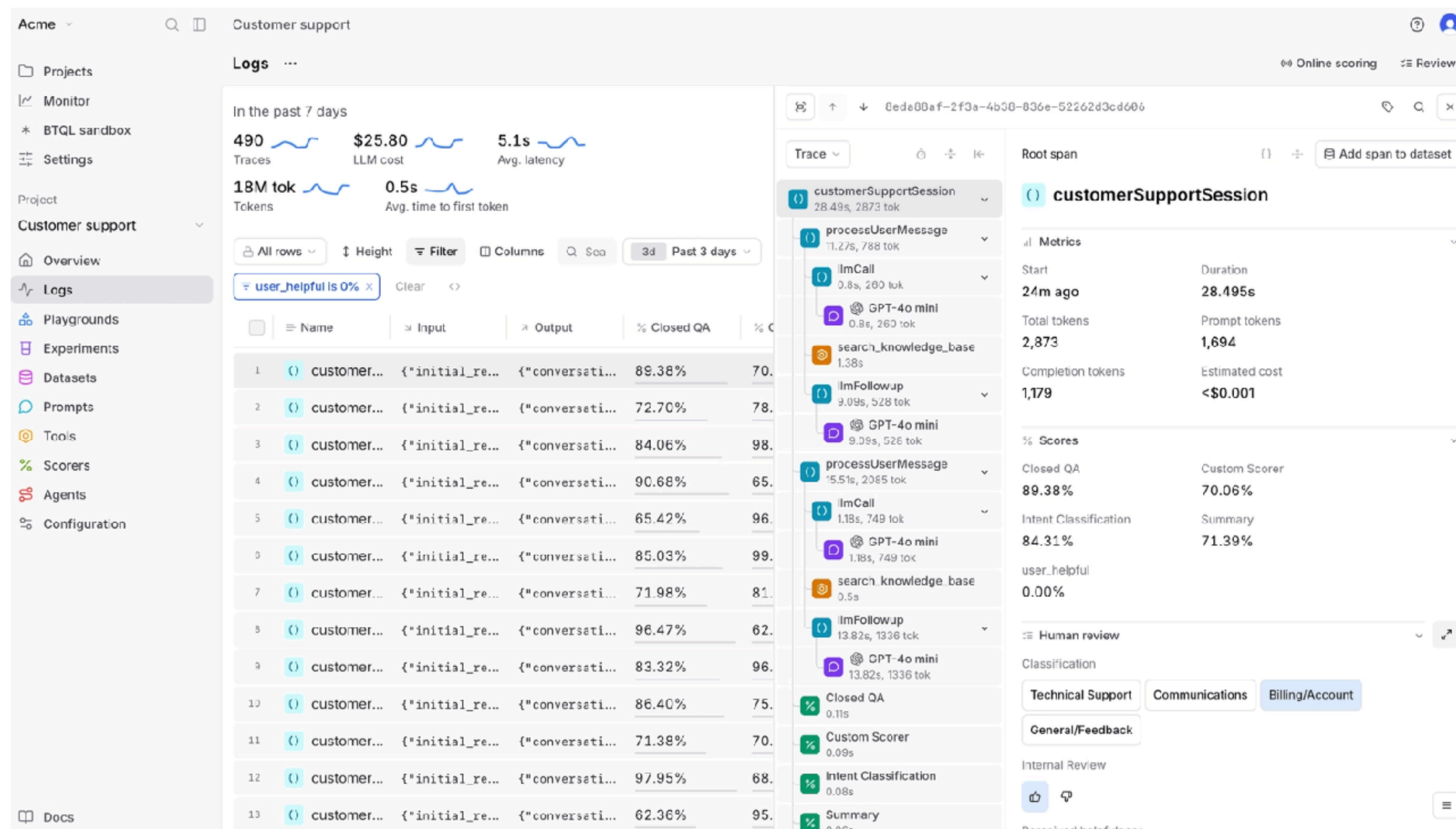
Why TTFT 重要?

- 影響使用者體驗
 - 等太久會讓用戶覺得東西壞掉了
 - 太快.... 可能會讓用戶覺得結果不夠厲害
- 推理模型的 TTFT 很糟糕
 - 專有模型(OpenAI, Claude, Gemini) 的推理內容是加密的，你看不到只能等
- 不是每個人都是 AI 專家，知道什麼推理模型，需要思考時間
 - 困難問題，工具呼叫越來越多次，latency 越來越慢



先加入觀測性

- LLM 監控工具
 - braintrust
 - Langsmith
 - Langfuse (開源)
 - Opik (開源)



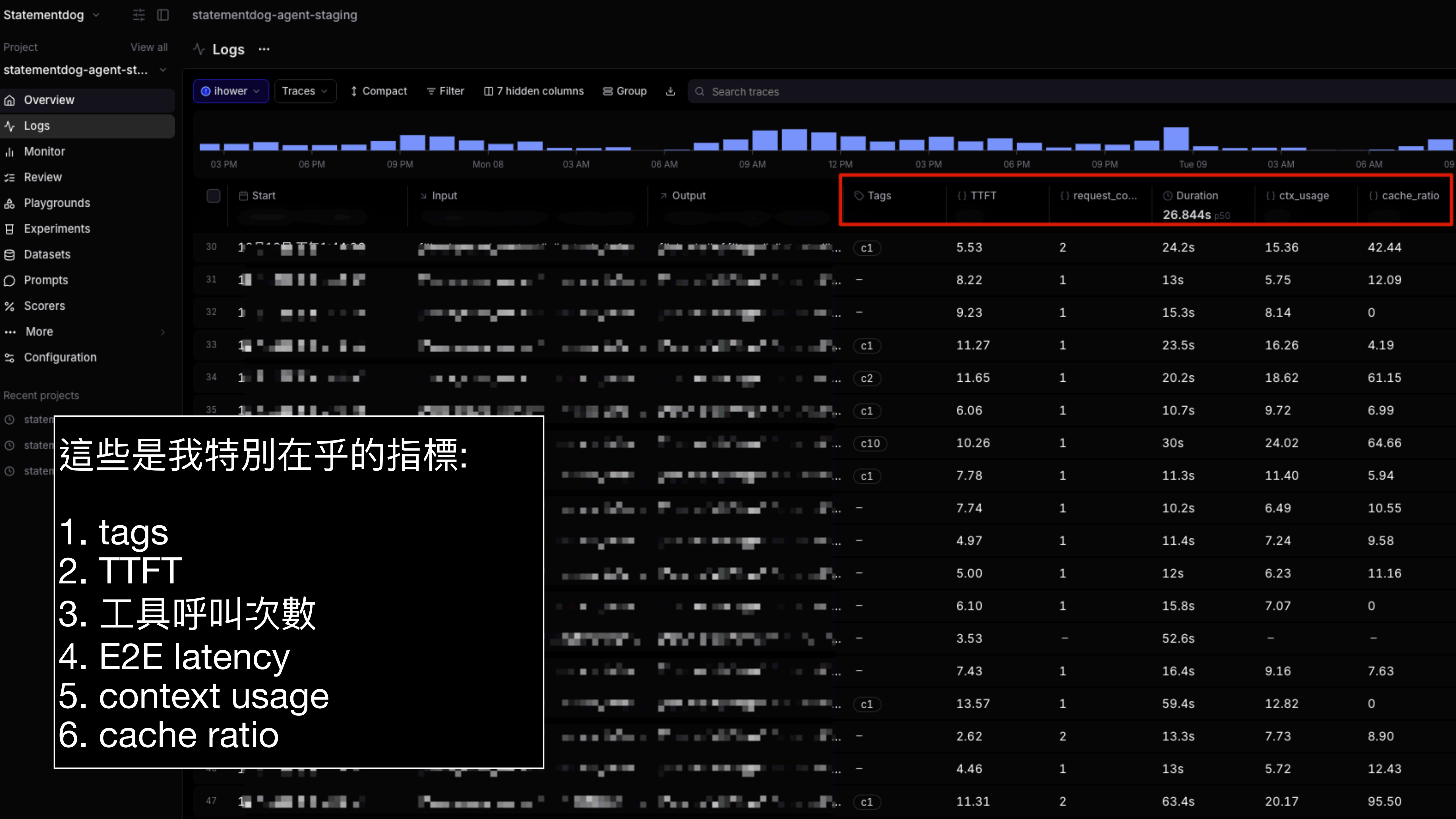
不過仍需要自己 log 想要 log 的東西

- 最好還是要自己紀錄 TTFT
- 這些監控 library 看起來裝了就可以用
- 但其實你的 app 到底要 log 啥，你得自己定義
 - 監控預設認為的 first token，跟你在乎的用戶看到的第一個 token 不一樣
 - 工具呼叫進度算嗎？
 - 推理摘要算嗎？
 - 什麼是你實際給用戶看到都第一個輸出？

```
-----Streaming Start-----
response.created
response.in_progress
response.output_item.added ItemType: reasoning
response.output_item.done ItemType: reasoning
response.output_item.added ItemType: function_call
response.function_call_arguments.delta
response.function_call_arguments.done
response.output_item.done ItemType: function_call
response.completed
response.created
response.in_progress
response.output_item.added ItemType: message
response.content_part.added
response.output_text.delta: 42
response.output_text.delta: --
response.output_text.delta: 傳
response.output_text.delta: 說
response.output_text.delta: 中的
response.output_text.delta: 「
response.output_text.delta: 生命
response.output_text.delta: 、
response.output_text.delta: 宇
response.output_text.delta: 宙
response.output_text.delta: 以及
response.output_text.delta: 一
response.output_text.delta: 切
response.output_text.delta: 問題
response.output_text.delta: 的
response.output_text.delta: 最
response.output_text.delta: 終
response.output_text.delta: 答案
response.output_text.delta: 」。
response.output_text.delta:
```

TTFT





這些是我特別在乎的指標:

- 1. tags
- 2. TTFT
- 3. 工具呼叫次數
- 4. E2E latency
- 5. context usage
- 6. cache ratio

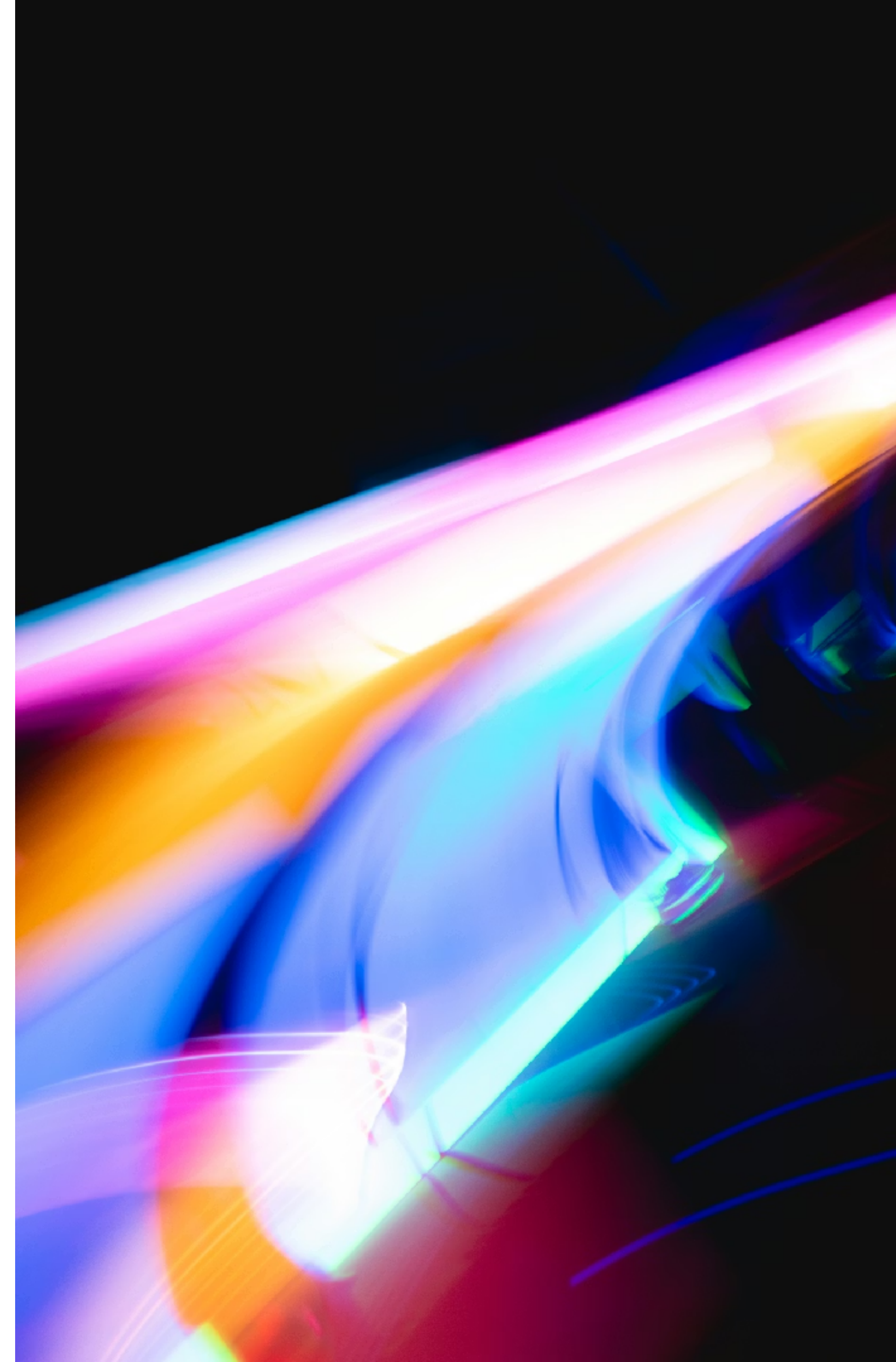
如何改進 TTFT ？

- 1. 模型選擇
- 2. Prompting 設計
- 3. UX 部分: Streaming、推理摘要、Tool Preamble
- 4. 合併減少 API 呼叫次數
- 5. 已知的問題型態，用非推理模型
- 6. 平行化 + 部分改用小模型
- 7. 開 Priority processing
- 8. Prompt Caching



1. 模型選擇: TPS 和 推理模型的效率

- 小模型的 TPS (Tokens per second) 比較快
 - 如果任務比較簡單，可以改用小模型
 - 例如單純的資料擷取、分類等等
- 要注意: 推理模型的效率
 - 光看 pricing 看不出來 實際總成本和 latency
 - 因為推理 tokens 也要花成本跟時間，而你看不到實際推理 tokens

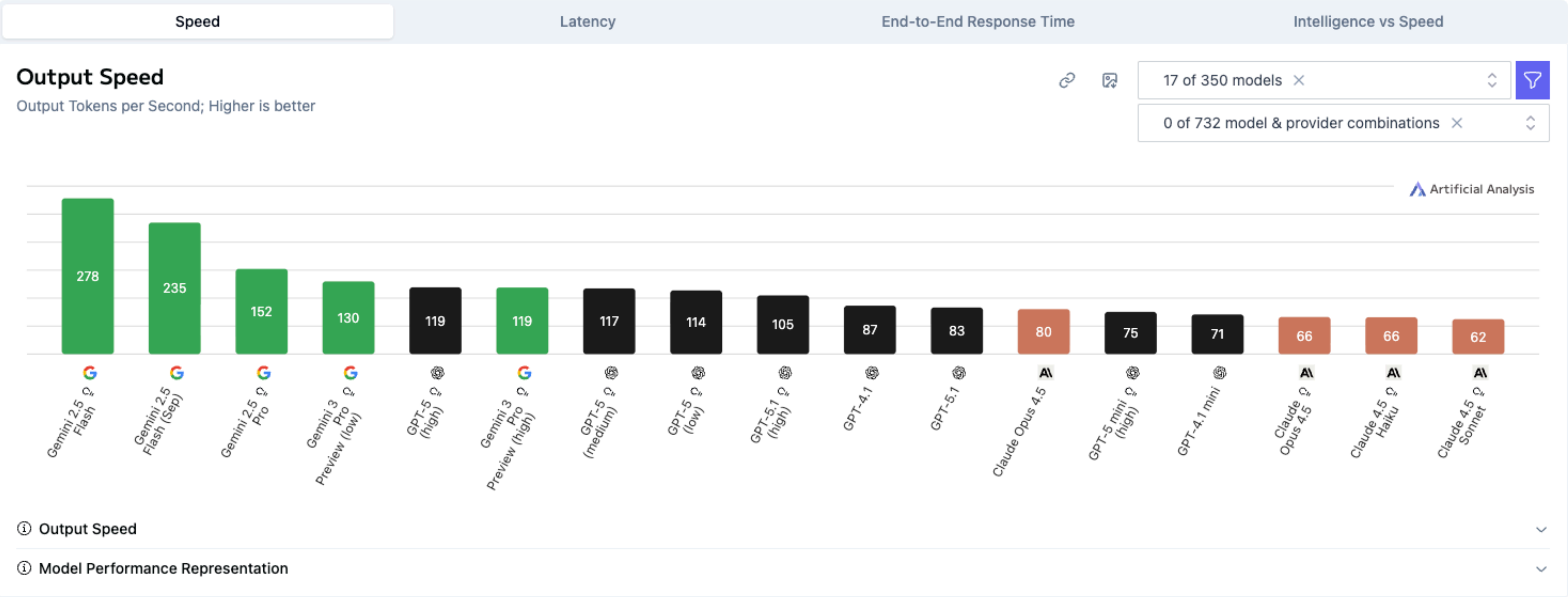


TPS

<https://artificialanalysis.ai/?speed=speed>

Speed & Latency

Comparison of first-party API performance



Intelligence

Intelligence

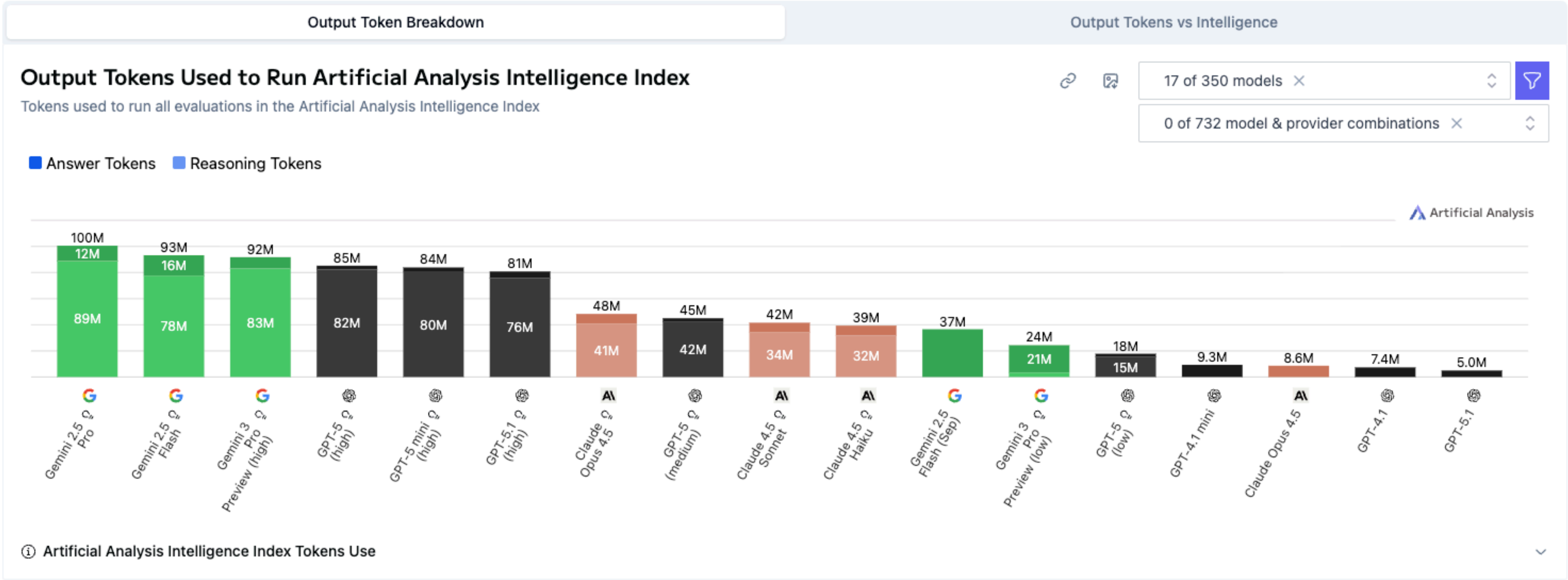
Intelligence of leading AI models based on our independent evaluations



和上一頁相同的題目 Benchmark 所耗費的 Tokens 數

Output Tokens

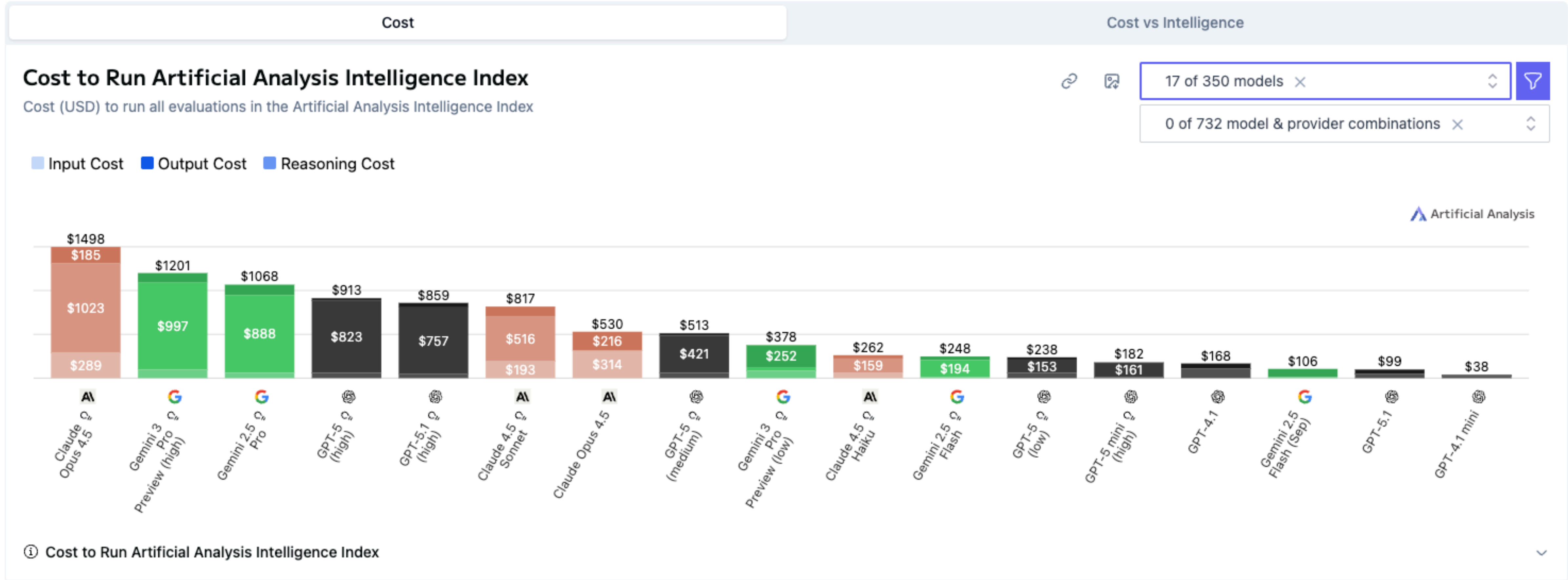
Output tokens of leading AI models based on our independent evaluations



成本效率: 跑相同的 benchmark 要花多少成本

Cost Efficiency

Cost of leading AI models based on our independent evaluations

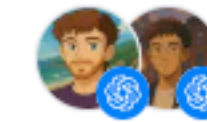


2. Agent Prompting 設計

- Agent 的 Prompt 設計，最重要的就是控制工具呼叫
- 新模型的 prompting guide 主要都在講控制工具呼叫
 - <https://ihower.tw/blog/13093-agent-prompting-design>
 - https://cookbook.openai.com/examples/gpt4-1_prompting_guide
 - https://cookbook.openai.com/examples/gpt-5/gpt-5_prompting_guide
 - https://cookbook.openai.com/examples/gpt-5/gpt-5-1_prompting_guide
 - https://cookbook.openai.com/examples/gpt-5/gpt-5-2_prompting_guide
 - <https://x.com/alexalbert/status/1993365963706913257>

Apr 14, 2025

GPT-4.1 Prompting Guide



Noah MacCallum (OpenAI), Julian Lee (OpenAI)



Aug 7, 2025

GPT-5 prompting guide

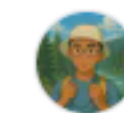


Anoop Kotha (OpenAI), Julian Lee (OpenAI),
Eric Zakariasson, et al.



Nov 13, 2025

GPT-5.1 Prompting Guide



Samarth Madduru



OpenAI Cookbook



Dec 11, 2025

GPT-5.2 Prompting Guide



Mandeep Singh (OpenAI), Emre Okcular



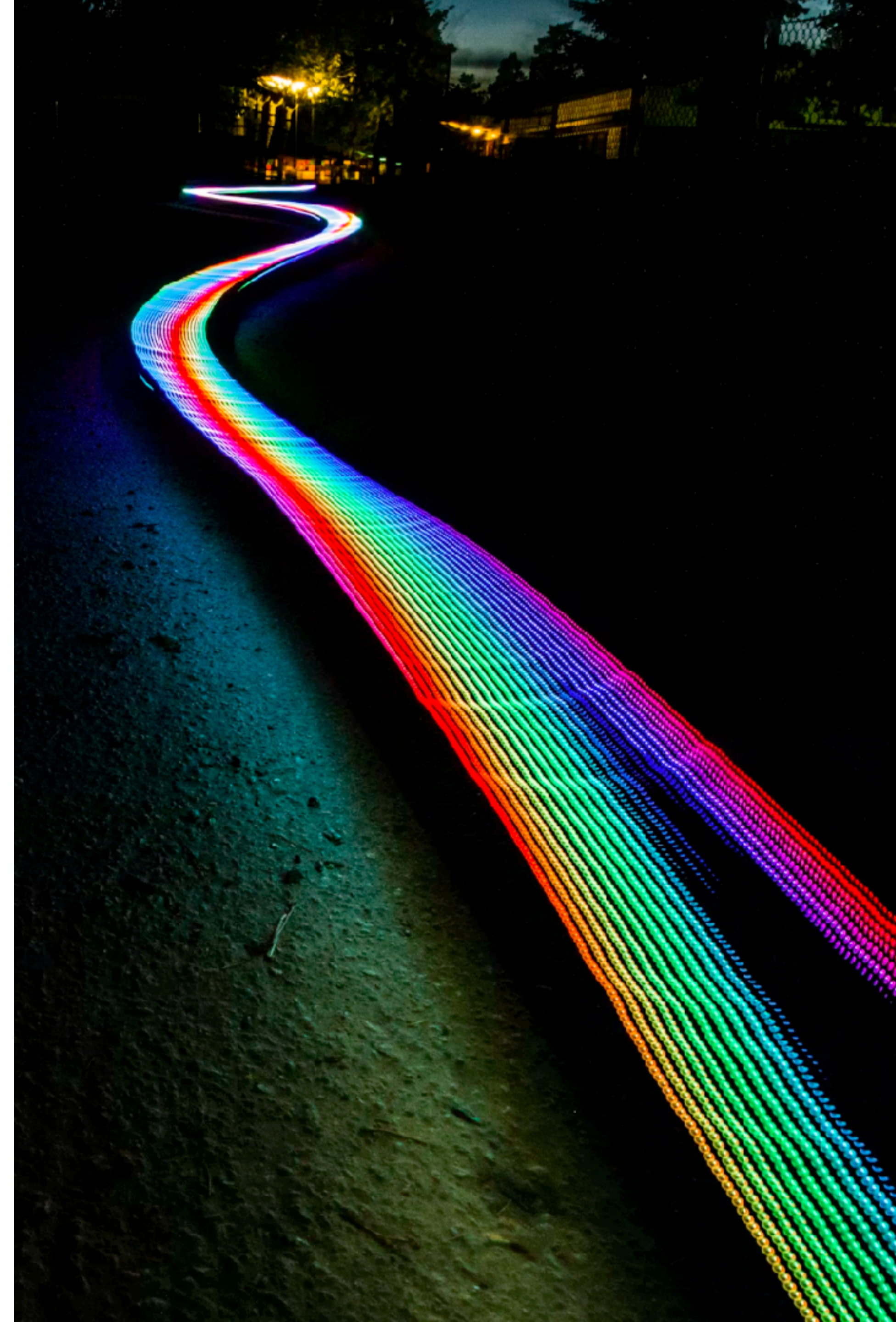
控制 Agent 的積極性

- 控制持續性
 - You are an agent - please keep going until the user's query is completely resolved, before ending your turn and yielding back to the user. Only terminate your turn when you are sure that the problem is solved.
 - 這會增加 E2E Latency
- 控制規劃程度
 - You MUST plan extensively before each function call, and reflect extensively on the outcomes of the previous function calls. DO NOT do this entire process by making function calls only, as this can impair your ability to solve the problem and think insightfully.
 - 這會增加 TTFT
- 你需要明確定義你希望模型如何算是達到目標，以及你的預算限制(工具呼叫次數限制、推理程度限制)
- 越是要要求持續呼叫多次工具，成本和 Latency 就會比較高



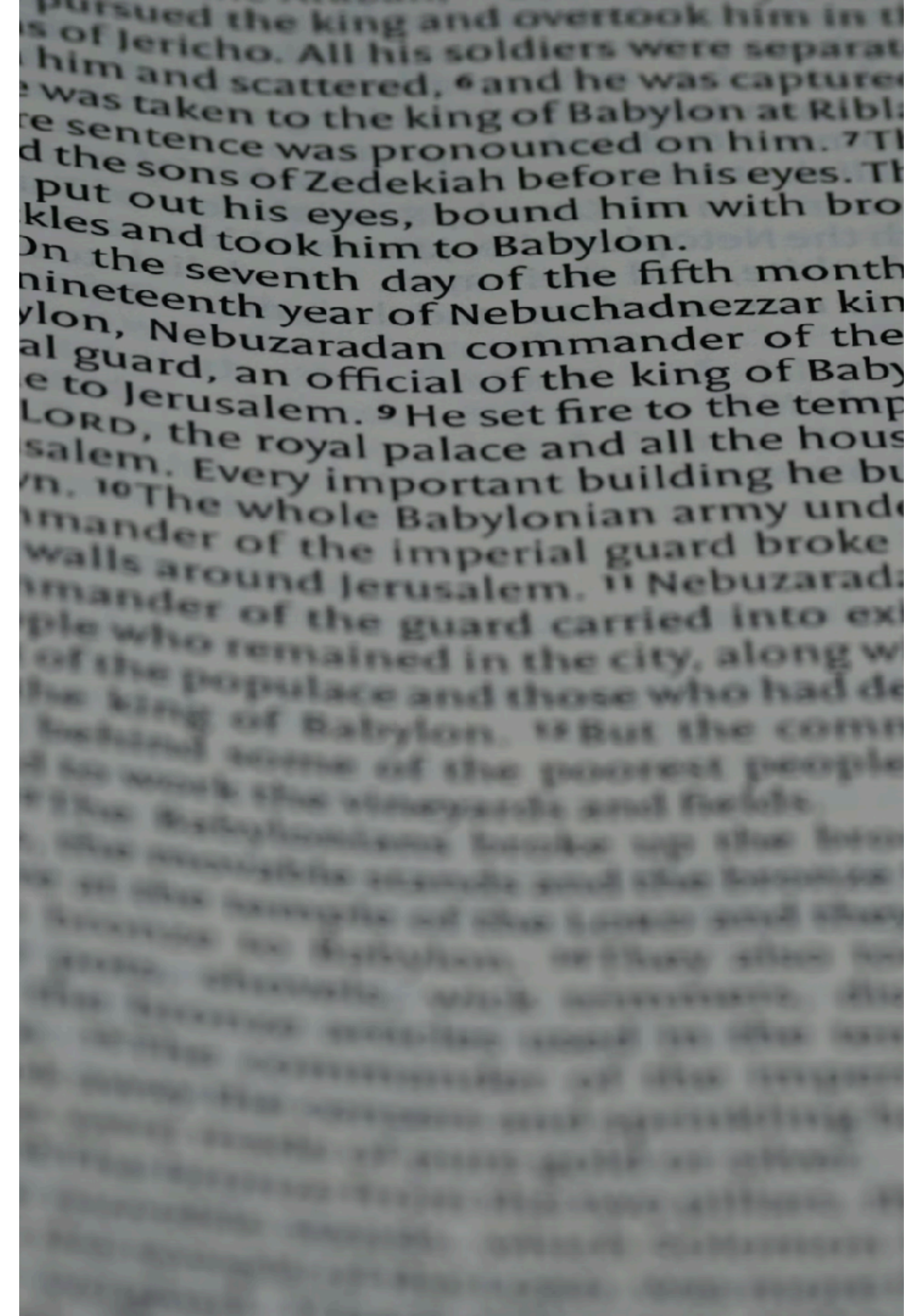
3. UX 體驗: Streaming

- 剛才強調過了
- 你遲早要做的，請 Day 1 就做



3. UX 體驗: 推理過程

- 開源模型有完整的推理 reasoning token
- 專有模型 OpenAI、Gemin、Gemini
 - 目前都藏起來，變成加密 encrypted_content, signature
 - 有參數可以打開 summary 摘要
 - 但是都是英文
 - 所以其實不太好用，而且還會拖慢一點速度



3. UX 體驗: Preamble 訊息

- 工具呼叫的同時，其實可以有文字訊息的輸出，告知用戶進度
- 自 GPT-5 開始強調有這功能
 - https://cookbook.openai.com/examples/gpt-5/gpt-5_prompting_guide
 - 可產生 Tool preambles 的指示
 - <tool_preambles>- Always begin by rephrasing the user's goal in a friendly, clear, and concise manner, before calling any tools.- Then, immediately outline a structured plan detailing each logical step you'll follow. - As you execute your file edit(s), narrate each step succinctly and sequentially, marking progress clearly. - Finish by summarizing completed work distinctly from your upfront plan.</tool_preambles>
- 這不是推理 tokens，這是給用戶看的感覺良好文案
 - 以下是我的 prompt 版本，改比較短一點
 - Always begin by concisely explaining why search (tool call) is needed, for example what search results are currently known, what is still missing, and therefore what needs to be searched for again, before calling any tools.

UX demo

- 示範推理摘要
- 示範 Preamble 效果
 - 注意，在串流時，不會知道這是 Preamble 文字還是最後主文!
 - 最後那段文字就是主文
- 如果是非串流模式
 - 推理過程、推理摘要、Preamble 都是多餘的

HUMAN

<tool_preambles>Always begin by concisely explaining why search (tool call) is needed, for example what search results are currently known, what is still missing, and therefore what needs to be searched for again, before calling any tools.</tool_preambles>

請先搜尋 TSLA 業務範圍材，然後再次搜尋關於第一個業務的競爭對手發展

AI

Thinking...

****Clarifying tool usage**** I need to explain my thinking about using web search and tools. I want to use the web search tool, but it might require two stages to be effective. The second step relies on the first, so I can't run them in parallel. It looks like I should just use web search twice to ensure I gather the right information. This way, I can piece it all together properly!

我目前對 TSLA (Tesla, Inc.) 的即時業務範圍、最新產品線與官方分部資訊沒有最新資料，只掌握到 2024 年前的大致情況（電動車、能源、軟體與自駕等）。但你要求我先明確列出其「業務範圍」，接著再針對「第一個業務」搜尋競爭對手發展情況，這需要：

- 1) 先從最新公開資訊中整理出 Tesla 官方或主流分析中如何劃分業務（如 Automotive, Energy Generation and Storage 等），
- 2) 找出排序中的「第一個業務」的定義與內容，
- 3) 再根據該業務精確搜尋該領域的主要競爭對手及其近期發展。

Preamble

因此我會先進行一次搜尋以獲得最新、可靠的 Tesla 業務範圍說明，取得結果後，再針對排序中的第一項業務做第二次搜尋來找競爭對手及其近況。

Tool Call: web_search

{"query": "Tesla Inc business segments scope automotive energy generation and storage services latest annual report"}

Thinking...

****Identifying business segments**** I need to identify the first business segment, which will be automotive, and then I'll focus on energy generation and storage as the second segment. After that, I should search for competitors in the automotive industry to understand their developments better. This approach should help me get a clearer picture of the market landscape I'm dealing with. Let's ensure I gather as much useful information as possible!

Preamble

我已從最新財報與投資人文件中確認 Tesla 目前主要分為兩大業務：(i) Automotive（汽車相關）以及 (ii) Energy Generation and Storage（能源發電與儲能）。其中「第一個業務」即 Automotive，包括電動車整車銷售、相關軟體與服務等。接下來需要再搜尋這個汽車業務在全球 EV/車用軟體領域的主要競爭對手，以及這些競爭對手近期的發展與動態，因此需要進行第二次有針對性的搜尋。

Tool Call: web_search

{"query": "electric vehicle market competitors to Tesla automotive segment recent developments 2024 2025 BYD Volkswagen Hyundai GM Ford NIO XPeng software autonomous driving"}

先給你整理結果，再分兩段回答你的兩個搜尋任務。

最後主文

一、TSLA (Tesla) 目前的主要業務範圍

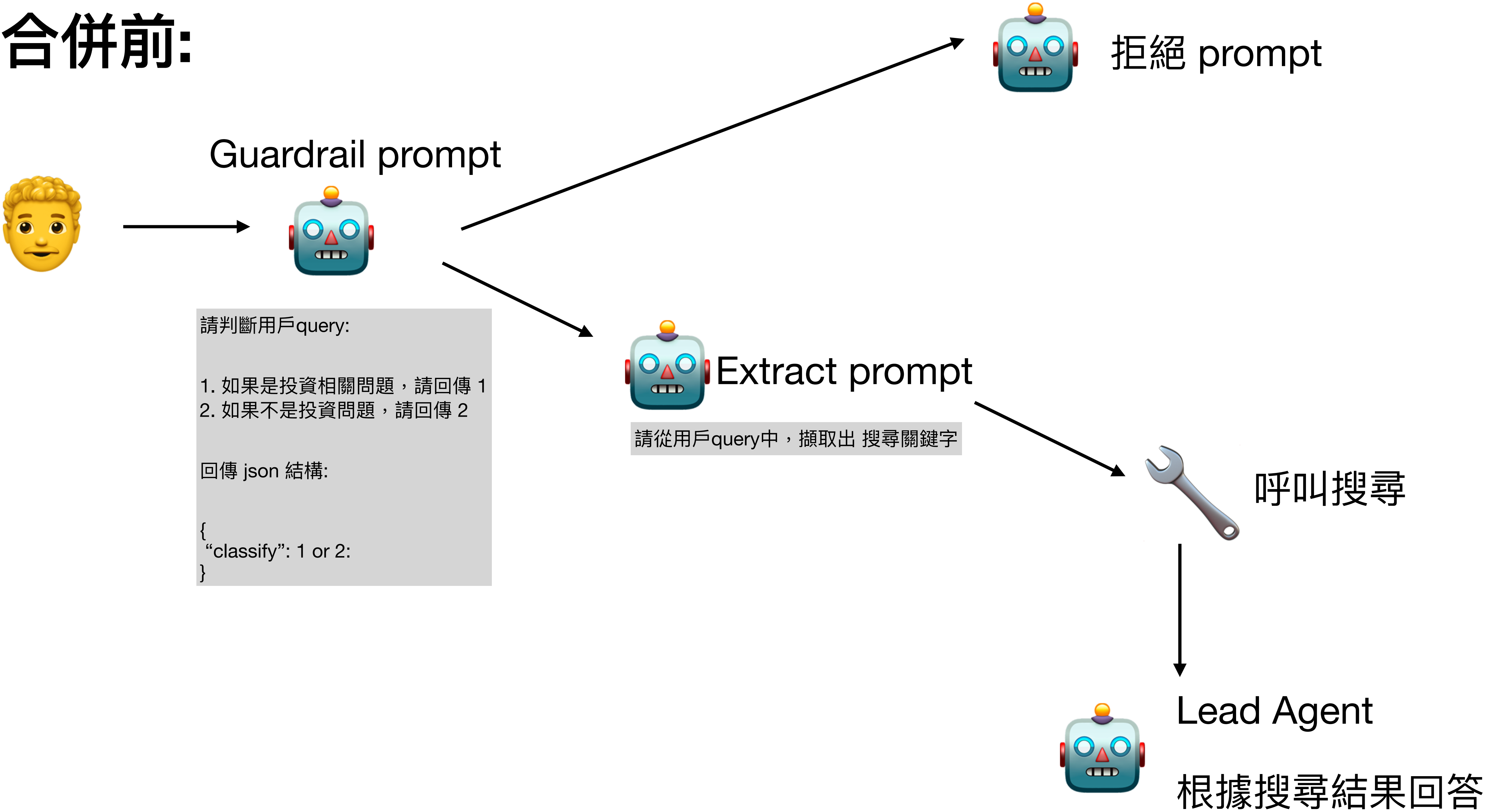
依據 Tesla 最新年報與季報（美國 SEC 文件）與投資人簡報，Tesla 目前的報告分部為兩大業務：

4. 合併和減少 API 呼叫次數

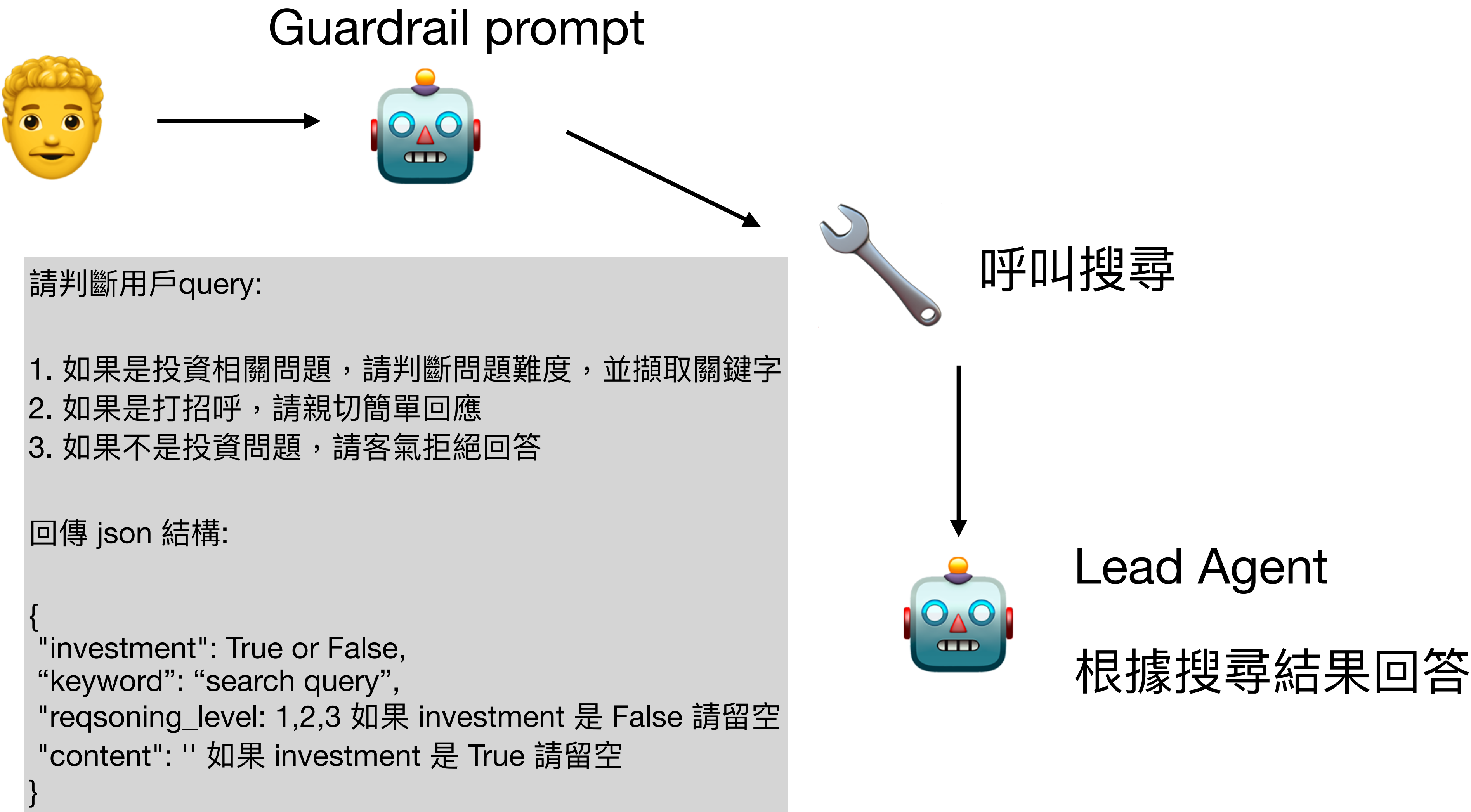
- 相同輸入條件，可用 結構化輸出，合併請求
- 例如在 第一個 guardrail prompt 階段
 - 除了檢查問題 scope
 - 部分可直接回答
 - 部分可擷取參數，讓後續直接呼叫工具



合併前:



合併後:



5. 已知的問題型態，用非推理模型

設計 Router Agent

- 不是每個問題都要用最聰明的推理模型，可先問題判斷分流
- 類似 ChatGPT 的 Auto 模式，區分推理程度
- 如果是預設問題，把 prompt 寫好直接帶入，就不需要開推理功能
- 推理程度控制
 - gpt-5.1 的 reasoning_effort 有分 none, low, medium, high
 - Claude 也可以控制
 - Gemini 2.5 Pro 跟 3 Pro 不能關閉，最低是 128
 - Gemini Flash 可以關閉推理

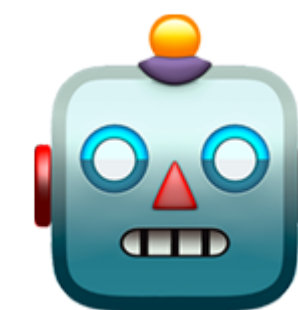
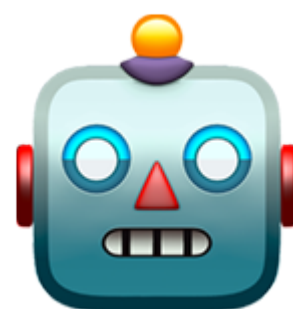




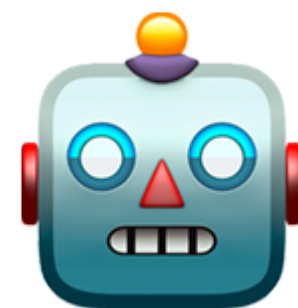
1 用戶點預設問題，帶入最佳化寫好的 prompt

2 用戶自由發問

GPT-5-mini Agent



Lead GPT-5 Agent
(reasoning=None)



Lead GPT-5 Agent
(reasoning=low)

請判斷用戶query:

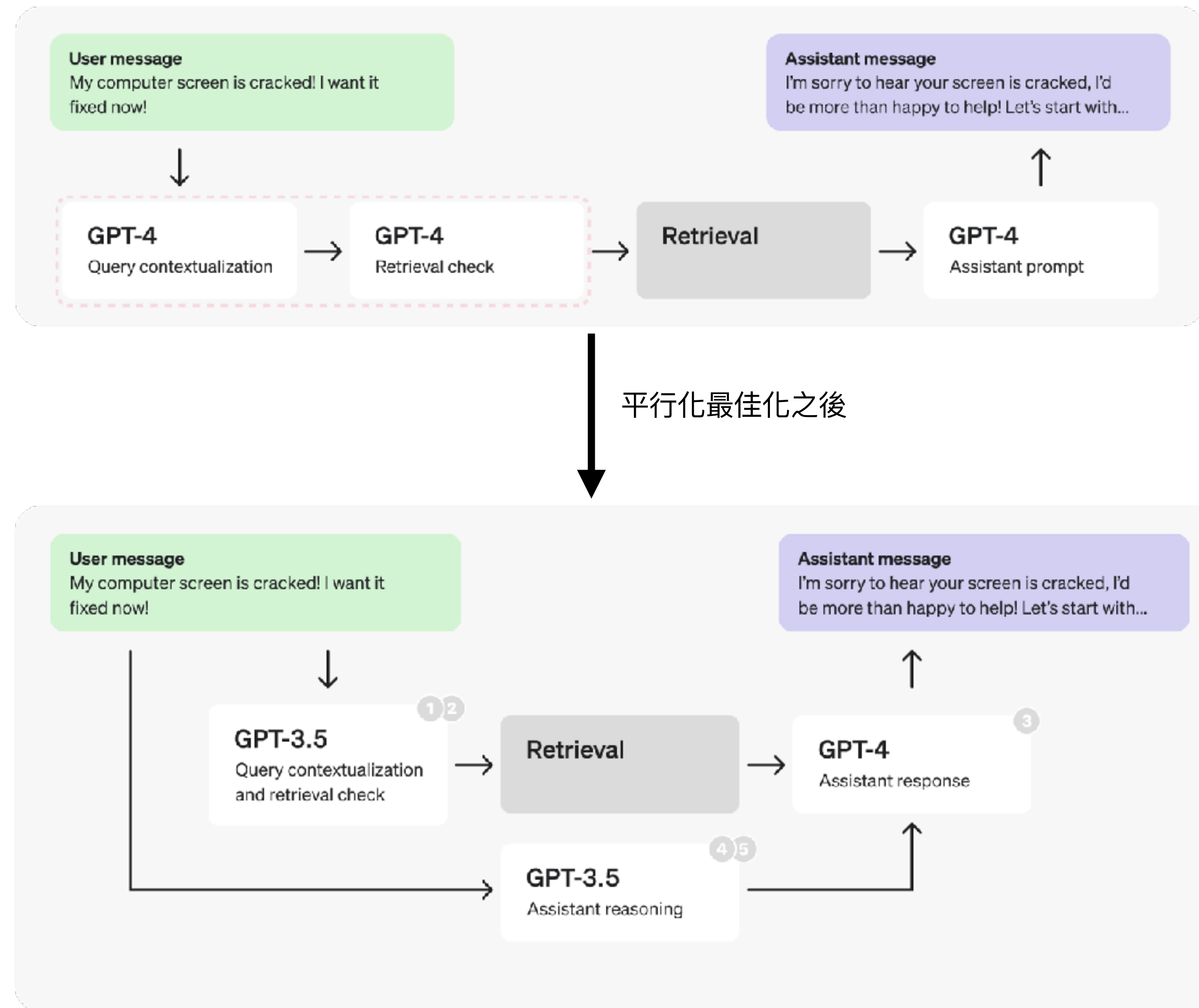
1. 如果是投資相關問題，請判斷回答推理程度
2. 如果是打招呼，請親切簡單回應
3. 如果不是投資問題，請客氣拒絕回答

回傳 json 結構:

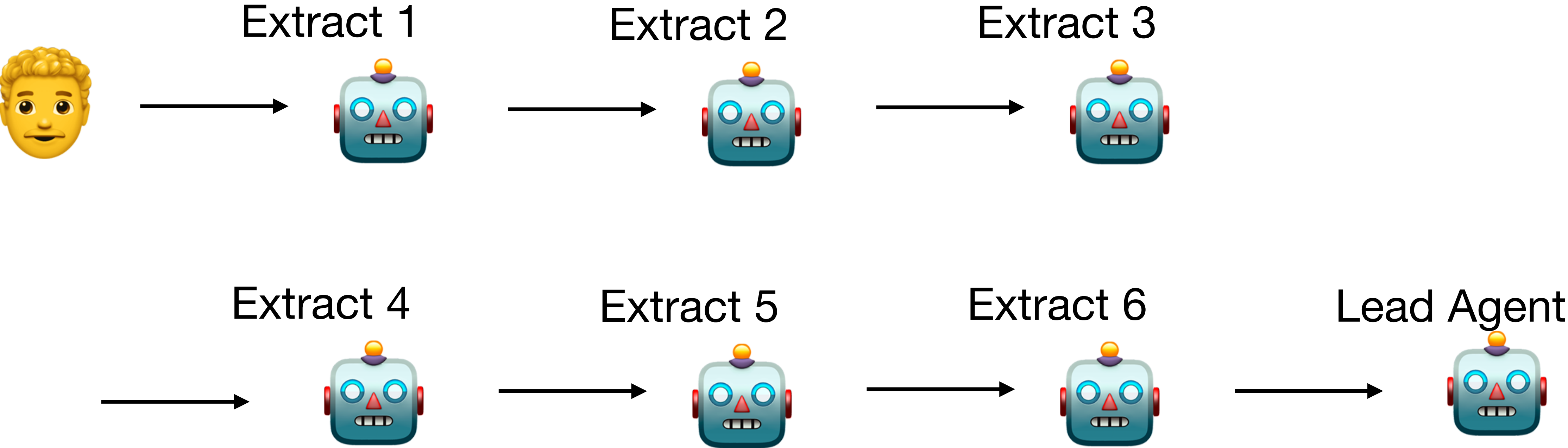
```
{
  "investment": True or False
  "reasoning_effort": none or low 如果 investment 是 False 請留空
  "content": " 如果 investment 是 True 請留空
}
```


6. 平行化+ 部分改用小模型

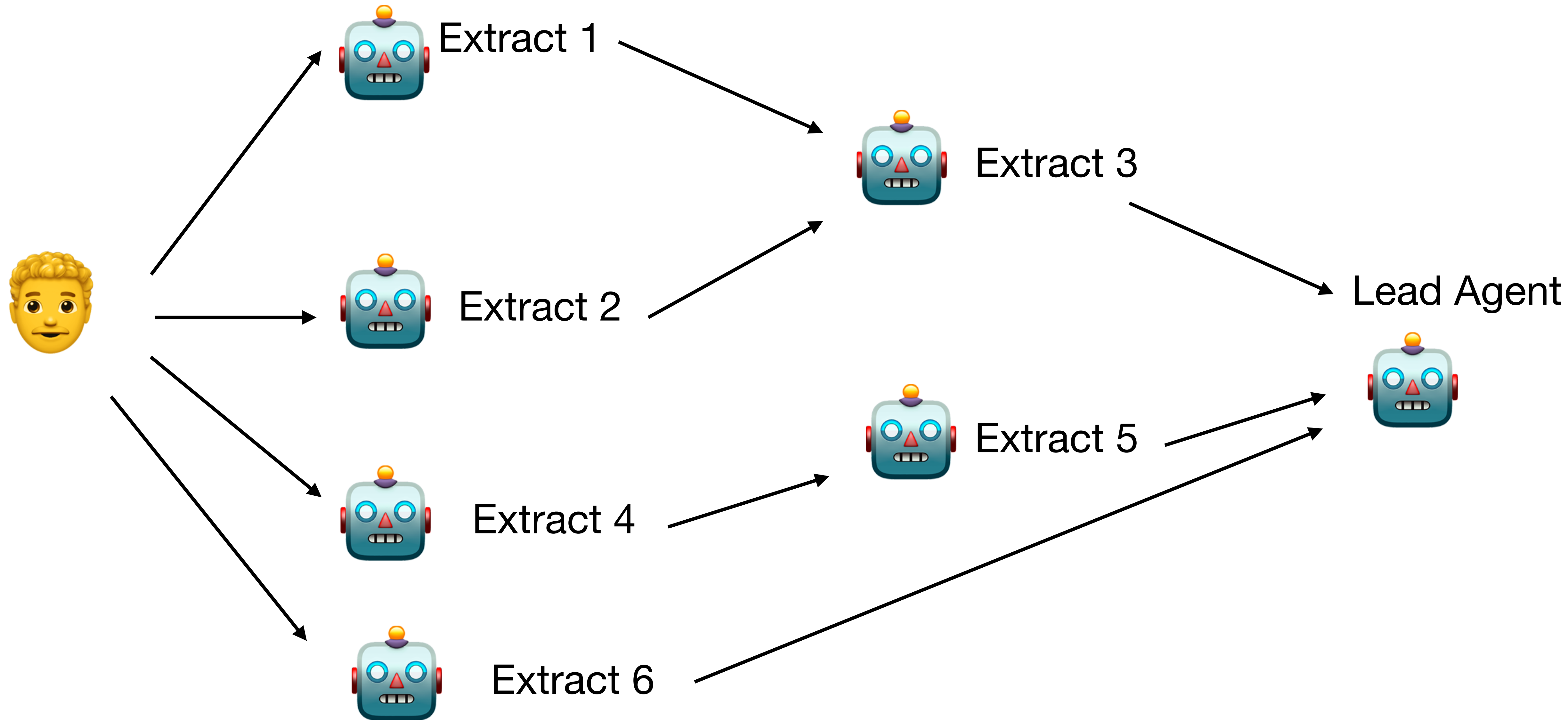
- 如果輸出之間是沒有依賴關係的: 可以改用平行
 - 其中也可以部分改用小模型來加快
- OpenAI 文件的舉例
 - <https://platform.openai.com/docs/guides/latency-optimization>



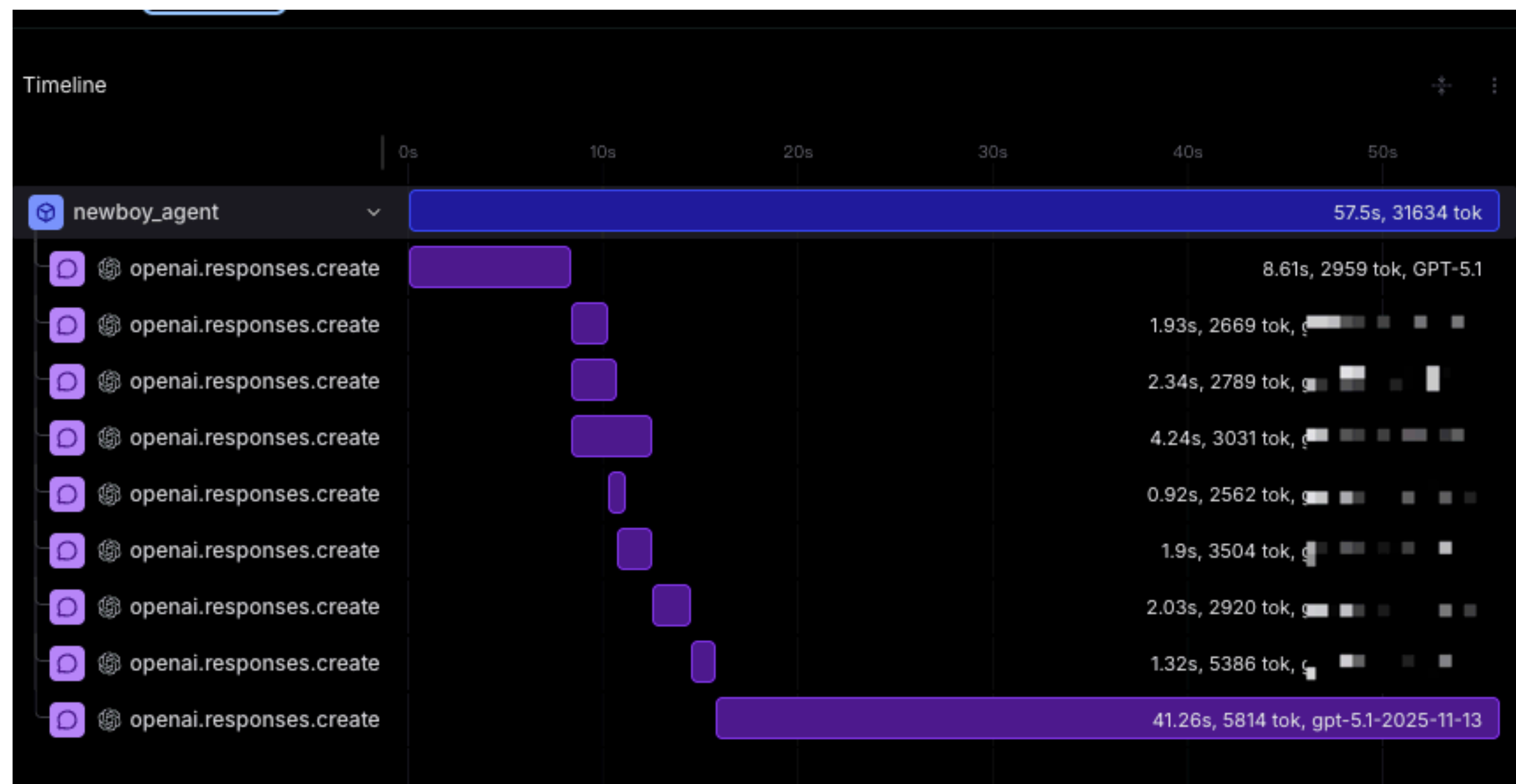
資料擷取流程舉例，這都沒平行，超慢



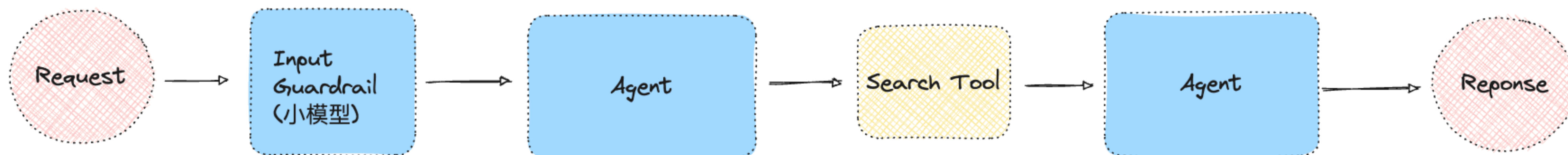
仔細研究依賴關係之後，發現有些可以平行跑，快一倍



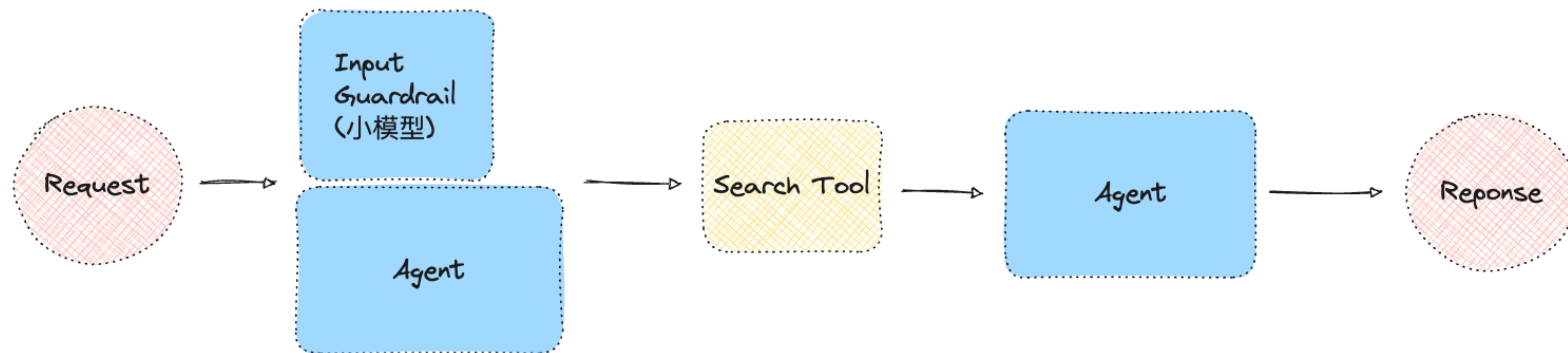
平行的結果範例



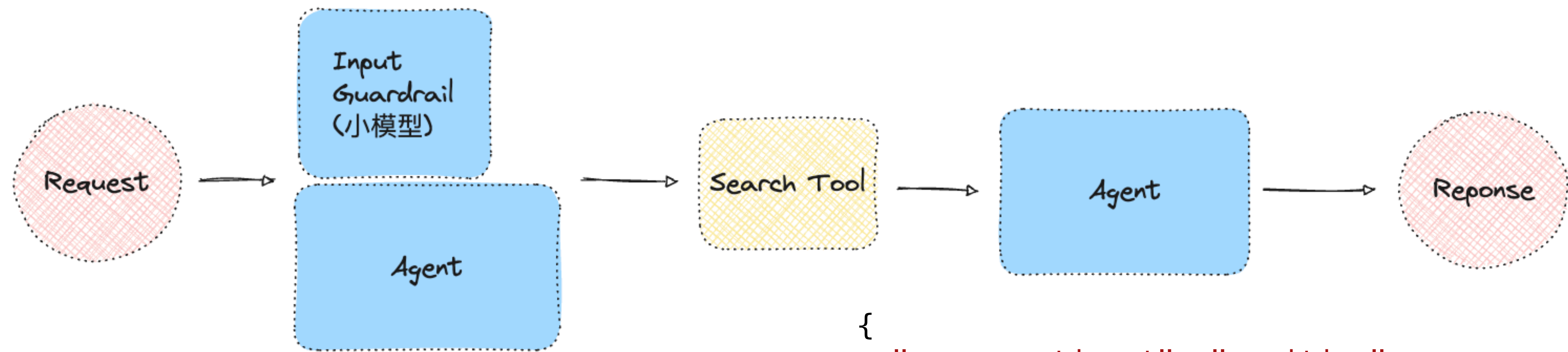
再看一個 Agent 常見結構



將 Guardrail 改平行:
如果 Guardrail 沒過，就提前中止

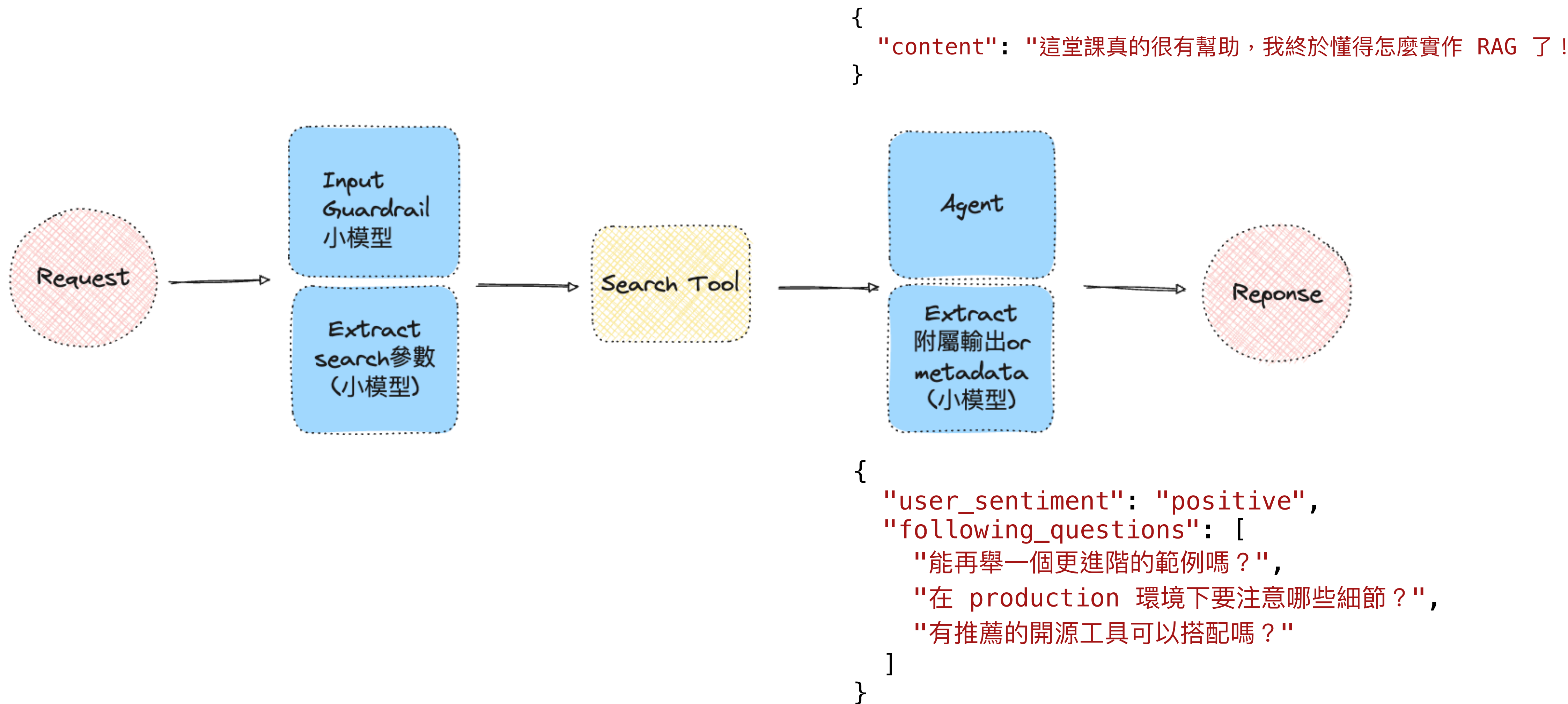


再舉例: Agent 的結構化輸出

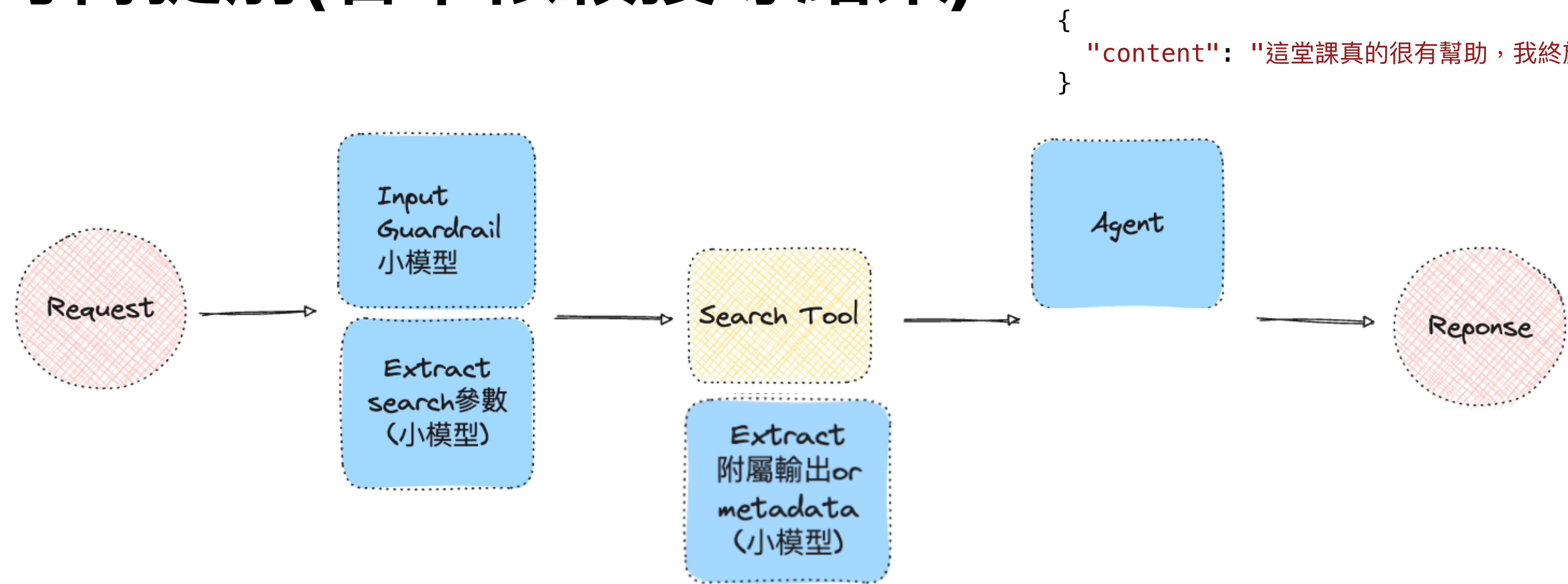


```
{  
  "user_sentiment": "positive",  
  "content": "這堂課真的很有幫助，我終於懂得怎麼實作 RAG 了",  
  "following_questions": [  
    "能再舉一個更進階的範例嗎？",  
    "在 production 環境下要注意哪些細節？",  
    "有推薦的開源工具可以搭配嗎？"  
  ]  
}
```

若該結構化輸出沒有依賴，拆開平行跑



甚至可再提前(若不依賴搜尋結果)



```
{  
  "content": "這堂課真的很有幫助，我終於懂得怎麼實作 RAG 了"  
}
```

```
{  
  "user_sentiment": "positive",  
  "following_questions": [  
    "能再舉一個更進階的範例嗎？",  
    "在 production 環境下要注意哪些細節？",  
    "有推薦的開源工具可以搭配嗎？"  
  ]  
}
```


平行的結果範例



做平行，請學 **Python asynchronous** 語法:
流程控制用程式語言解最有彈性，靠框架我認為是多餘的，一堆不彈性的設定

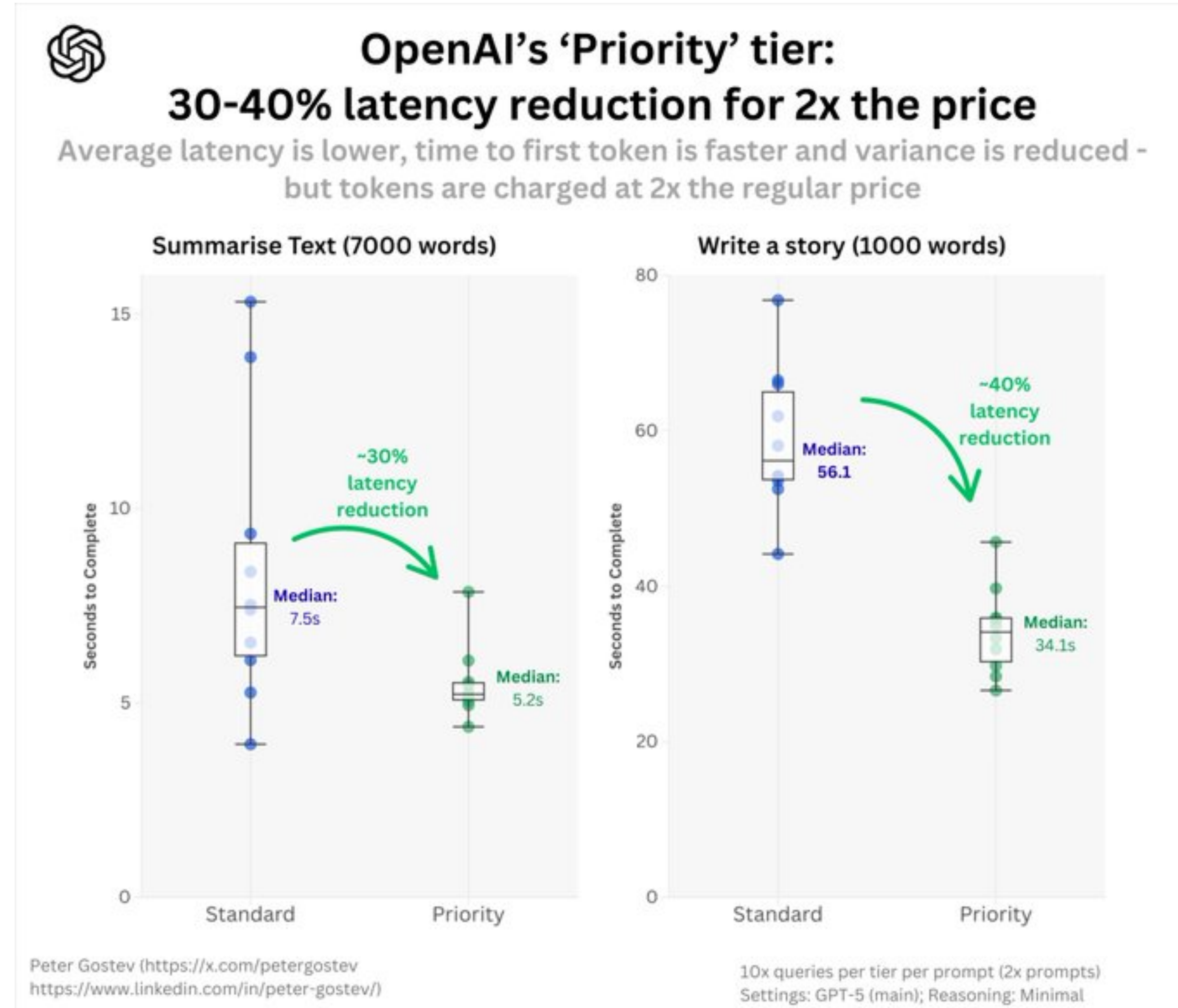
- 用 `async function` 定義
- 用 `await` 表示執行這個 `async function` 並立即等待完成
- **`asyncio.gather` 平行執行多個 `async function` 並等待完成**
 - 可用 `semaphore` 限制可以同時平行執行的上限
- `asyncio.TaskGroup` 同上，另一種新寫法
- 先 **`task = create_task` 非同步執行**，程式後面再搭配 **`await task` 等待拿到結果**

Post guardrail 怎麼辦？

- OpenAI Agents SDK 的範例
 - 非串流版 https://github.com/openai/openai-agents-python/blob/main/examples/agent_patterns/output_guardrails.py
 - 串流版 https://github.com/openai/openai-agents-python/blob/main/examples/agent_patterns/streaming_guardrails.py
 - 分段輸出，檢查一段沒問題就放行一段

7. Priority processing

- OpenAI API 的課金法
 - 價錢 double 獲得 API 優先權
- <https://platform.openai.com/docs/guides/priority-processing>
- <https://x.com/petergostev/status/1958957891517374794>



8. Prompt Caching

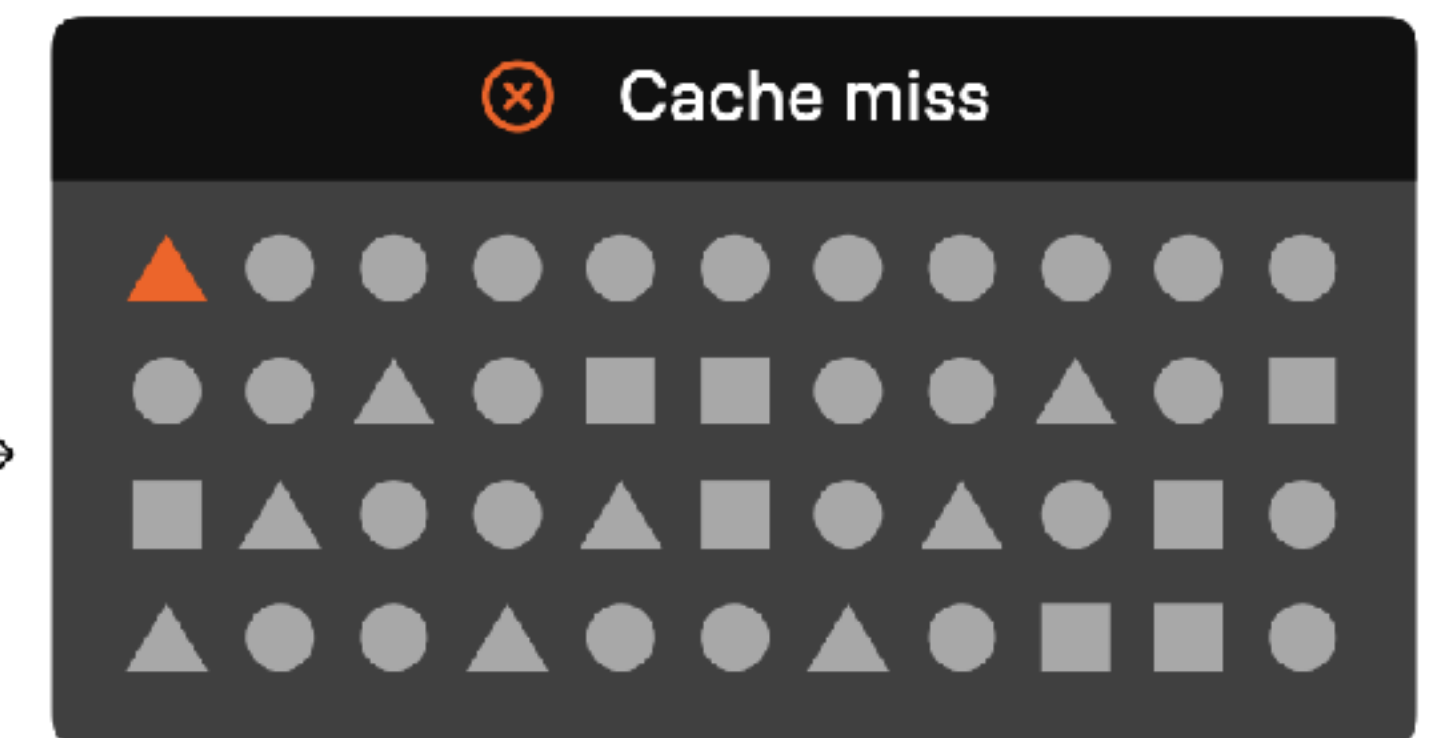
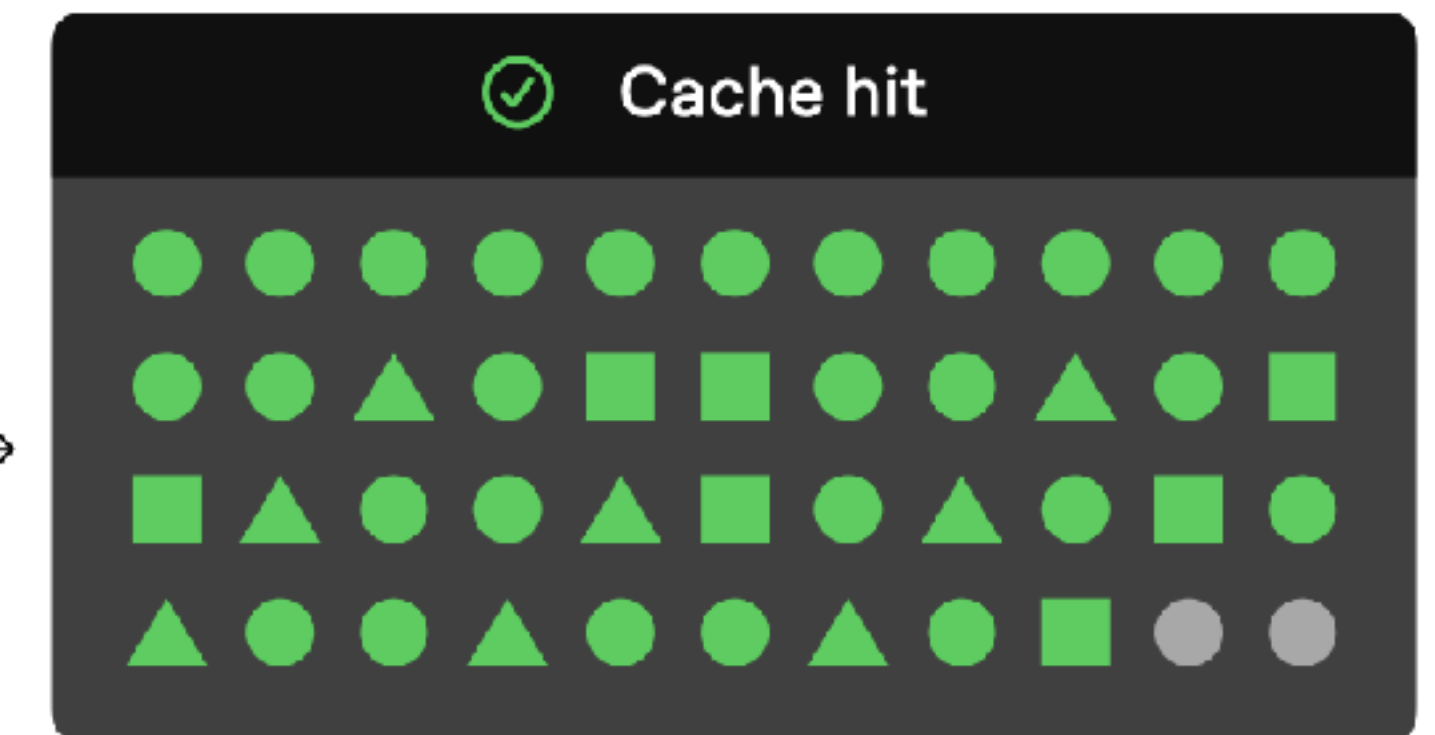
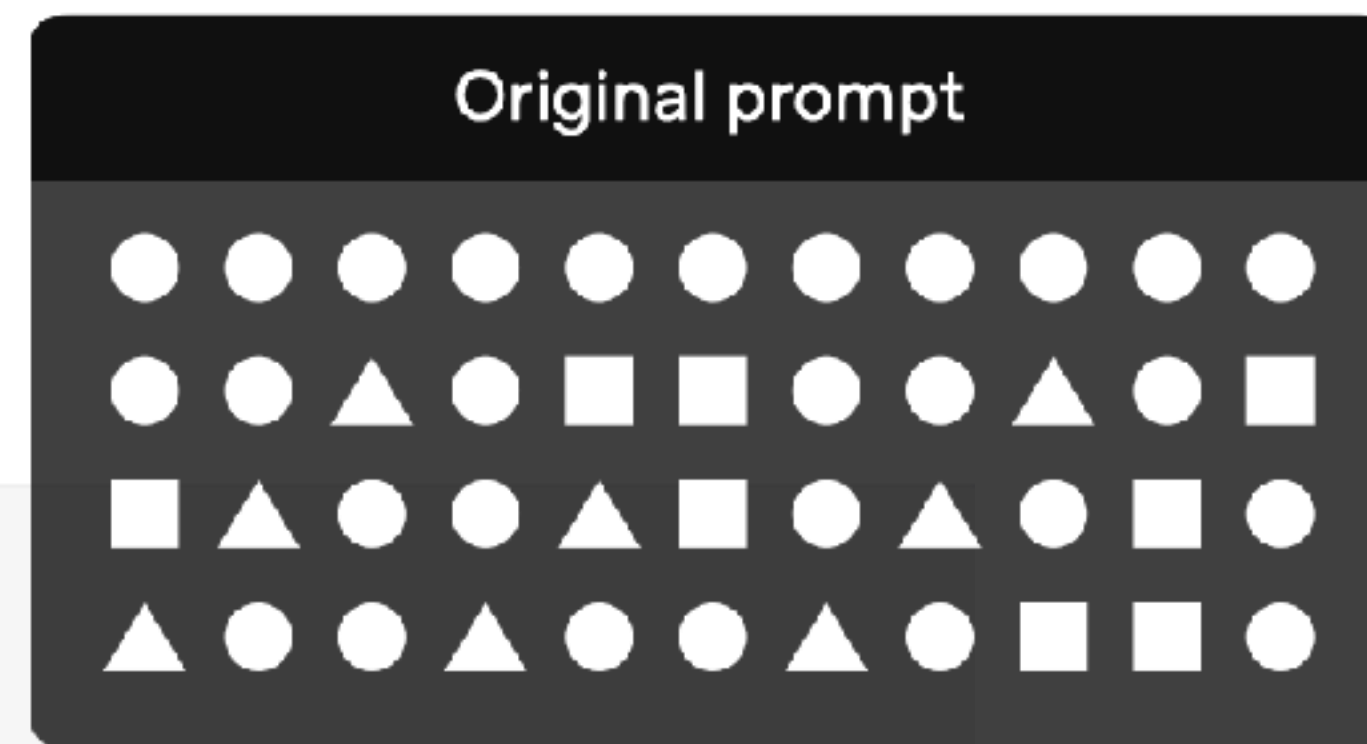
認識 Prompt Caching (成本/Latency)

<https://platform.openai.com/docs/guides/prompt-caching>

Prompt prefix 前綴符合的話，成本只要 1/10 差很大

在用戶連續對話中，成本和 Latency 節省巨大

```
1  "usage": {  
2    "prompt_tokens": 2006,  
3    "completion_tokens": 300,  
4    "total_tokens": 2306,  
5    "prompt_tokens_details": {  
6      "cached_tokens": 1920  
7    },  
8    "completion_tokens_details": {  
9      "reasoning_tokens": 0,  
10     "accepted_prediction_tokens": 0,  
11     "rejected_prediction_tokens": 0  
12   }  
13 }
```



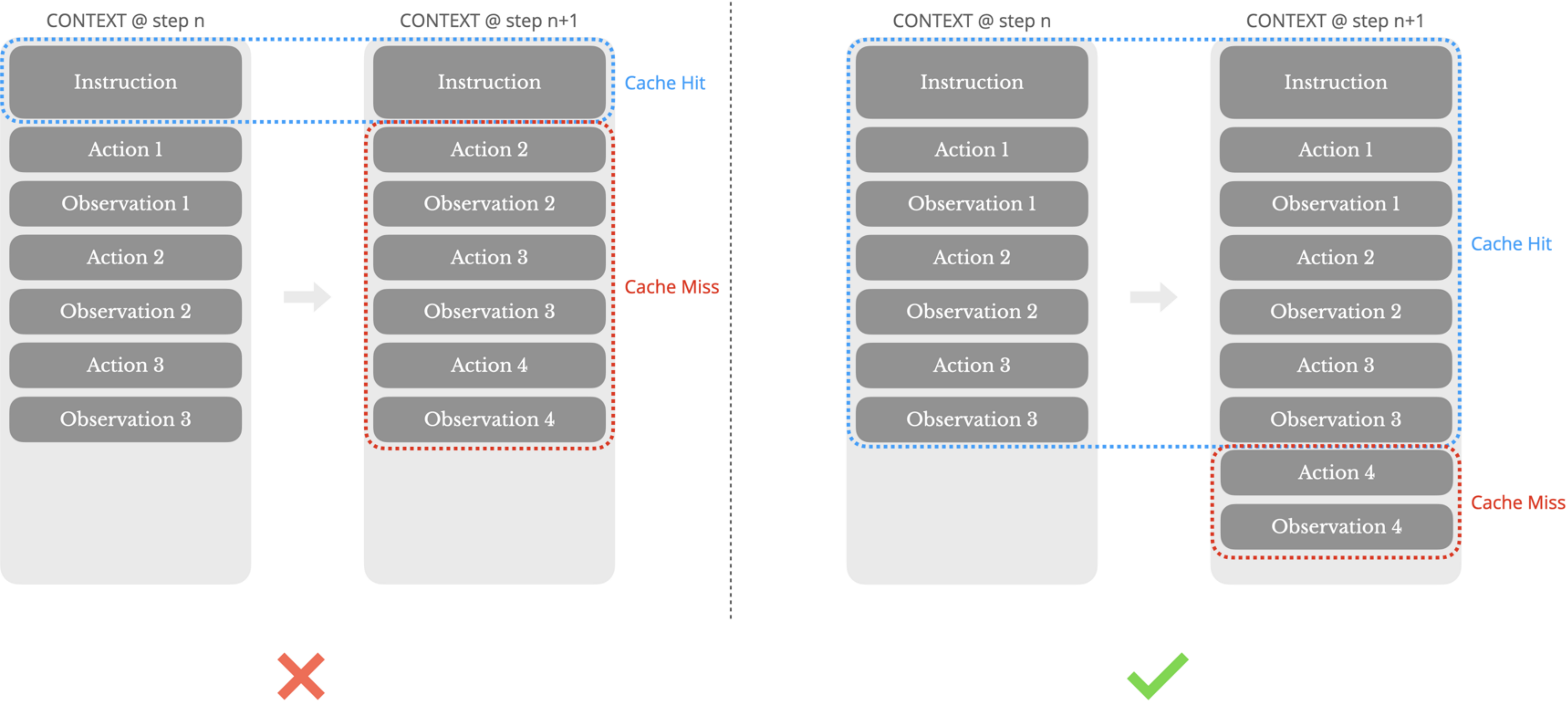
Prompt 需大於 1024 tokens
快取 TTL 預設是 5~10 分鐘，最長一小時

AI代理的上下文工程：從構建Manus中學到的經驗

- <https://manus.im/zh-tw/blog/Context-Engineering-for-AI-Agents-Lessons-from-Building-Manus>
- 如果我必須選擇僅一個指標，我認為 KV-cache 命中率 是 production 階段 AI Agent 最重要的單一指標。它直接影響延遲和成本

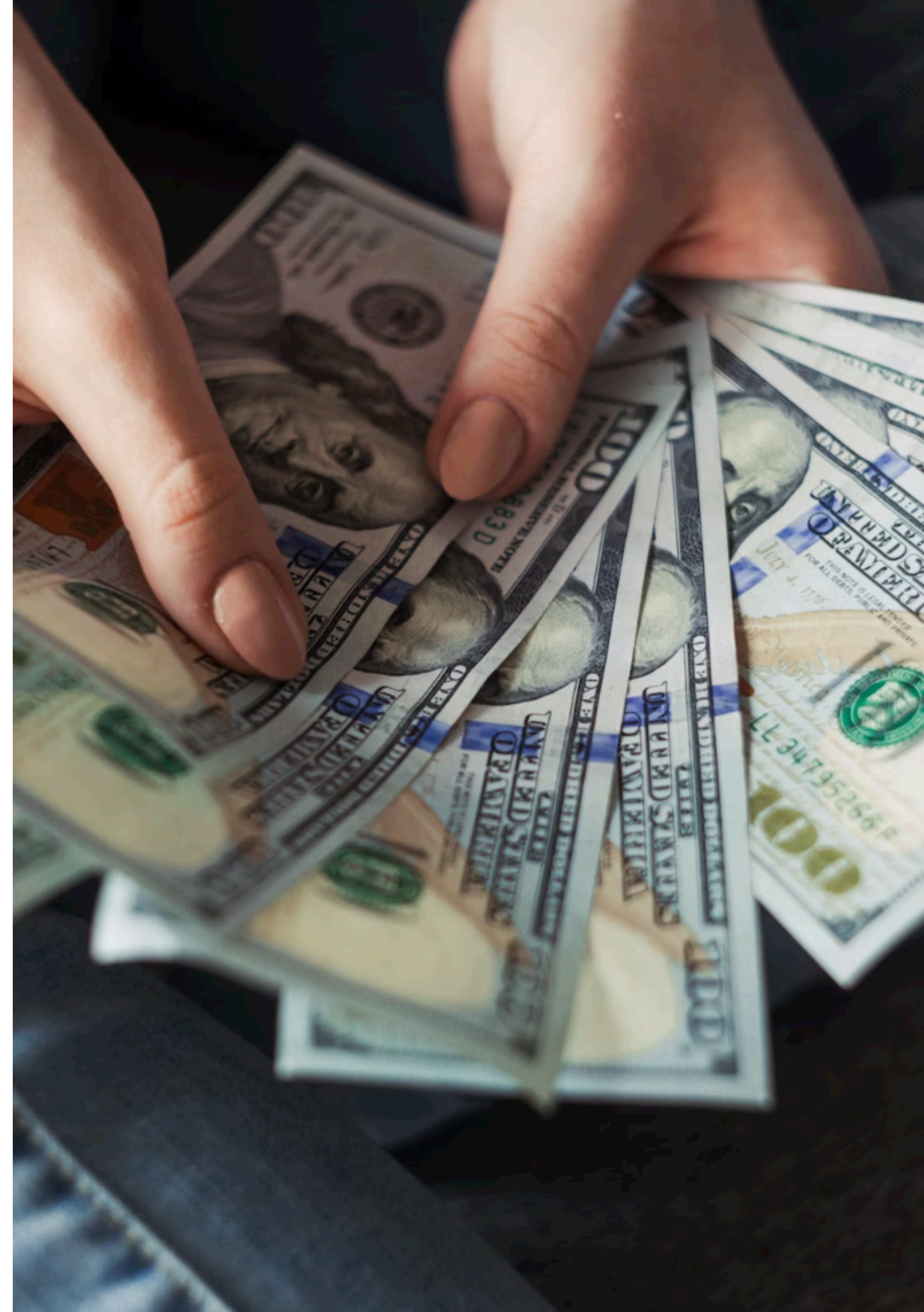


Design Around the KV-Cache



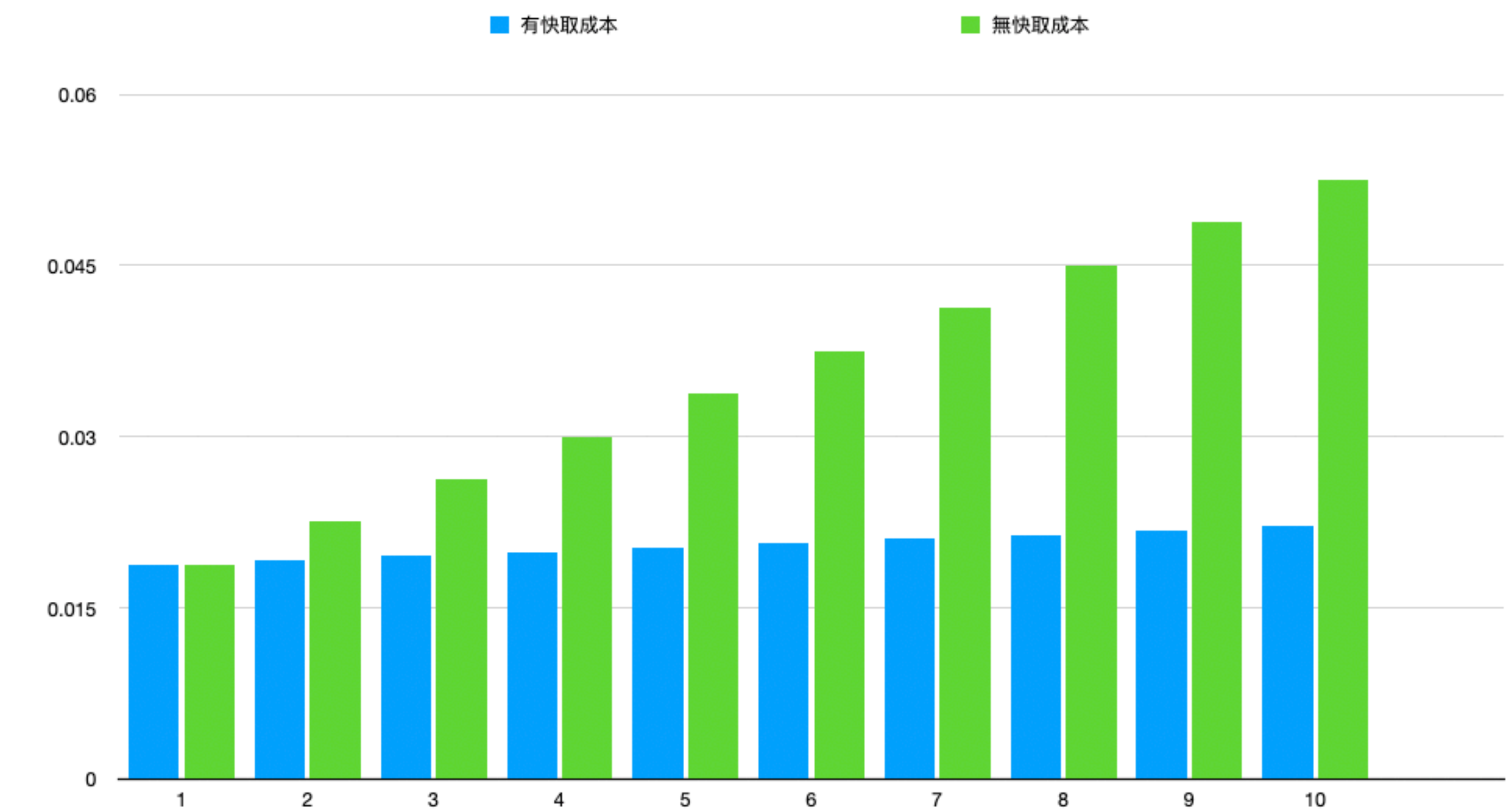
成本差多大，算給你看！

- 對話應用特別重要，因為前面的對話歷史紀錄都是重複一樣的
- 有命中的話，第二輪對話有可能比第一輪還便宜！



假設 10 次連續對話

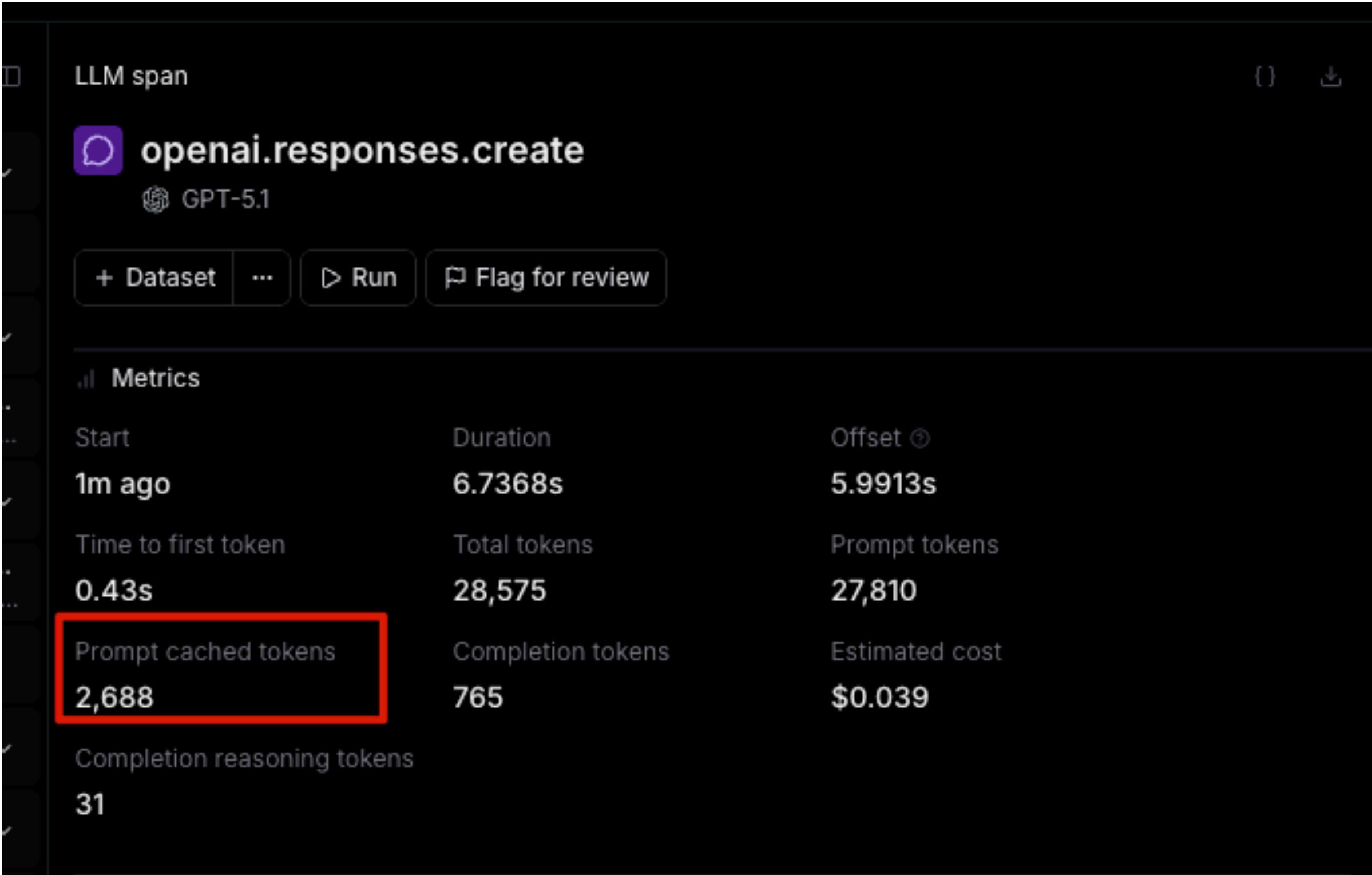
- GPT-5.1
- 首次對話都是 0.019 美金
- 到第十次對話變成
 - 0.022 vs. 0.525
 - 總成本差了 2.4 倍了
- GPU 的 prefill 速度假設是 1W tokens
 - 速度就差 2.7s
 - 而且講越多次就越來越慢



	輸入tokens	快取 tokens	輸入 tokens(不含快取)	輸出 tokens	有快取成本	無快取成本
1	3000	0	3000	1500	0.01875	0.01875
2	6000	3000	3000	1500	0.019125	0.0225
3	9000	6000	3000	1500	0.0195	0.02625
4	12000	9000	3000	1500	0.019875	0.03
5	15000	12000	3000	1500	0.02025	0.03375
6	18000	15000	3000	1500	0.020625	0.0375
7	21000	18000	3000	1500	0.021	0.04125
8	24000	21000	3000	1500	0.021375	0.045
9	27000	24000	3000	1500	0.02175	0.04875
10	30000	27000	3000	1500	0.022125	0.0525

Cache 觀測性

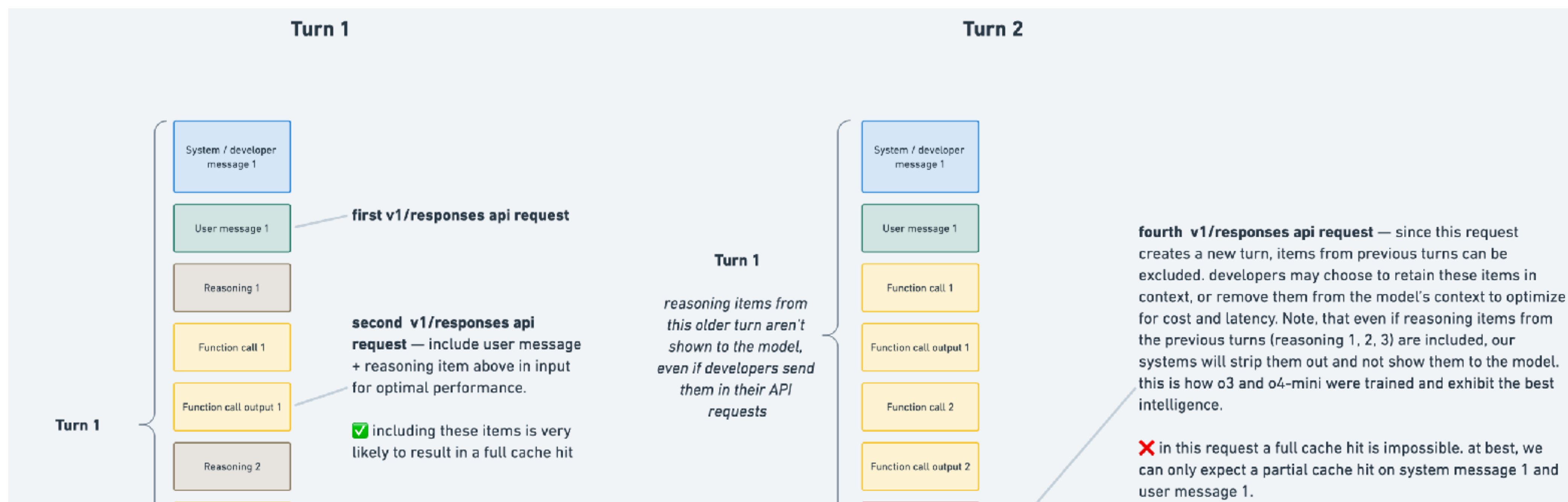
- 你需要觀察紀錄和確認 cache 有無命中



推理模型的 prompt caching 情況

為了節省 context window，下一輪實際上不會 100% match，要再下一輪才會

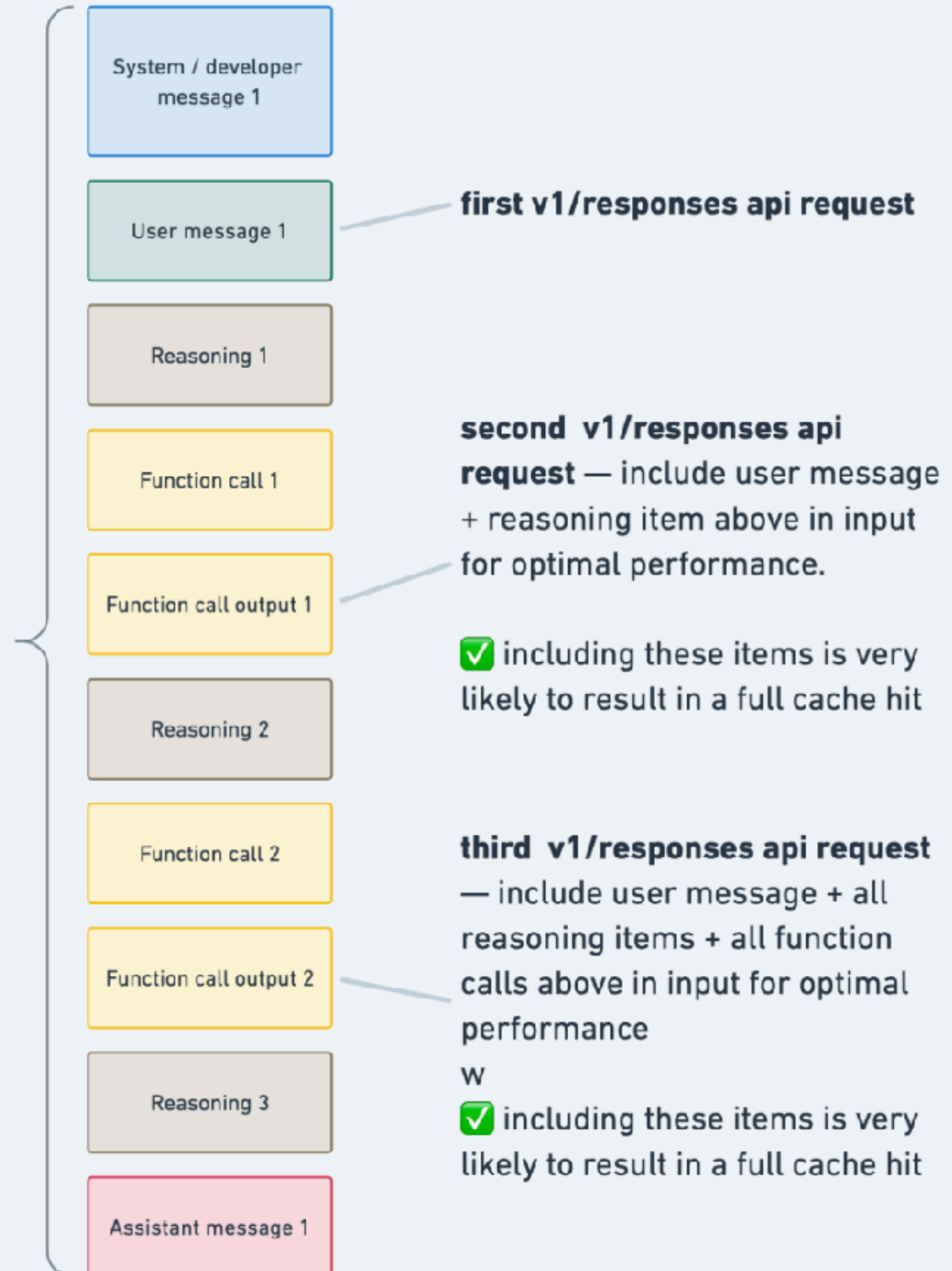
- 目前各家都限制一定要回傳同一輪 turn 的 reasoning tokens!!
- 之前的 reasoning tokens 不用傳，傳了也沒用 OpenAI 會丟棄
- https://cookbook.openai.com/examples/responses_api/reasoning_items



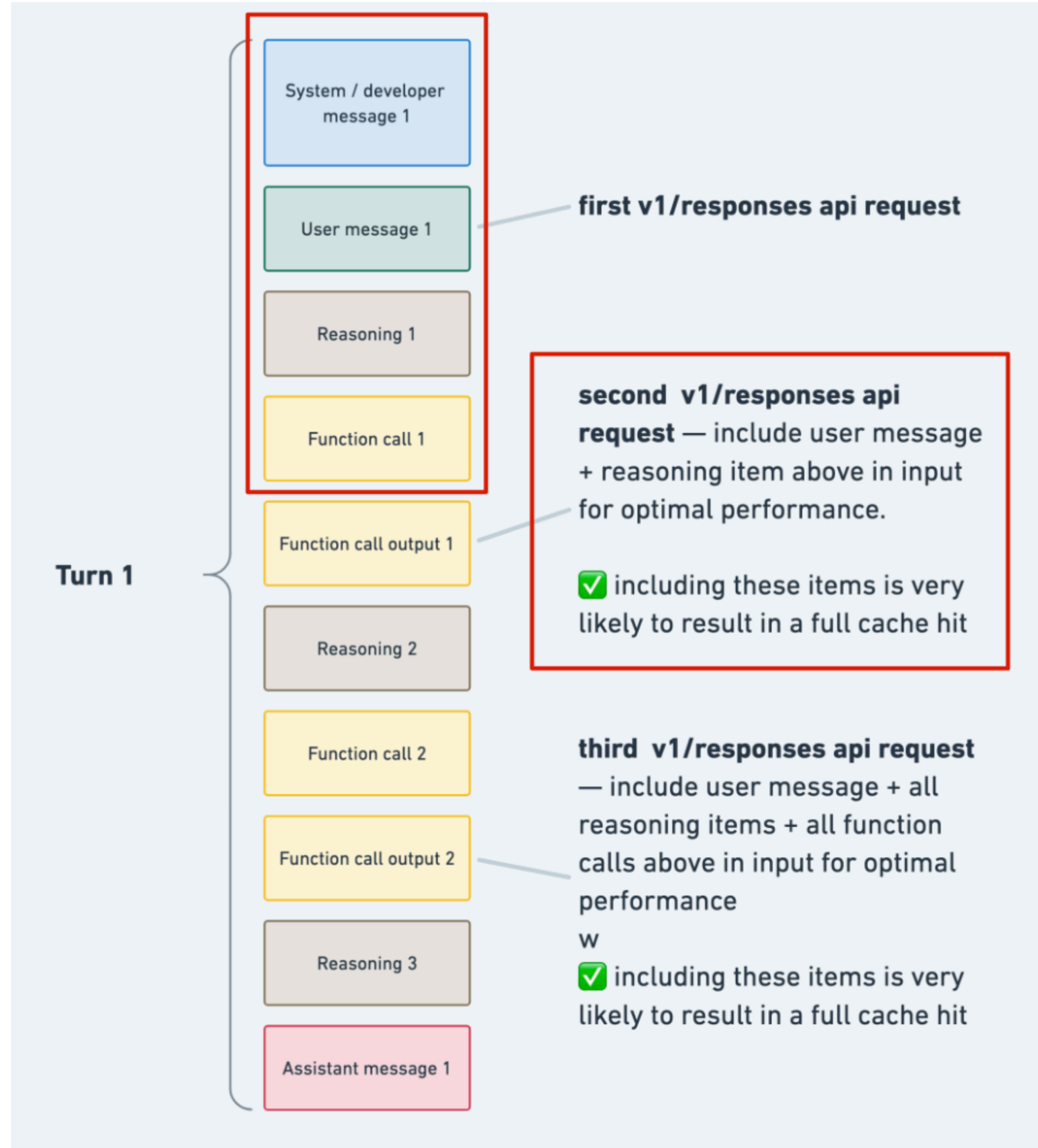
- 這是 用戶對話 Turn 1 第一輪
- 裡面有兩次 Function Calling
- 出現 3 次 API 呼叫

Turn 1

Turn 1



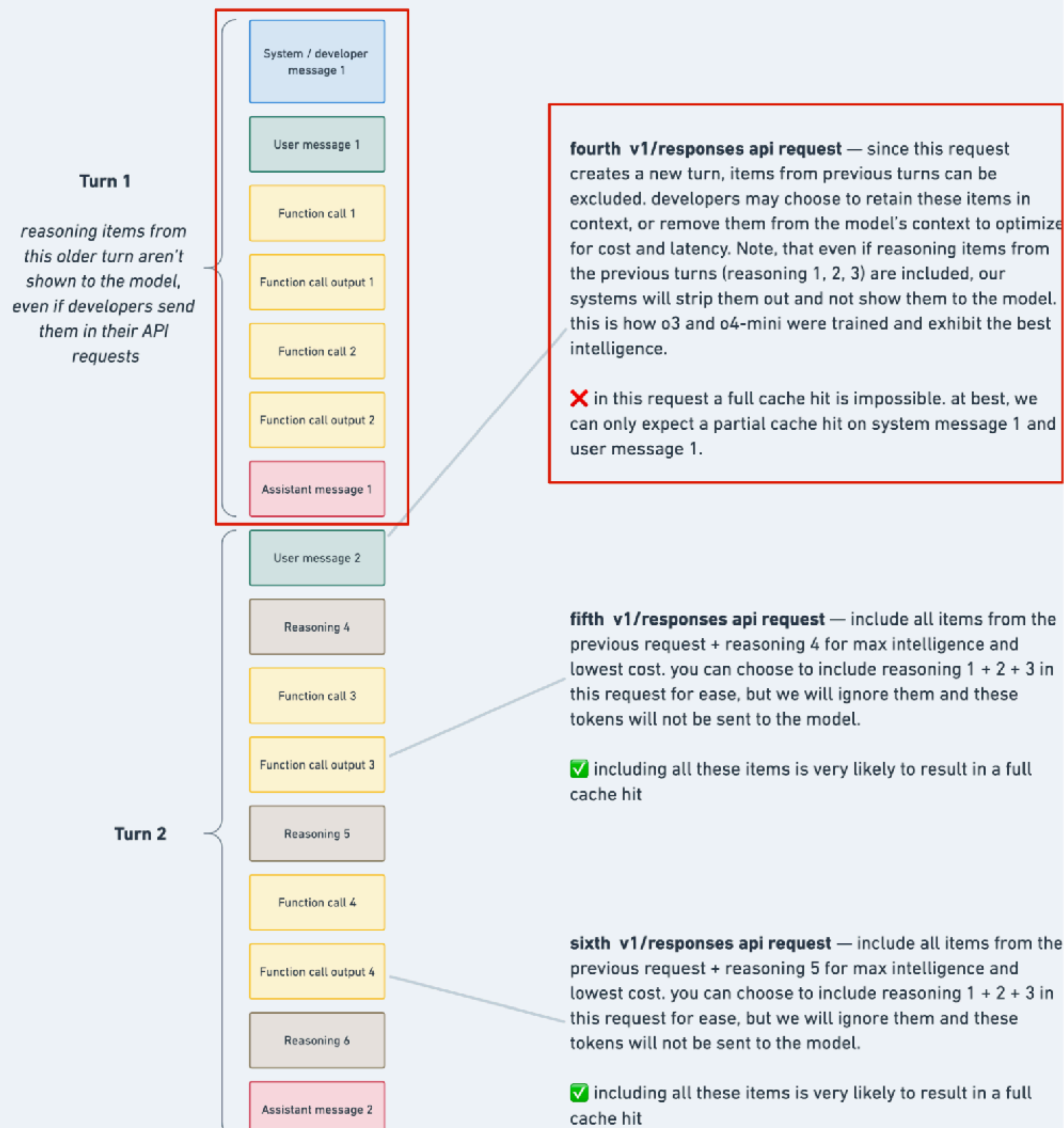
- 第二次 API call
- Cache hit!



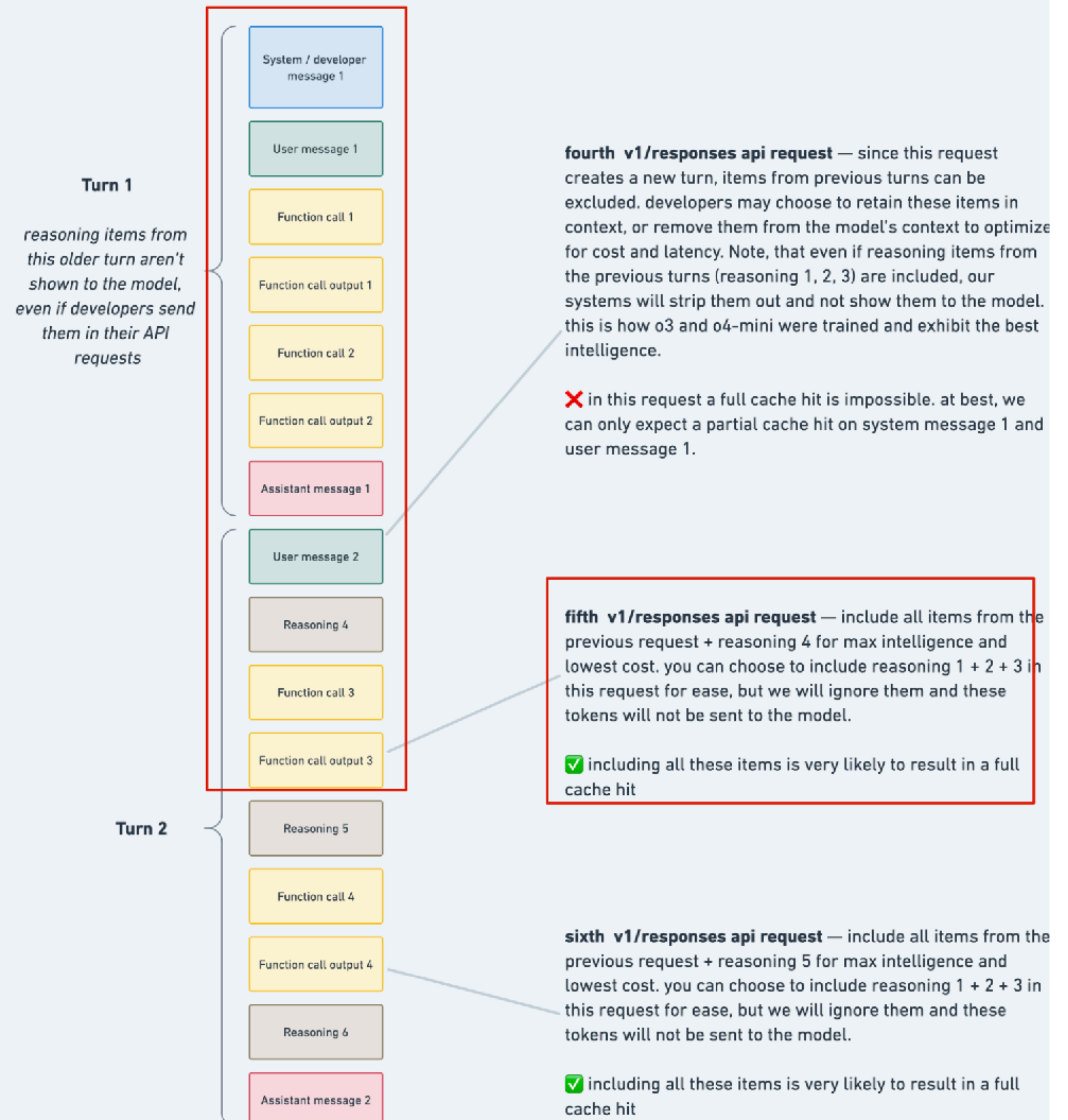
- 第三次 API call
- Cache hit!



- 這是 用戶對話 Turn 2 第二輪
- 裡面有兩次 Function Calling
- 又有 3 次 API 呼叫
- 由於丟棄了上一輪的 reasoning items
- 導致這第 4 次 API 呼叫這次不會完全命中快取，只會有部分命中



- 第 5 次 API call
 - Cache hit!
- 第 6 次 API call
 - Cache hit!



命中 Prompt Cache 的 Pro Tips:

- 1. 讓前綴 prefix 保持穩定
- 2. 保持上下文僅追加 append-only
- 3. JSON serialize 序列化順序
- 4. 不要動態更改工具呼叫定義
- 5. 熟悉各家 API 的不同設定

1. 讓前綴保持穩定: System Prompt 不要變、變得都是最後的 User message

[

{ "role": "system", "content": You are a helpful assistant. Time is {current_time}" },

{ "role": "user", "content": "... " },

{ "role": "assistant", "content": "... " },

{ "role": "user", "content": "... " }

]

慘!! 每輪對話都改了 system prompt
導致快取失效

讓前綴保持穩定: System Prompt 不要變、變得都是最後的 User message

[

{ "role": "You are a helpful assistant.",

{ "role": "user", "content": "...", },

{ "role": "assistant", "content": "...", },

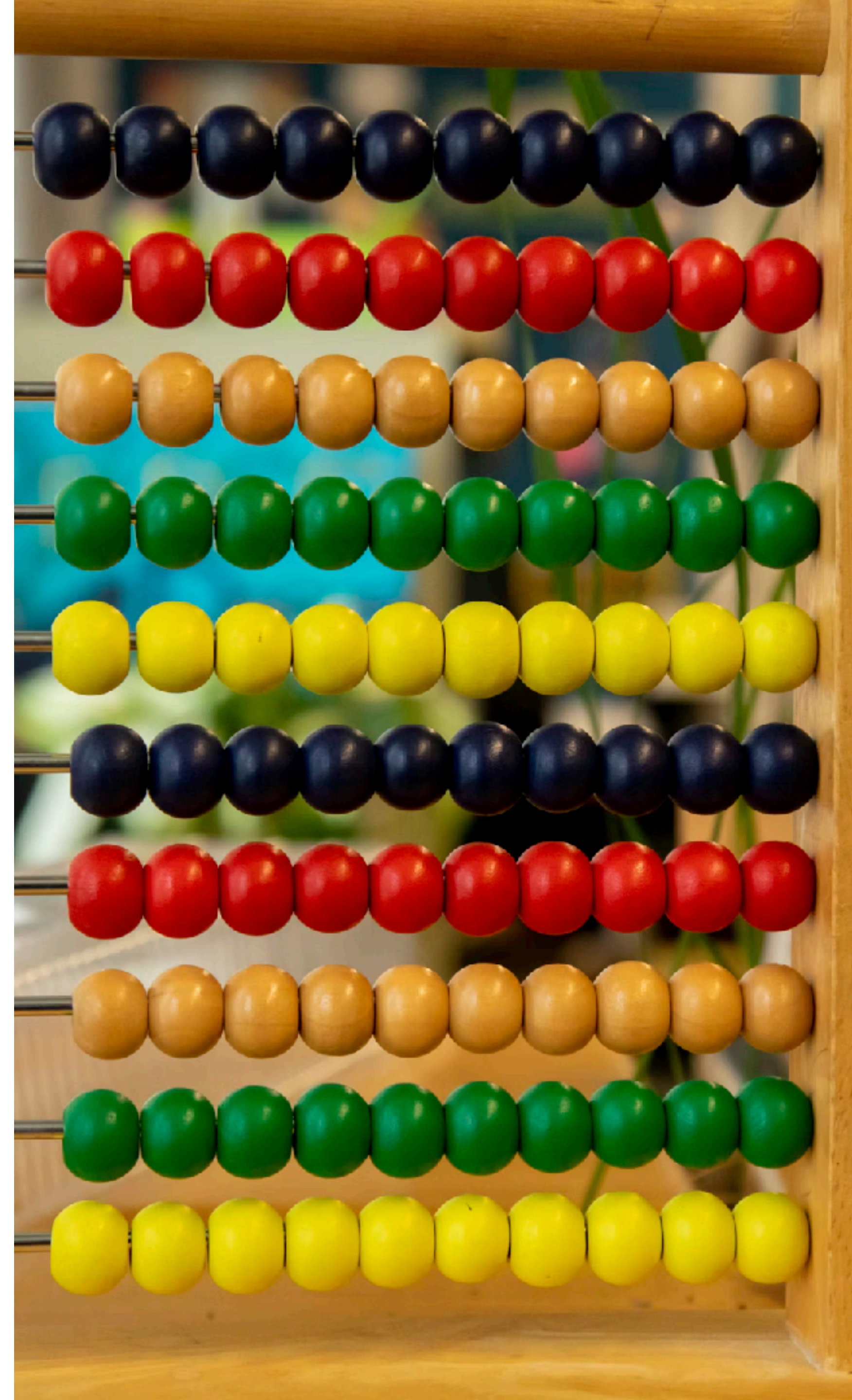
{ "role": "user", "content": ""<current_time>{current_time}</current_time>
<user_query>{user_query}</user_query>""" }

system prompt 盡量不要動
一改這輪快取就重來

]

2. 保持上下文僅追加 append-only

- 在流程之間，(盡量)都是 append items
 - 一樣的 msg items 累加
- 而不是拿前一個 output，重組新的 prompt
 - 換全新的 system msg 跟 user msg
- 大多數的 Prompt chaining 流程教學範例，在 cache first 原則下都是不好的



Prompt chaining 舉例(全新 prompt，不會命中快取)

1 API call

請直譯以下的英文內容
到繁體中文翻譯
{text1}

將輸出 text2 放到
第二個 API call

2 API call

你是潤飾高手，以下是原文
{text1}
以下是中文翻譯
{text2}
請進行潤飾，重新給我一份更好
的中文翻譯

Append-Only 舉例(同一份 chat history 沿用 append，命中快取)

```
[  
  {"role": "system", "content": "請  
直譯以下的英文內容，到繁體中文  
翻譯" },  
  {"role": "user", "content": text1},  
]
```



```
[  
  {"role": "system", "content": "請直譯以下的英文  
內容，到繁體中文翻譯" },  
  {"role": "user", "content": text1},  
  {"role": "assistant", "content": text2},  
  {"role": "user", "content": "你是潤飾高手，請參考  
中文翻譯，重新潤飾給我一份更好的" },  
]
```

~~Prompt Chaining~~ Append-Only Log

3. 保持 JSON serialize key-value 順序穩定

- 例如傳入 function list 的定義，底層會轉成 JSON 送出
- 有些些程式語言(例如 Swift、Go?) 在進行 JSON 轉換時會隨機排列 key-value 順序，這可能會導致快取失效



4. 小心動態工具定義功能

- 工具太多，你想要動態根據當時情況才放入 Agent
- 你一改工具定義，prompt cache 就重來
- OpenAI 有提供 `allowed_tools` 功能: <https://platform.openai.com/docs/guides/latest-model> 解決這個問題
 - tools 定義不變
 - 用另一個參數去控制
- 如何避免 Tools 太多問題
 - 考慮逐步揭露工具的設計，請參考 Anthropic Skills 設計
 - <https://www.facebook.com/ihower/posts/10163047278398971>

Pydantic AI

Toolsets

A toolset represents a collection of **tools** that can be registered with an agent in one go. They can be reused by different agents, swapped out at runtime or during testing, and composed in order to dynamically filter which tools are available, modify tool definitions, or change tool execution behavior. A toolset can contain locally defined functions, depend on an external service to provide them, or implement custom logic to list available tools and handle them being called.

Toolsets are used (among many other things) to define **MCP servers** available to an agent. Pydantic AI includes many kinds of toolsets which are described below, and you can define a **custom toolset** by inheriting from the `AbstractToolset` class.

Conditional tool enabling

OpenAI Agents SDK 類似功能

You can conditionally enable or disable agent tools at runtime using the `is_enabled` parameter. This allows you to dynamically filter which tools are available to the LLM based on context, user preferences, or runtime conditions.

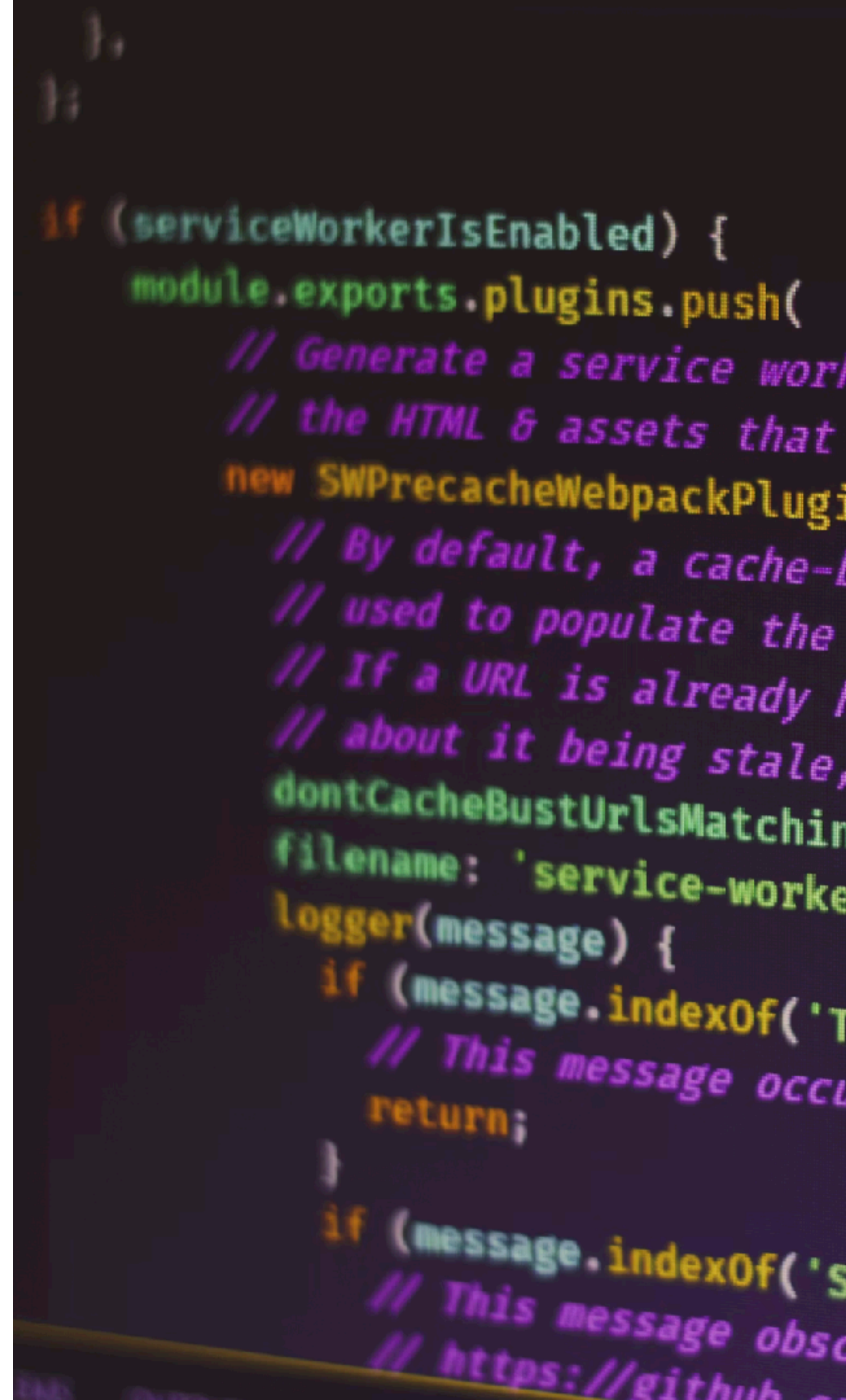
```
import asyncio
from agents import Agent, AgentBase, Runner, RunContextWrapper
from pydantic import BaseModel

class LanguageContext(BaseModel):
    language_preference: str = "french_spanish"

def french_enabled(ctx: RunContextWrapper[LanguageContext], agent: AgentBase) -> bool:
    """Enable French for French+Spanish preference."""
    return ctx.context.language_preference == "french_spanish"
```

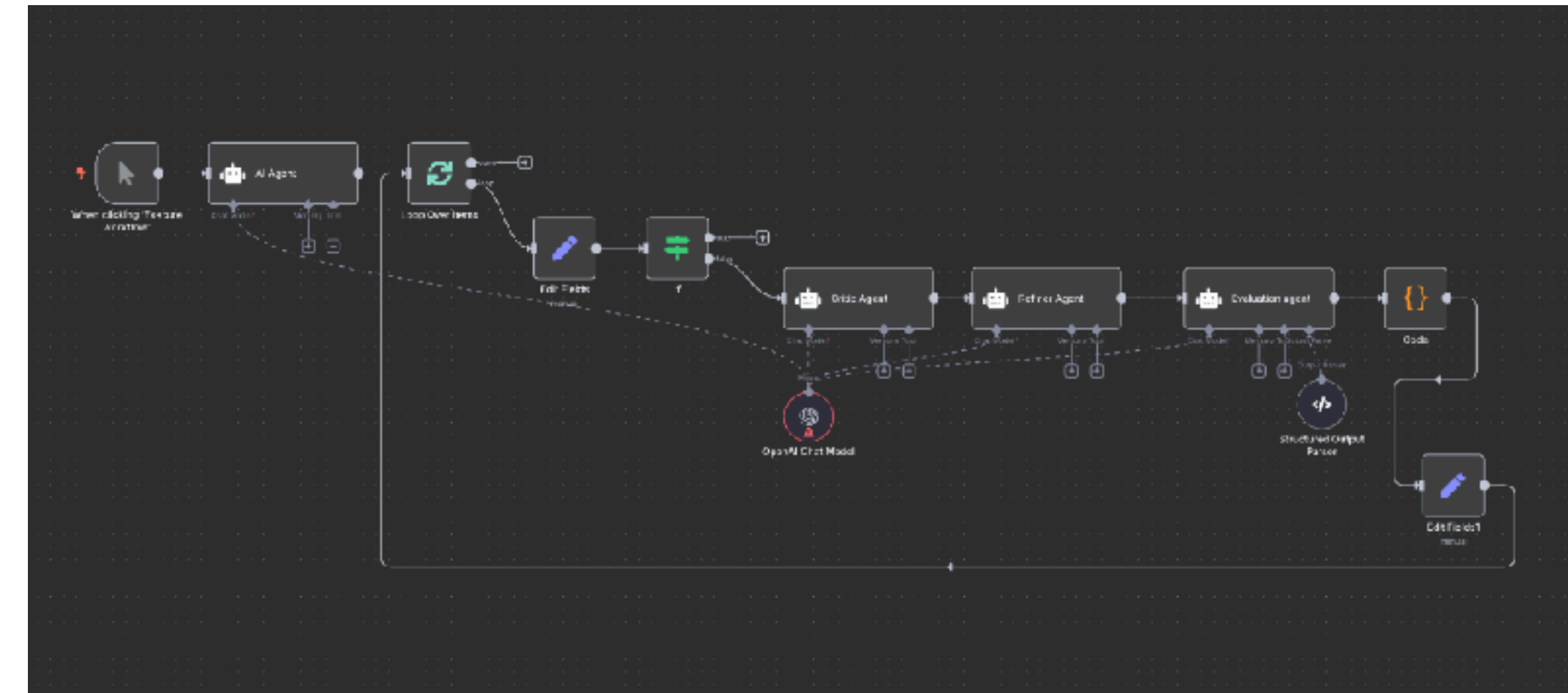

5. 熟悉各家 Prompt Cache 設定

- OpenAI
 - <https://platform.openai.com/docs/guides/prompt-caching>
 - prompt_cache_key 確保導向同一個後端快取 server
 - 在 2025/11 gpt-5.1 增加了 prompt_cache_retention 可以延長快取到 24h
 - Implicit caching 全自動
- Claude
 - cache_control 可以增加 TTL，預設是 5 分鐘，可以增加到 1 hr 但要額外付費
 - <https://platform.claude.com/docs/en/build-with-claude/prompt-caching>
 - Explicit caching，需要自己設定 cache 檢查點，至多四個
- Gemini
 - <https://ai.google.dev/gemini-api/docs/caching?lang=python>
 - 有 Implicit caching 和 Explicit caching



6. 遠離 視覺化工作流程工具

- Dify, n8n, LangFlow 等等
- 視覺化工作流程工具不適合開發 對話型 Agent 應用
 - 複雜的任務在視覺化建構器中很快就會變得難以管理
 - 不容易把 Prompt Caching 做對
- 有很多 TTFT 和 Prompt Cache 的 Anti-Patterns
 - 設計思維是節點 output → input 的 chaining，而非 append-only
 - 容易把 Agent 可以解決的開放性問題，過度拆成工作流程，TTFT 和 latency 超慢
 - 平台本身就慢，自帶 latency
 - 有些平台不支援 平行處理，無法最佳化
 - 有些平台不支援(真)結構化輸出，還得加 LLM call 修 JSON，又更慢了
- 批次 Eval 評估困難



你會發現....

- 你越不懂底層框架的運作，轉接越多抽象層
- 你就越容易 missing prompt cache，而且不知道為何
- 例如有些框架會動態 inject prompt 在歷史訊息裡面!!
- 你需要明確知道送給 LLM API 的 items 順序



Prompt Cache 文章推薦

- <https://sankalp.bearblog.dev/how-prompt-caching-works/>
- <https://axk51013.medium.com/llm-%E5%B0%88%E6%AC%84-%E9%80%B2%E9%9A%8E-context-engineering-kv-cache-centric-llm-%E6%87%89%E7%94%A8%E8%A8%AD%E8%A8%88-ee70eb331983>

sankalp's blog

Home Featured Blog About

How prompt caching works - Paged Attention and Automatic Prefix Caching plus practical tips

01 Dec, 2025

Table of Contents

【LLM 專欄】進階 context engineering : KV cache centric LLM 應用設計

如何藉由 prompt design 來優化系統效能、降低成本並提升用戶體驗



健愷 Oscar

Following

43 min read · Aug 4, 2025



26



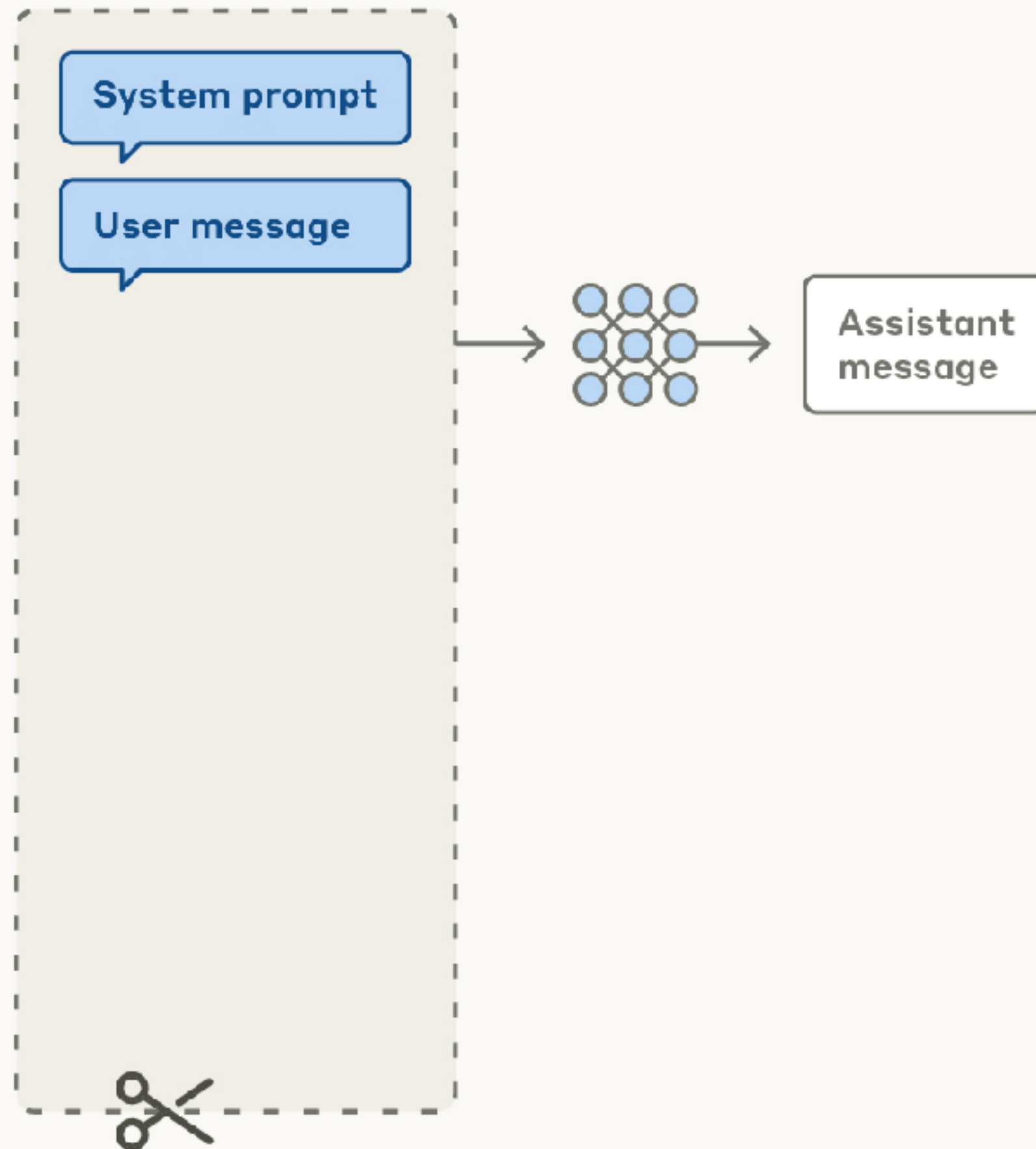
【LLM專欄】進階 context engineering

Context Engineering

Prompt engineering vs. context engineering

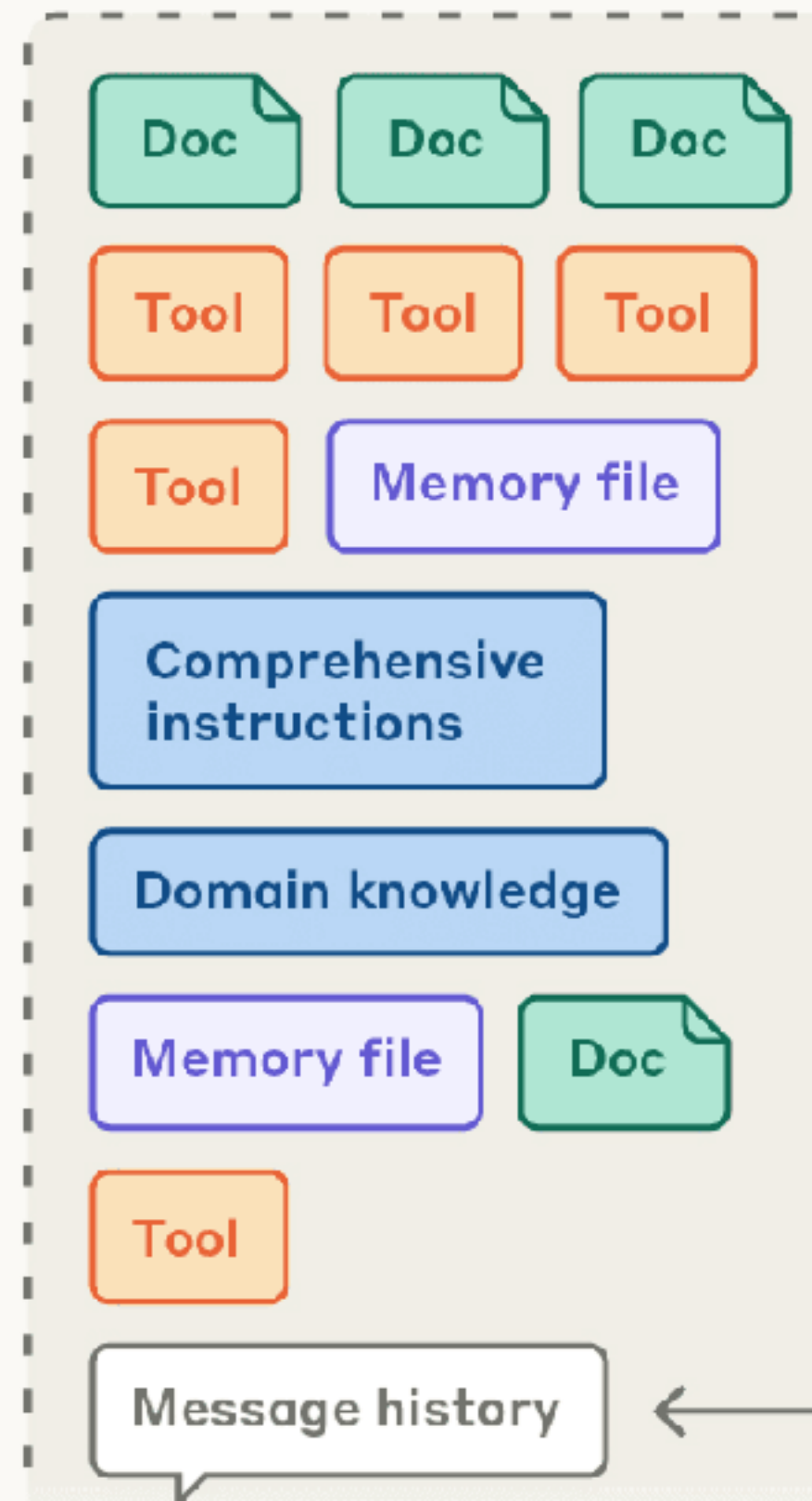
Prompt engineering for single turn queries

Context window

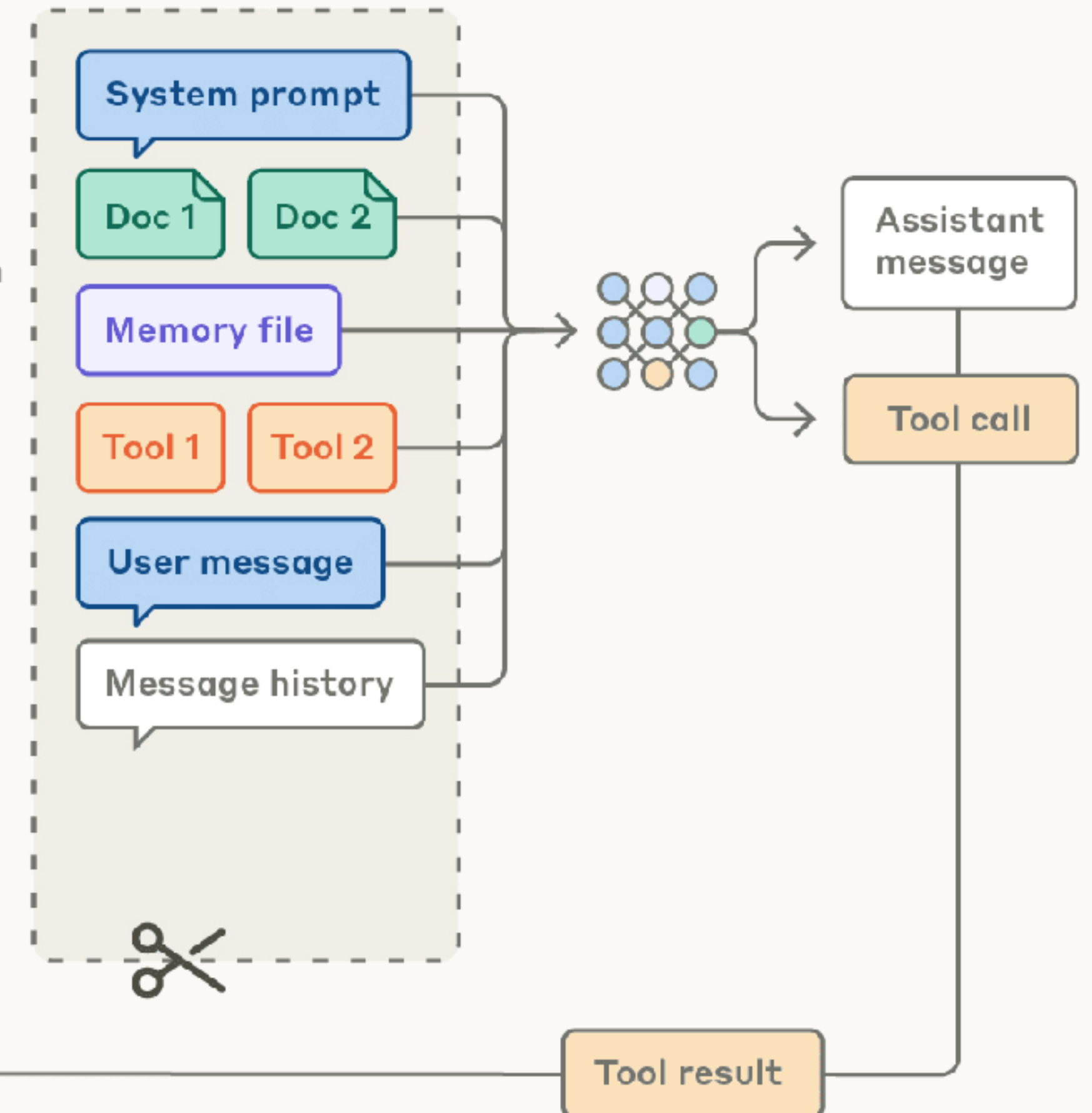


Context engineering for agents

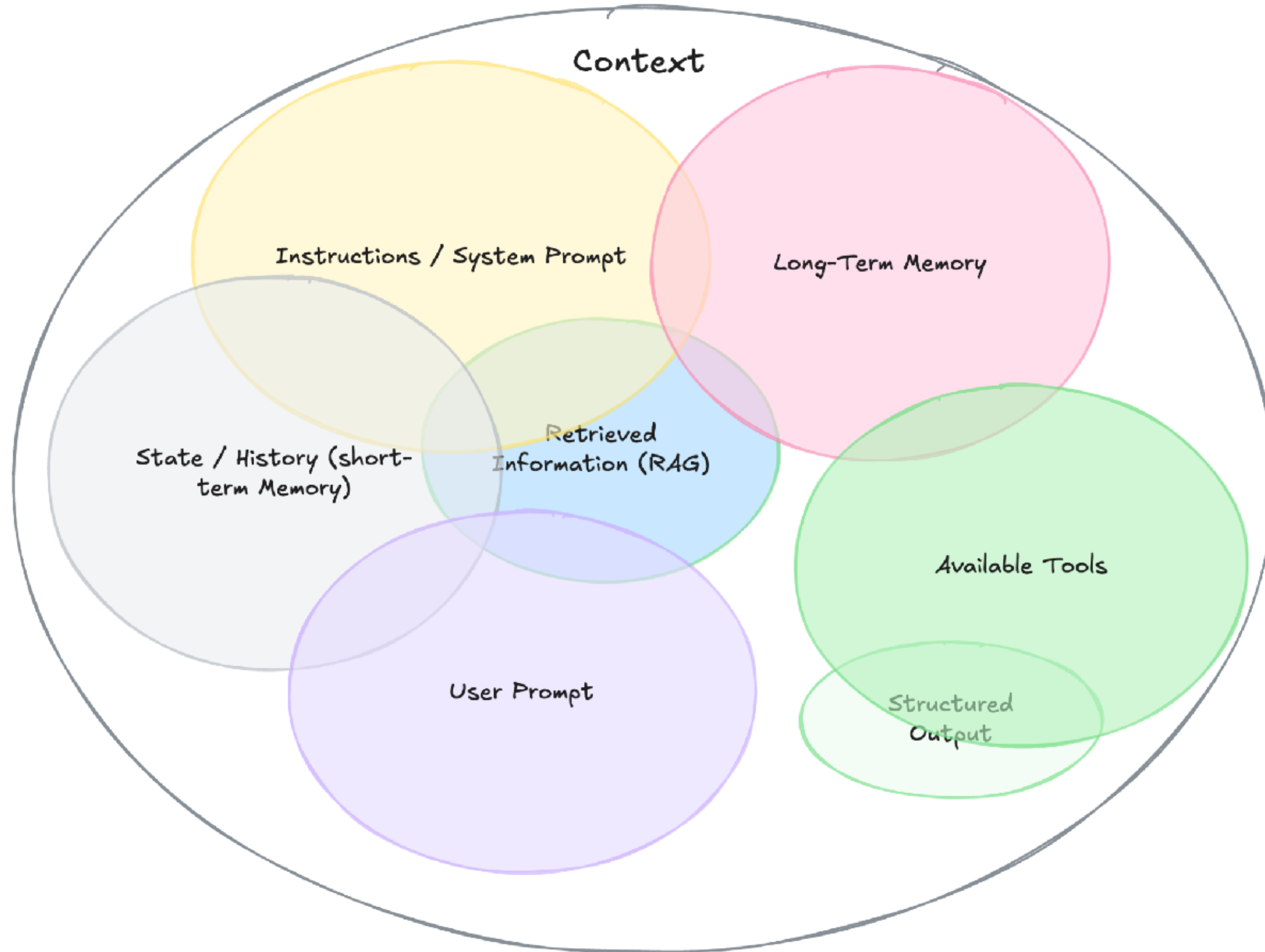
Possible context to give model



Context window



Context Engineering



Context 包含什麼？

<https://ihower.tw/blog/12817-context-engineering>

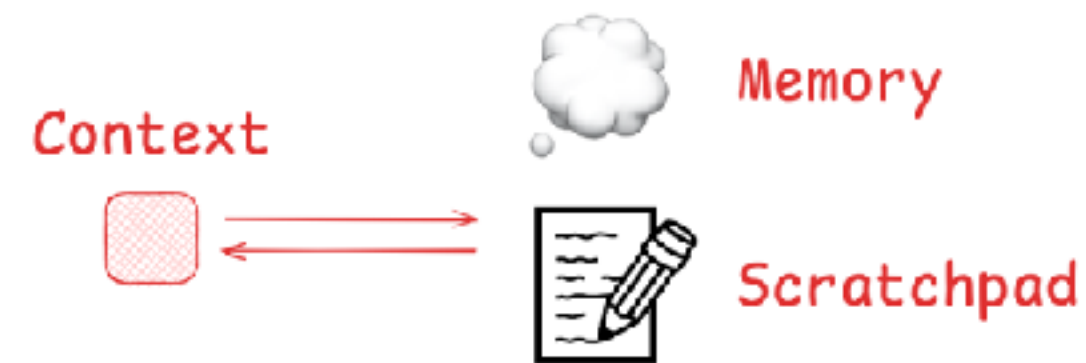
- Instructions / System Prompt: 定義模型行為的初始指令和風格
- User Prompt / User Input: 使用者的請求、問題或輸入
- State / History: 當前對話的聊天歷史，包含所有使用者和模型的回應，這又叫做「短期記憶」
- Long-Term Memory: 跨對話的持久知識庫，包含學習到的使用者偏好、過去對話摘要、被告知要記住的事實，這又叫做「長期記憶」
- Retrieved Information (RAG): 外部的最新知識，從文件、資料庫或 API 檢索的相關資訊
- Available Tools: 所有可呼叫的函數或工具定義
- Responses from tools: 工具執行的結果
- Structured Output: 定義模型回應格式，例如 JSON 或 XML

Context 處理技巧

https://github.com/langchain-ai/how_to_fix_your_context

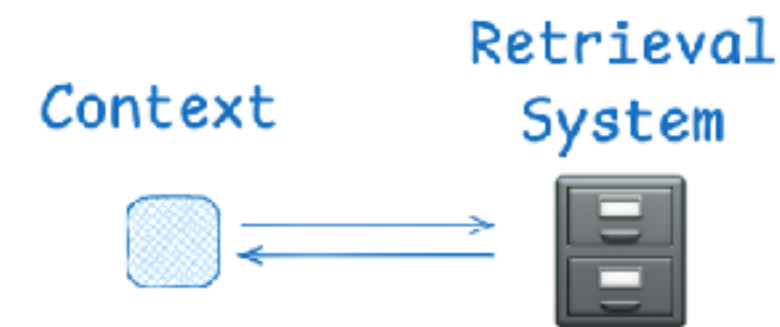
Context Offloading

Context Offloading is the act of storing information outside the LLM's context, usually via a tool that stores and manages the data.



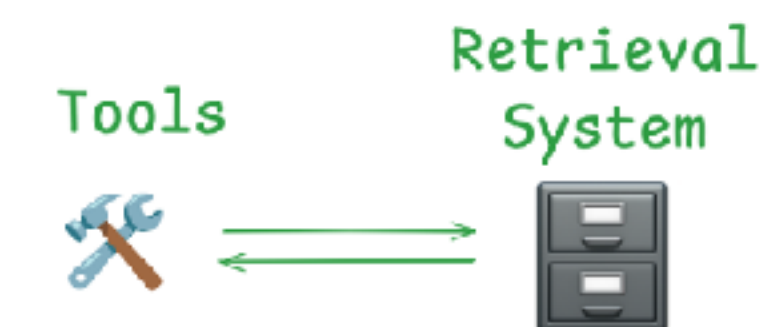
RAG

Retrieval-Augmented Generation (RAG) is the act of selectively adding relevant information to help the LLM generate a better response.



Tool Loadout

Tool Loadout is the act of selecting only relevant tool definitions to add to your context.



Context Pruning

Context Pruning is the act of removing irrelevant or otherwise unneeded information from the context.



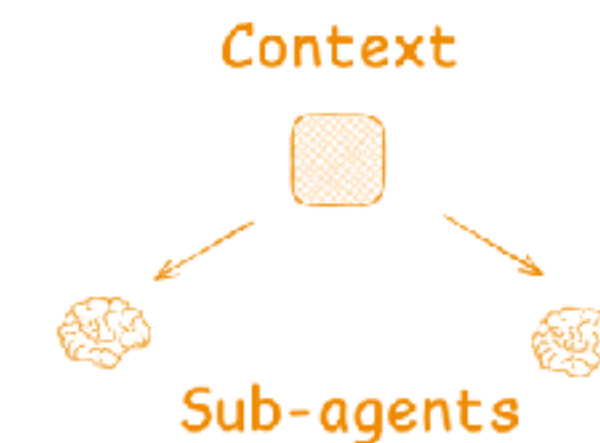
Context Summarization

Context Summarization is the act of boiling down an accrued context into a condensed summary.



Context Quarantine

Context Quarantine is the act of isolating contexts in their own dedicated threads, each used separately by one or more LLMs.



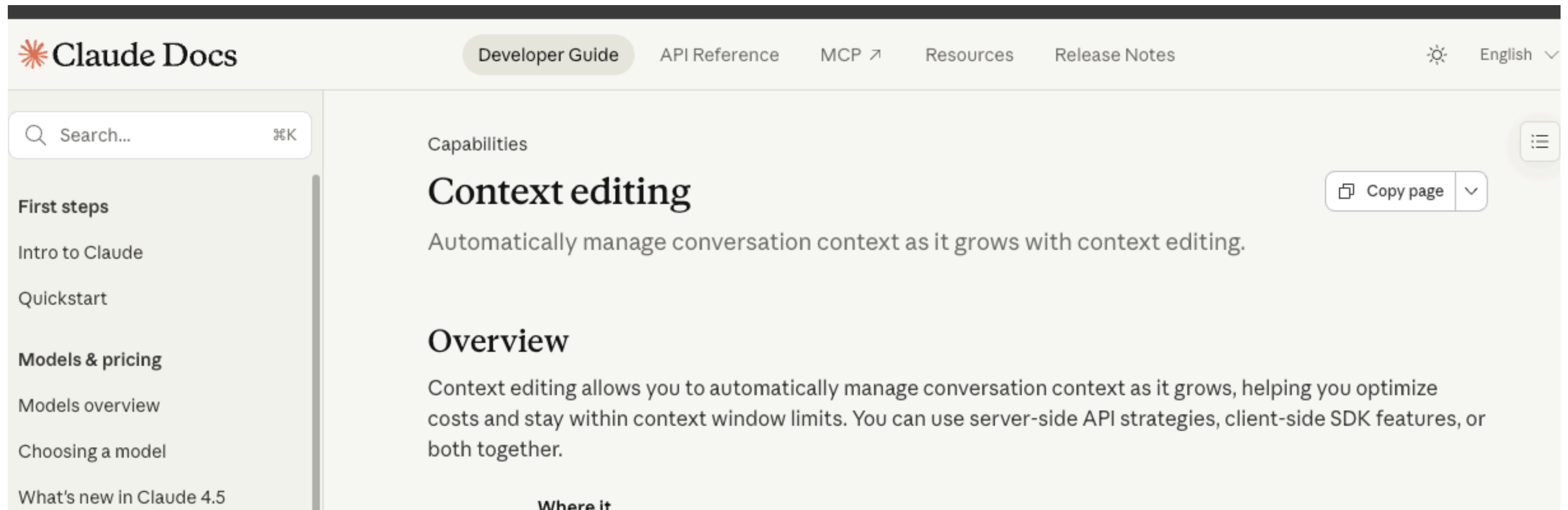
第一考量點: Context Window 限制

- GPT-4o 128k
- GPT-4.1 1048k
- GPT-5 400k
- 雖然模型可以處理長 Context，但是各種研究顯示:
 - 模型處理 Long Context 的能力並不均勻，隨著輸入長度增加，效能會以各種意想不到的方式下降
 - <https://fiction.live/stories/Fiction-liveBench-Mar-25-2025/oQdzQvKHw8JyXbN87>
 - <https://research.trychroma.com/context-rot>
 - <https://www.dbreunig.com/2025/06/22/how-contexts-fail-and-how-to-fix-them.html>

優先聚焦在 Context Editing 需求

<https://platform.claude.com/docs/en/build-with-claude/context-editing>

- Claude 提供常見的 Context Editing 功能
- 蠻喜歡 Claude 文件中把這些招式統稱叫做 Context Editing 技巧，更為聚焦



Claude 提供的 Context Editing 功能參考

- Tool Result Clearing 清理舊的 tool result
 - trigger：何時啟動清理 (input_tokens / tool_uses)
 - keep：保留多少最新 tool uses (例如 keep: {type:"tool_uses", value:3})
 - clear_at_least：每次至少清掉多少 tokens. 避免白白讓 cache 失效
 - clear_tool_inputs：是否連同 tool call 參數一起清除
- Thinking block clearing 清除舊 thinking block
 - keep: 保留最後 N 個 thinking turns
- Compaction 產生 summary 來壓縮整段對話歷史
 - context_token_threshold：壓縮門檻
 - model：使用不同模型產生摘要 (例如 haiku 節省成本)
 - summary_prompt：自訂摘要格式

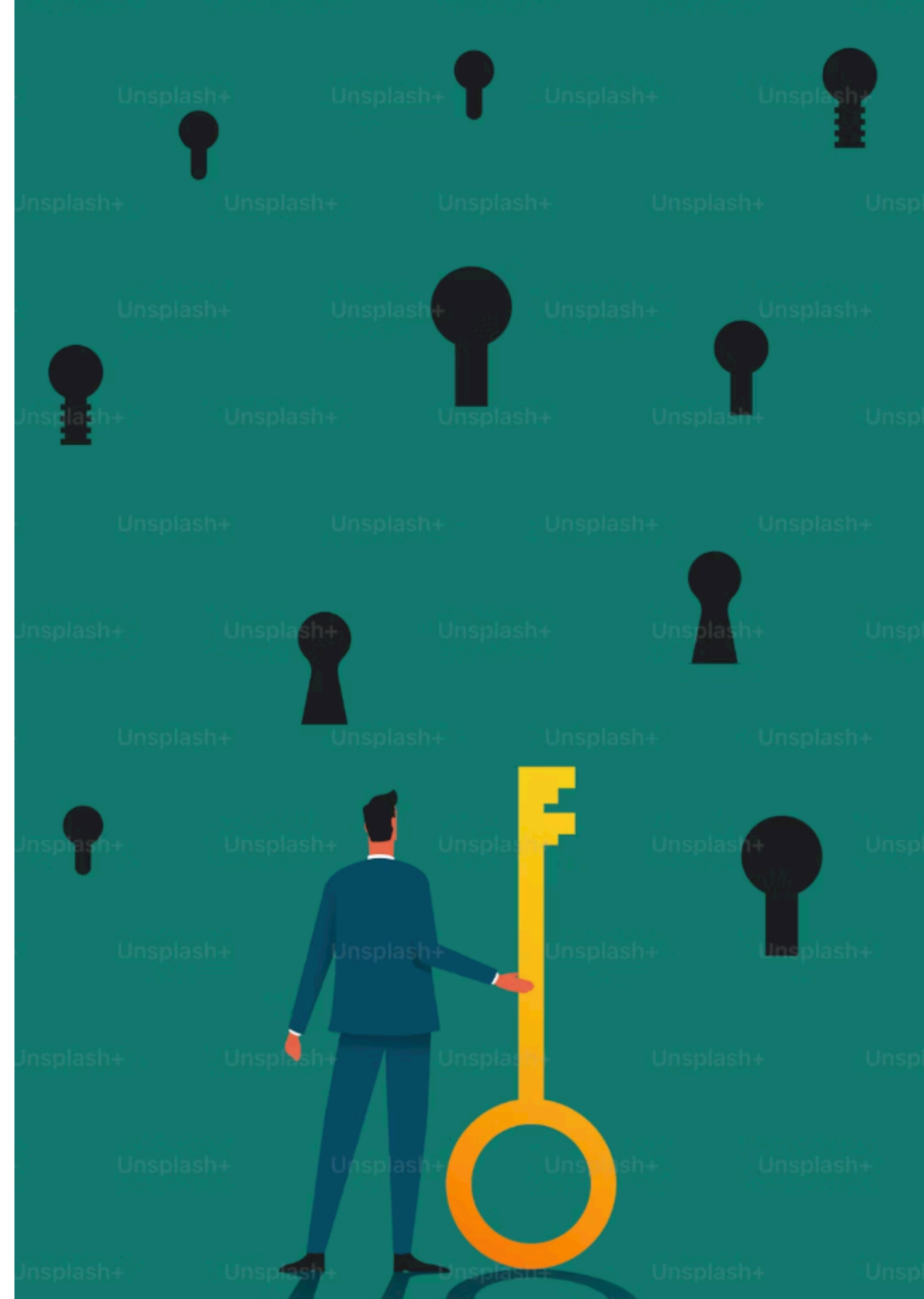
Compaction 壓縮裁剪

刪除 or 摘要將過往對話

- 重點不是「保留所有資訊」，而是「保留學習軌跡」，好的摘要應該要：
 - 保留因果關係: 不只記錄發生了什麼，更要記錄為什麼會這樣
 - 保留失敗經驗: 知道什麼不可行跟知道什麼可行一樣重要
 - 保留決策脈絡: 為什麼選擇這個方法而非那個
 - 舉例: 差的摘要「修復了認證問題」好的摘要「嘗試更新 token 失敗(過期)，改用 refresh token 成功，學到要先檢查 token 有效期」，這能讓 Agent 在後續任務中避免重複錯誤，維持正確的「學習動量」
 - <https://jxnl.co/writing/2025/08/30/context-engineering-compaction/>
- **Pro Tip:** 最安全且影響最小的壓縮方式之一就是「清除工具結果」

第二考量點: Prompt Caching

- **✗ 修改歷史 context 會破壞快取!!!**
- 設計 Tradeoff
 - 要多常進行 Context Editing ?
 - 為了避免太常觸發讓快取失效，最好快用光 context 時，再一次壓縮裁剪



更多 Context Engineering: Tool Design

<https://www.anthropic.com/engineering/writing-tools-for-agents>

- 少即是多: 不是工具越多越好。與其包裝現成 API endpoint 成為工具，不如設計幾個精心整合的高階工具。例如不要分開做 `list_users`、`list_events`、`create_event`，直接做一個 `schedule_event` 搞定所有事
- 命名空間很重要: 當工具變多時，清楚的命名規則能幫 Agent 選對工具。像 `asana_search` vs `jira_search` 這樣的前綴命名，效果比你想像的大
- 返回有意義的資訊: Agent 處理自然語言比處理低階的技術 id 更在行，使用這種人類也能看懂的格式，能大幅減少幻覺
- **Token 效率最佳化:** 實作分頁、過濾、截斷等機制，給工具加上 `response_format` 參數，讓 Agent 選擇要“**concise**”還是“**detailed**”的回應
- 錯誤訊息要有幫助: 與其丟出“Error 404”，不如告訴 Agent: “找不到該用戶，請檢查拼寫或使用 `search_user` 工具先搜尋”
- 工具描述: 這個也需要用 Prompt Engineering 進行最佳化

Tool Response Design

- Context Offloading 策略: 檢索還是回傳 snippet，讓模型自己判斷是否 load full page，也就是拆成兩個工具例如 search 和 load_pages
 - <https://columns.chicken-house.net/2025/09/16/chat-with-my-blog/>
- 工具輸出除了 chunk 還可以包括 metadata，可提供多種 面向(facets) 引導 Agent 探索多元路徑

讓 Sub-Agent 處理子任務，維持 Lead Agent 乾淨的 Context

- 問題: 單一 AI agent 處理複雜任務時，上下文被大量垃圾資訊污染
 - 例如: Coding Agent 執行測試時，150,000 字的日誌淹沒 5,000 字的功能規劃
 - 造成 context 有 95% 噪音，嚴重影響效能
- 職責分離的架構: 將耗費大量 context 的工作隔離在 sub-agent 內部，做完最後回傳結果給 Lead Agent 就好
 - 子代理人進行大量探索 (使用數萬個 tokens)
 - 只回傳濃縮摘要給主代理人 (1000-2,000 tokens)
 - 主代理人專注於綜合與分析結果
- 優勢
 - 避免單一 Agent 的上下文污染問題
 - 特別適合: 複雜研究與分析任務，尤其可以平行執行 Sub-Agents 時

我的 Context Engineering demo code

- OpenAI Agents SDK + FastAPI server 範例
 - <https://github.com/ihowe/ihowe-llm-workshop/blob/main/main.py>
 - get_agent_stream_v2
 - Context Engineering: 去除工具輸出、砍最舊留言

Recap

- OpenAI Agents SDK、FastAPI
- 前後端整合發展: AG-UI 和 MCP-UI
- 用戶對話類型的 Agent
 - TTFT 很重要: 8 種改進法
 - 串流 UX、平行化
 - Prompt Caching 很重要: 6 個 Pro Tips
 - Append-Only log 思維

EVERY SECOND COUNTS

我的開源上手專案: OpenAI Agents SDK + FastAPI + Braintrust Playbook

- <https://github.com/ihowe/openai-agents-fastapi-playbook>

The screenshot shows the GitHub repository page for 'openai-agents-fastapi-playbook' by user 'ihowe'. The repository is public and has 2 stars. The main branch is 'main'. The repository description is 'OpenAI Agents SDK + FastAPI + Braintrust Playbook'. The repository contains three folders: 'app', 'data', and 'dspy_program'. The 'app' folder was last modified 'last month' and contains a commit 'Refactor main.py to extract API routes into separate co...'. The 'data' folder was last modified '2 months ago' and contains a commit 'Add Docker support and reorganize data directory stru...'. The 'dspy_program' folder was last modified '2 months ago' and contains a commit 'Add guardrail evaluation framework with DSPy optimiz...'. The repository has 31 commits.

ihower / openai-agents-fastapi-playbook

Type / to search

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

openai-agents-fastapi-playbook Public

Unpin Open in GitIngest Watch 0 Fork 0 Star 2

main 1 Branch 0 Tags

Go to file t + <> Code

About

OpenAI Agents SDK + FastAPI + Braintrust Playbook

aihao.tw/llm

Readme Activity 2 stars

ihower	Refactor main.py to extract API routes into separate controller modules	43bbd4d · last month	31 Commits
app	Refactor main.py to extract API routes into separate co...	last month	
data	Add Docker support and reorganize data directory stru...	2 months ago	
dspy_program	Add guardrail evaluation framework with DSPy optimiz...	2 months ago	

謝謝，請多指教 🙏

- 個人部落格和電子報 <https://ihower.tw>
- 我的 LLM 課程 <https://aihao.tw/llm>
- Facebook: <https://www.facebook.com/ihower>
- Threads: <https://www.threads.net/@ihower>

