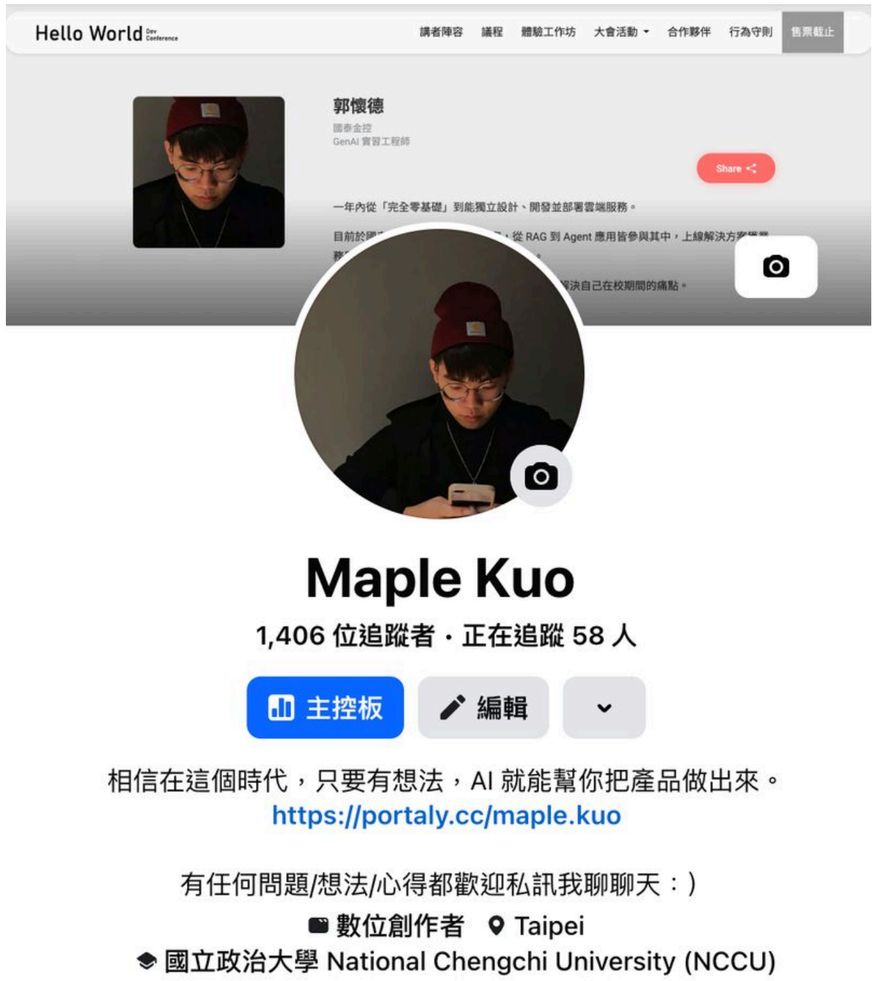
那些跟 Context 管理有關的兩三事



Maple Kuo

2026/08/04

經歷



- 國立政治大學 英國語文學系
- 人工智慧跨域微學程修畢
- 2025 Hello World Dev Conference 講者
- 衛生所「來學 AI 吧！數位轉型的第一步」講者
- Generative AI 小聚 講者
- Claude Taiwan 版主
- Zeabur “Ship It” Hackathon 第三名
- 一年從 0 開始和 AI 自學轉職軟體工程師

企業在意什麼?

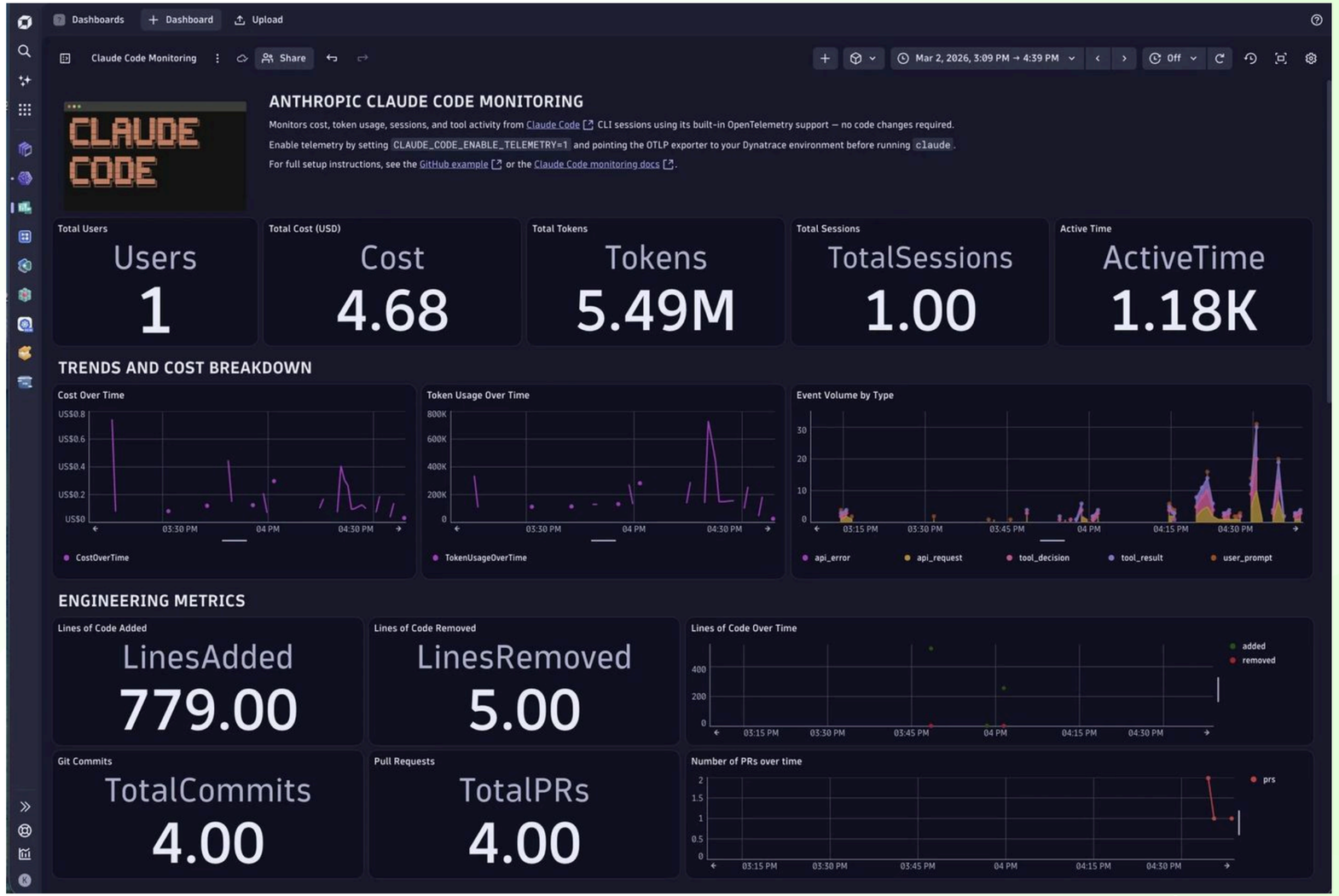


一家企業最在意的是什麼

企業在意什麼?

- 合規與可稽核性（與人有關）
- 持續獲利（與錢有關）

可觀測/可稽核性



Source: Reddit r/ClaudeCode

可觀測/可稽核性

```bash
# 1. 啟用遙測
export CLAUDE_CODE_ENABLE_TELEMETRY=1

# 2. 選擇匯出器（兩者都是可選的 - 僅配置您需要的）
export OTEL_METRICS_EXPORTER=otlp       # 選項：otlp、prometheus、console、none
export OTEL_LOGS_EXPORTER=otlp          # 選項：otlp、console、none

# 3. 配置 OTLP 端點（用於 OTLP 匯出器）
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317

# 4. 設定身份驗證（如果需要）
export OTEL_EXPORTER_OTLP_HEADERS="Authorization=Bearer your-token"

# 5. 用於除錯：減少匯出間隔
export OTEL_METRIC_EXPORT_INTERVAL=10000  # 10 秒（預設：60000ms）
export OTEL_LOGS_EXPORT_INTERVAL=5000     # 5 秒（預設：5000ms）

# 6. 執行 Claude Code
claude
```

Source: Claude Code Docs

企業在意什麼?

- 合規與可稽核性（與人有關）
- **持續獲利（與錢有關）**

企業可能的解方



怎麼透過 Context 管理來盡可能的降低成本

先理解 Context 組成



Source: “Using Claude Code: session management and 1M context”

先理解 Context 組成



Context Engineering

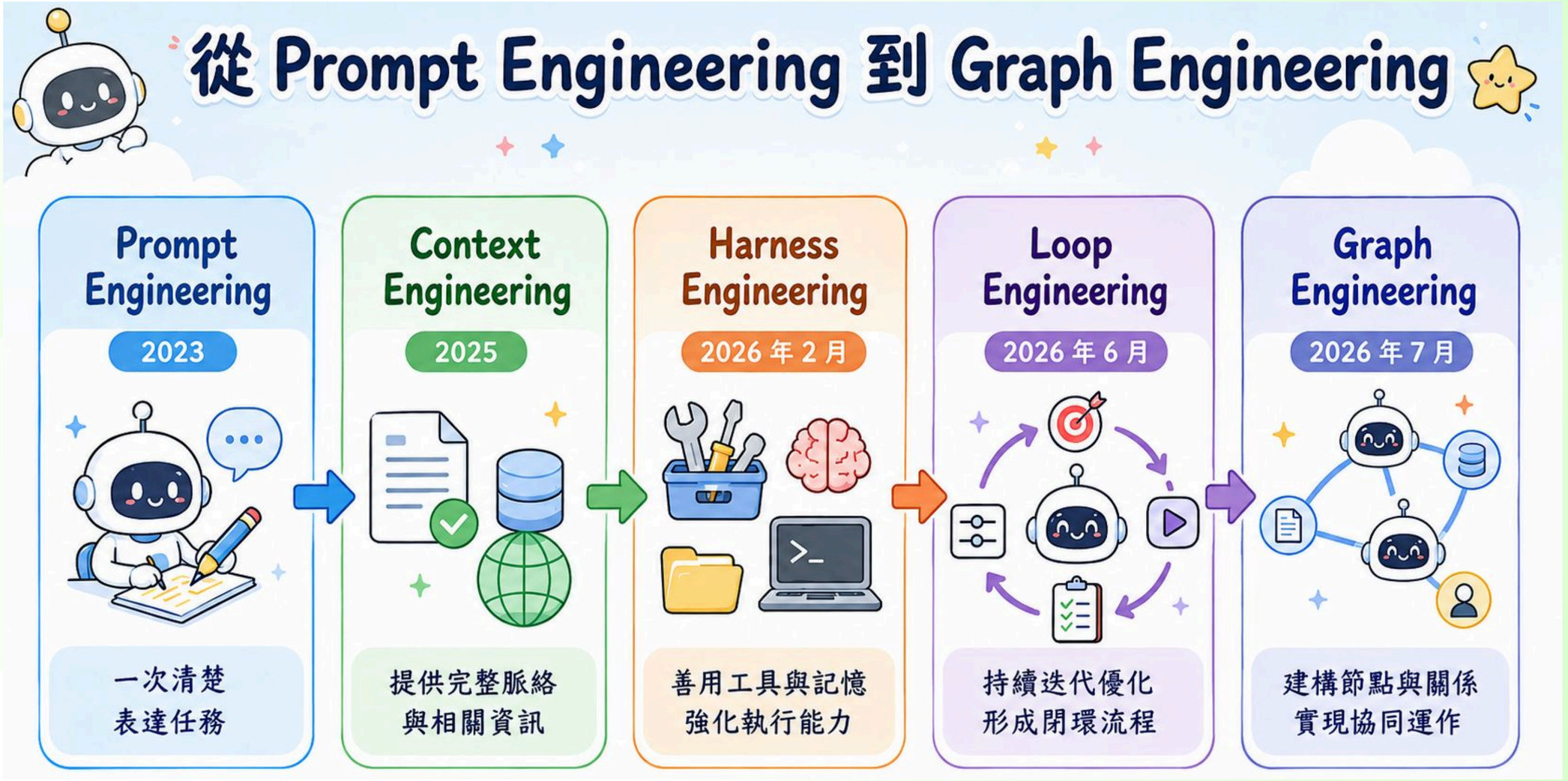
Source: “Using Claude Code: session management and 1M context”

Context Engineering?

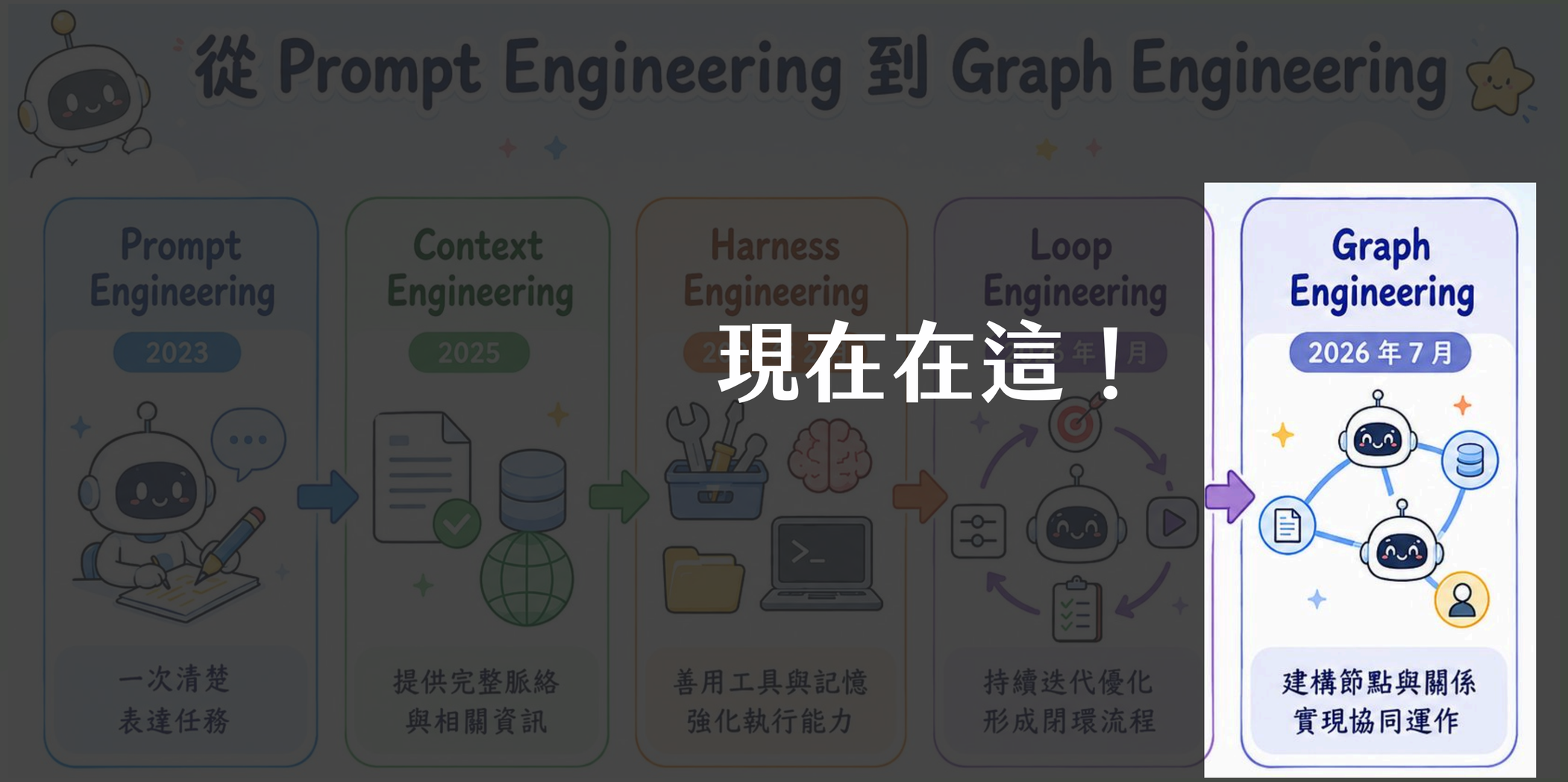
都已經 2026 年 8 月了

誰還在談 Context Engieering 啊！

OOO Engineering?

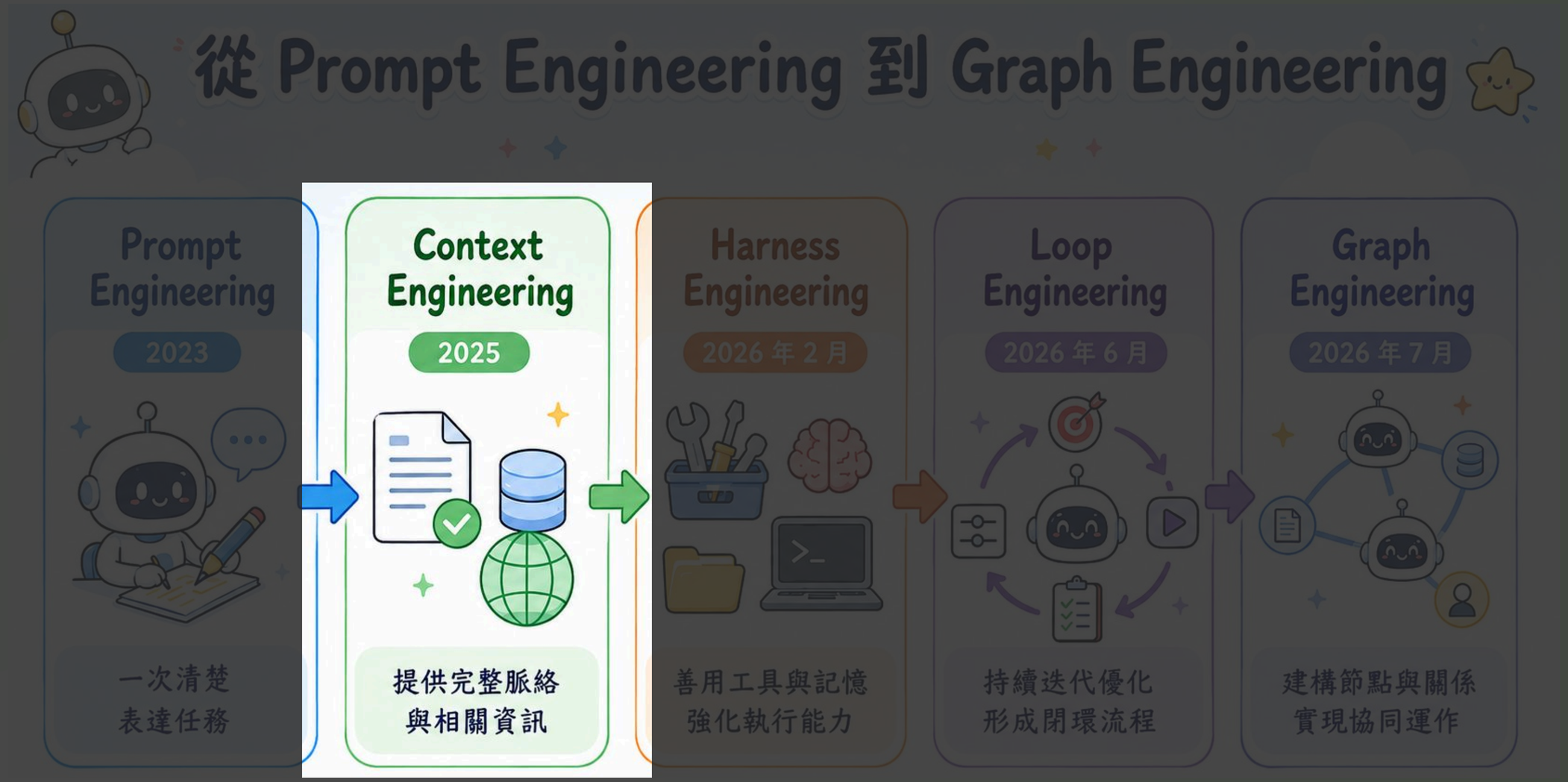


OOO Engineering?

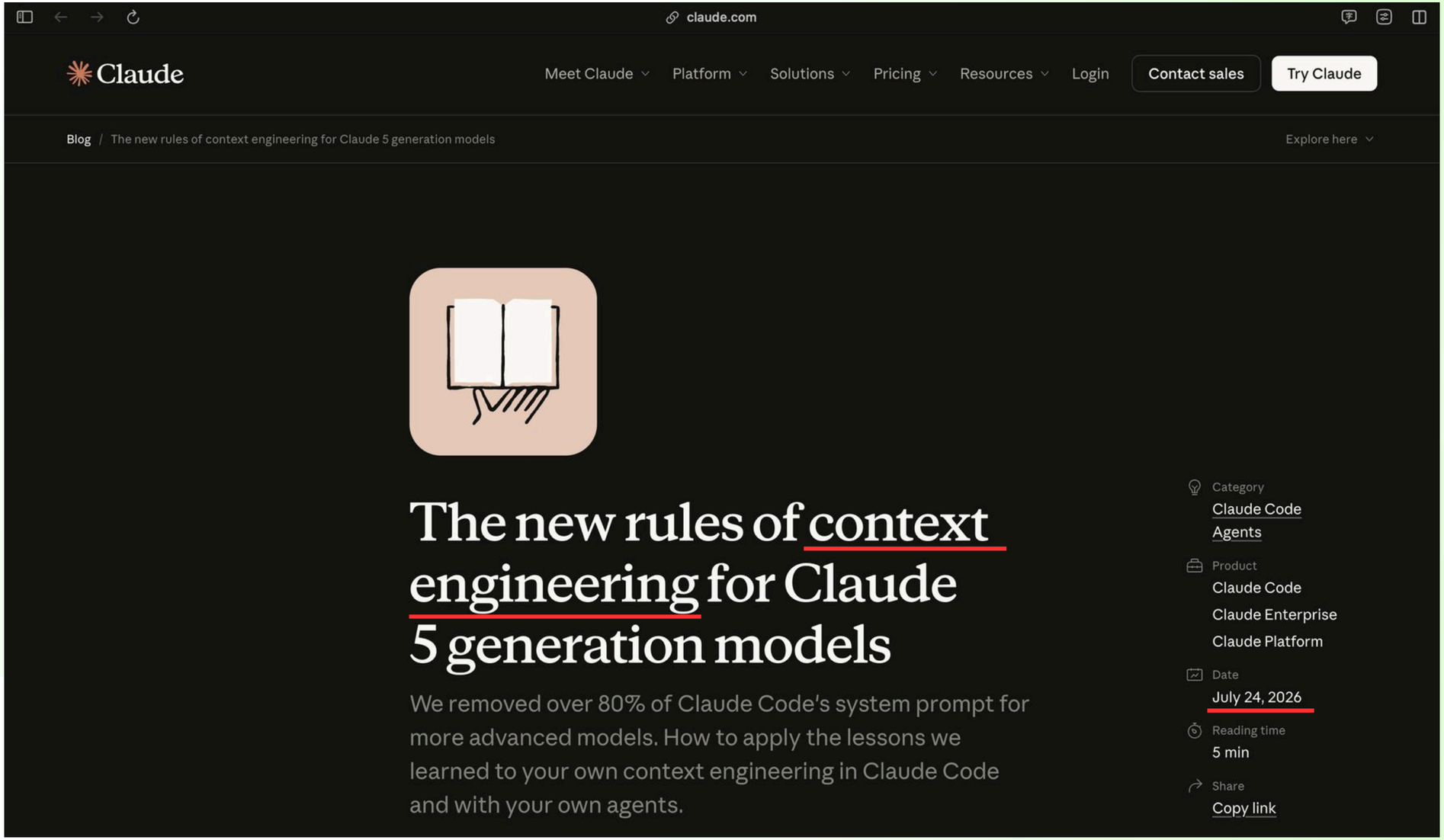


現在在這！

OOO Engineering?



Context Engineering !



為什麼 Context 重要

- Context 有限，LLM 短暫能記住的東西有限
- 影響模型輸出品質
- 跟錢有關

為什麼 Context 重要

- Context 有限，LLM 短暫能記住的東西有限
- 影響模型輸出品質
- 跟錢有關

盡可能讓模型獲得更好的輸入
得到更好的輸出

管理 Context 的主要方法



Skills



Subagents



Hooks

管理 Context 的主要方法



Skills



Subagents



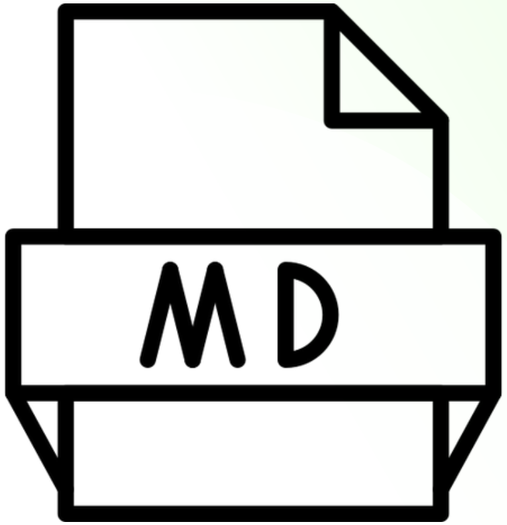
Hooks

Skills?



Skills 是什麼?

Skills 是...



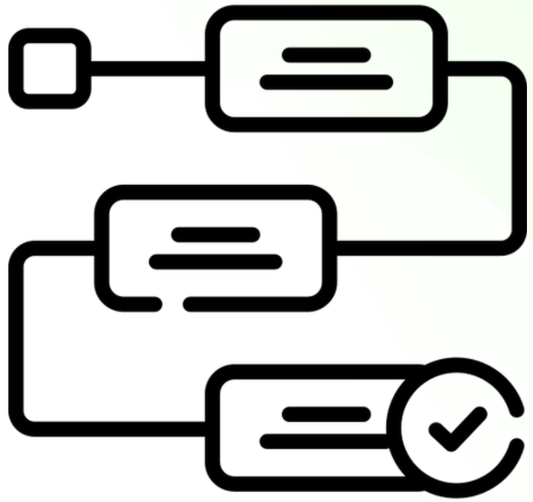
一系列 markdown 文件

Skills 是...



重複指令的 shortcuts

Skills 是...



別人寫好的 workflow

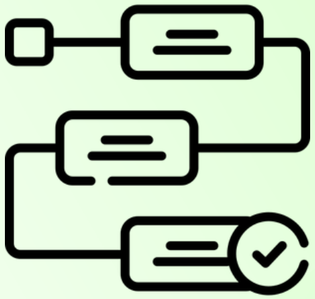
Skills 是...



一系列 markdown 文件



重複指令的 shortcuts



別人寫好的 workflow

為什麼 Skills 重要

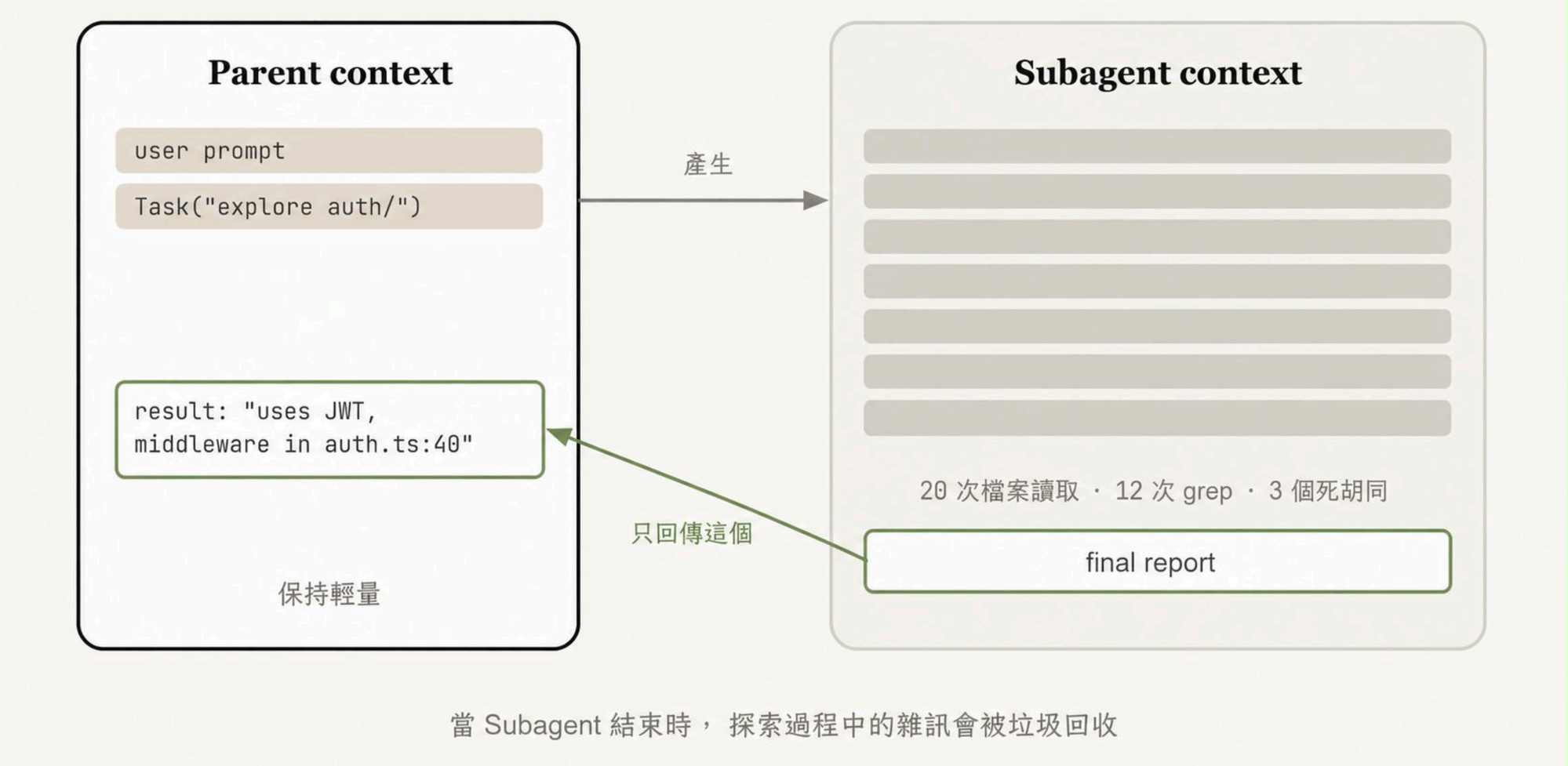
- 漸進式揭露
- LLM 具有隨機性，每次生成的指令不盡相同

Subagents?



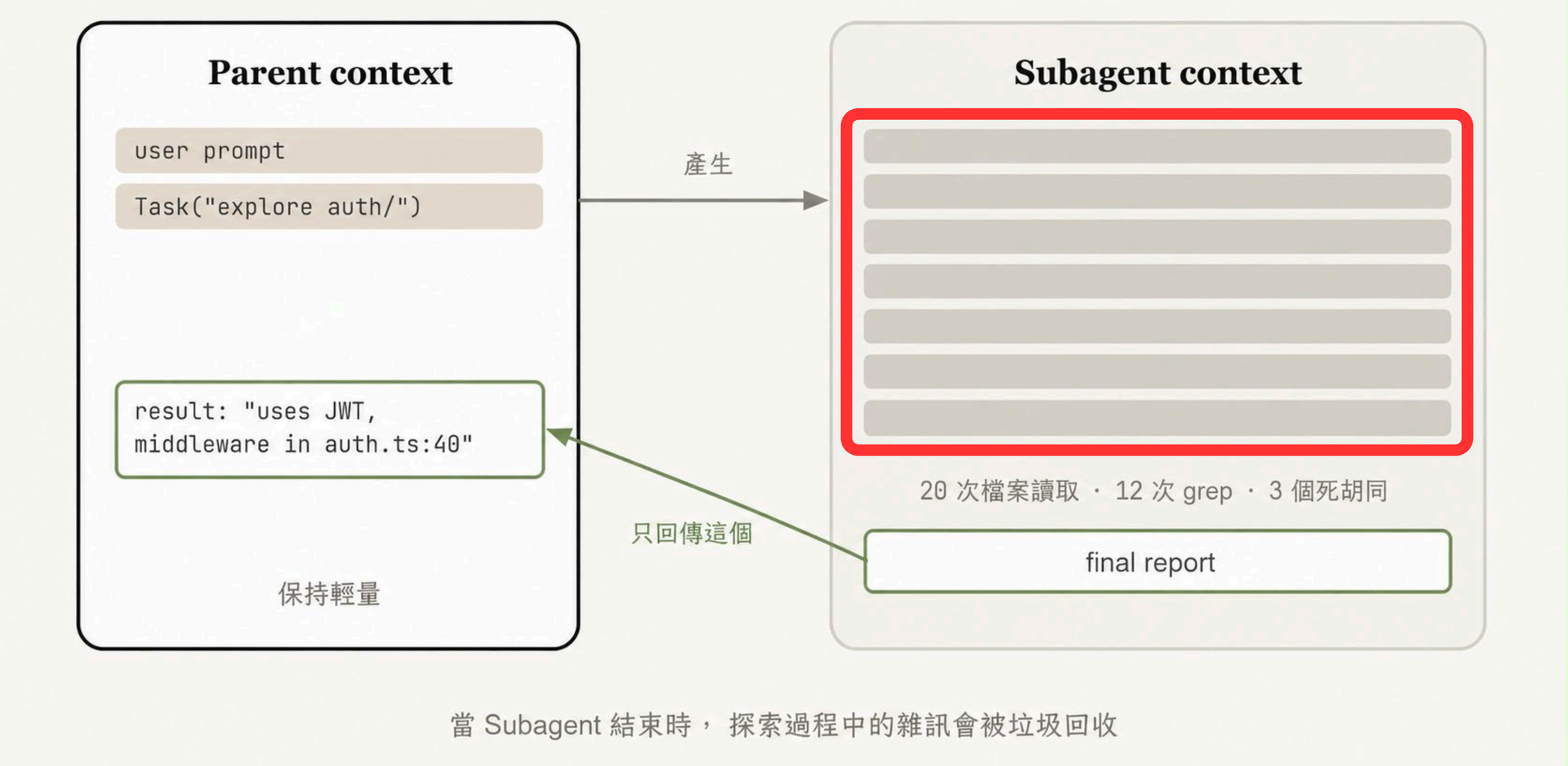
Subagents 是什麼?

Subagents 的好處



Source: “Using Claude Code: session management and 1M context”

Subagents 的好處



Source: "Using Claude Code: session management and 1M context"

為什麼 Subagents 重要

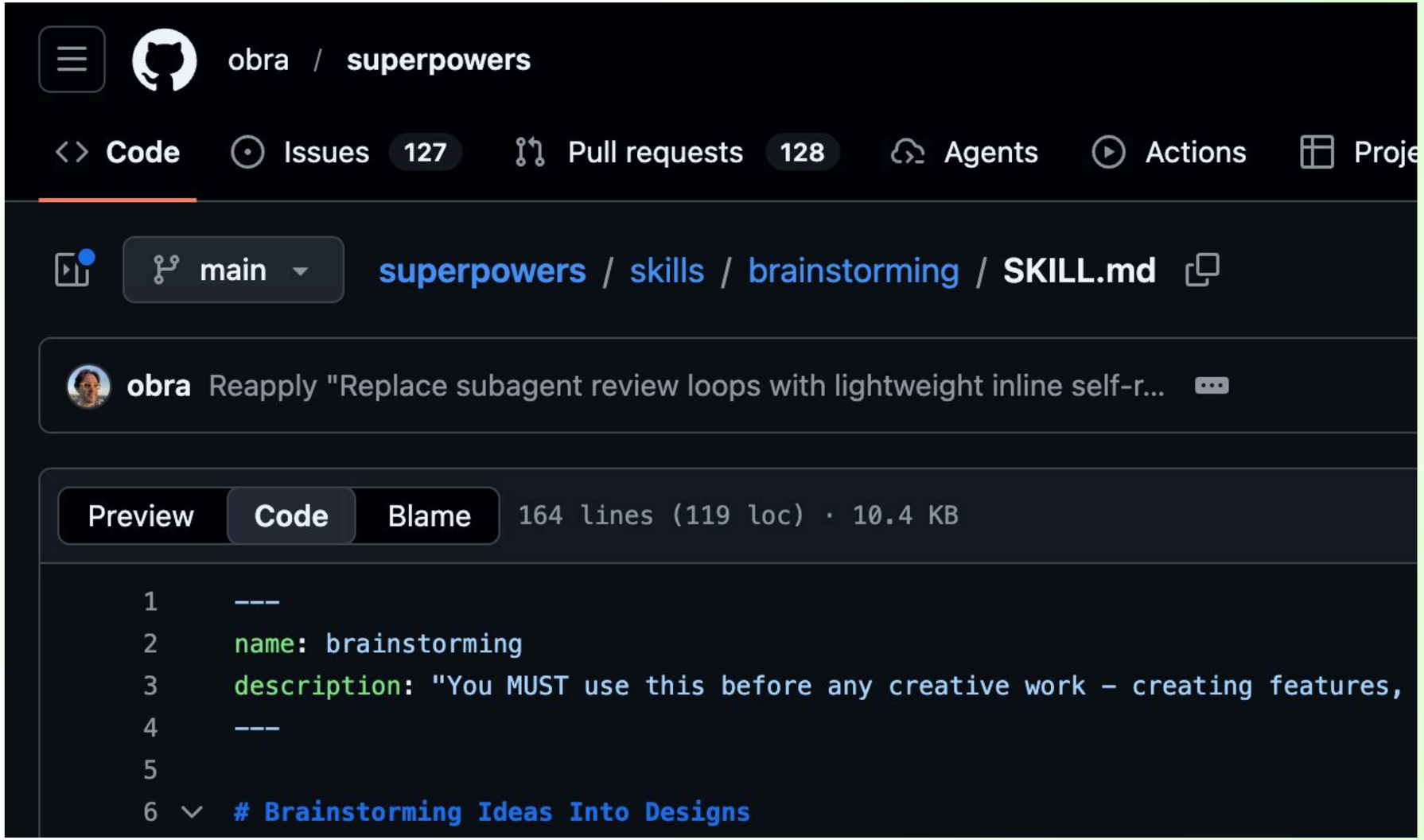
- 分工合作
- 避免為不必要的資訊一直不斷付錢

Skills 結合 Subagents 的 Context Engineering

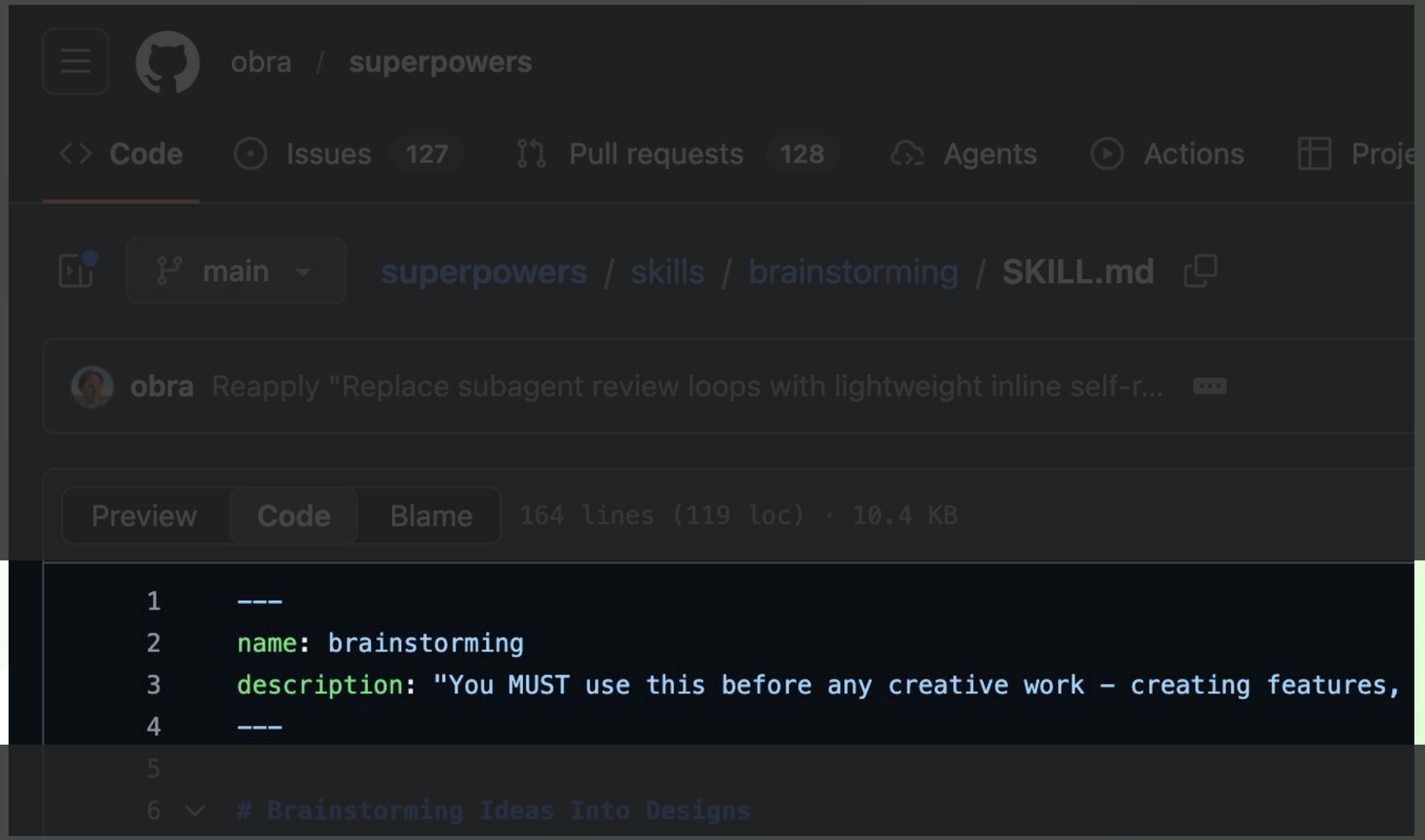


怎麼結合兩者做到呢

先認識機制


obra / superpowers
Code
Issues 127
Pull requests 128
Agents
Actions
Proje
main
superpowers / skills / brainstorming / SKILL.md
obra Reapply "Replace subagent review loops with lightweight inline self-r...
Preview
Code
Blame
164 lines (119 loc) · 10.4 KB
1 ---
2 name: brainstorming
3 description: "You MUST use this before any creative work - creating features,
4 ---
5
6 # Brainstorming Ideas Into Designs

先認識機制


obra / superpowers
Code
Issues 127
Pull requests 128
Agents
Actions
main
superpowers / skills / brainstorming / SKILL.md
obra Reapply "Replace subagent review loops with lightweight inline self-r...
Preview
Code
Blame
164 lines (119 loc) · 10.4 KB
1 ---
2 name: brainstorming
3 description: "You MUST use this before any creative work - creating features,
4 ---
5
6 # Brainstorming Ideas Into Designs

但 Skills 可以做更多事

```yaml
---
name: deep-code-review
description: 深度審查 PR 變更，產出結構化報告

disable-model-invocation: true
argument-hint: "[PR number]"
allowed-tools: Bash(gh *), Bash(git diff *), Read, Grep
paths: "src/**/*.ts"
user-invocable: true

context: fork
agent: Explore
model: sonnet
effort: high

hooks:
  PreToolUse:
    - matcher: Bash
      command: echo "Tool used: $TOOL_NAME"
shell: bash
---
```

但 Skills 可以做更多事

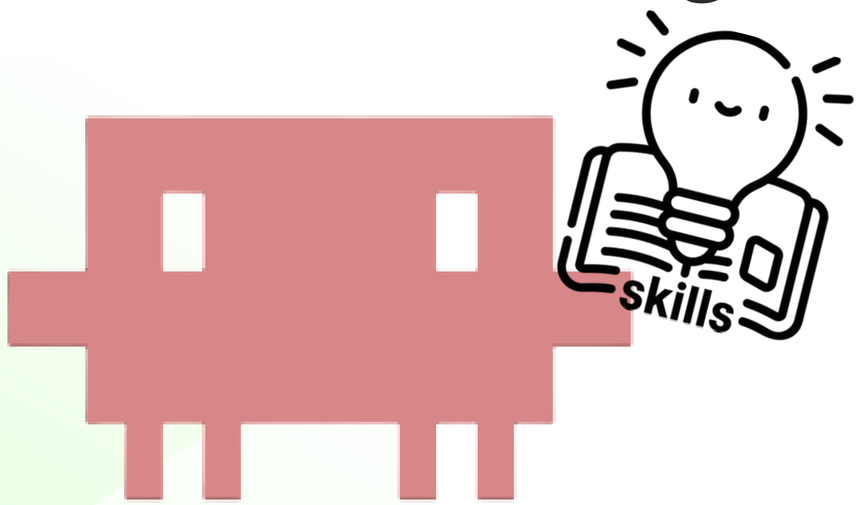
```yaml
---
name: deep-code-review
description: 深度審查 PR 變更，產出結構化報告

disable-model-invocation: true
argument-hint: "[PR number]"
allowed-tools: Bash(gh *), Bash(git diff *), Read, Grep
paths: "src/**/*.ts"
user-invocable: true

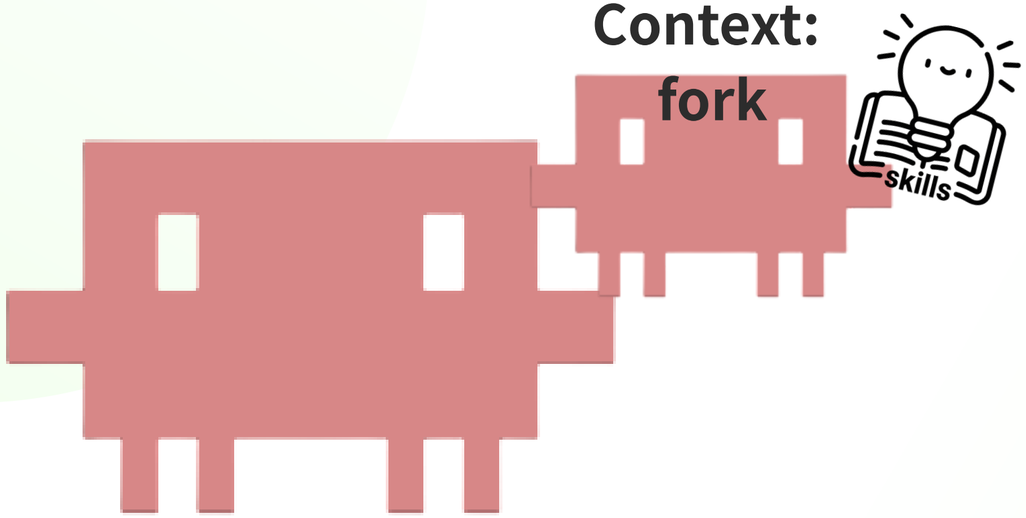
context: fork
agent: Explore
model: sonnet
effort: high

hooks:
  PreToolUse:
    - matcher: Bash
      command: echo "Tool used: $TOOL_NAME"
shell: bash
---
```

Skills vs fork agent Skills



- 常用 Prompt 的快捷鍵
- 能力擴充 e.g., /reddit-search
- 按需載入的背景知識擴充



- 通用性的重複任務
- 不在意過程只在意結果

管理 Context 的主要方法



Skills

Subagents



Hooks

Hooks?

Hooks = 當條件成立時自動發生的副作用

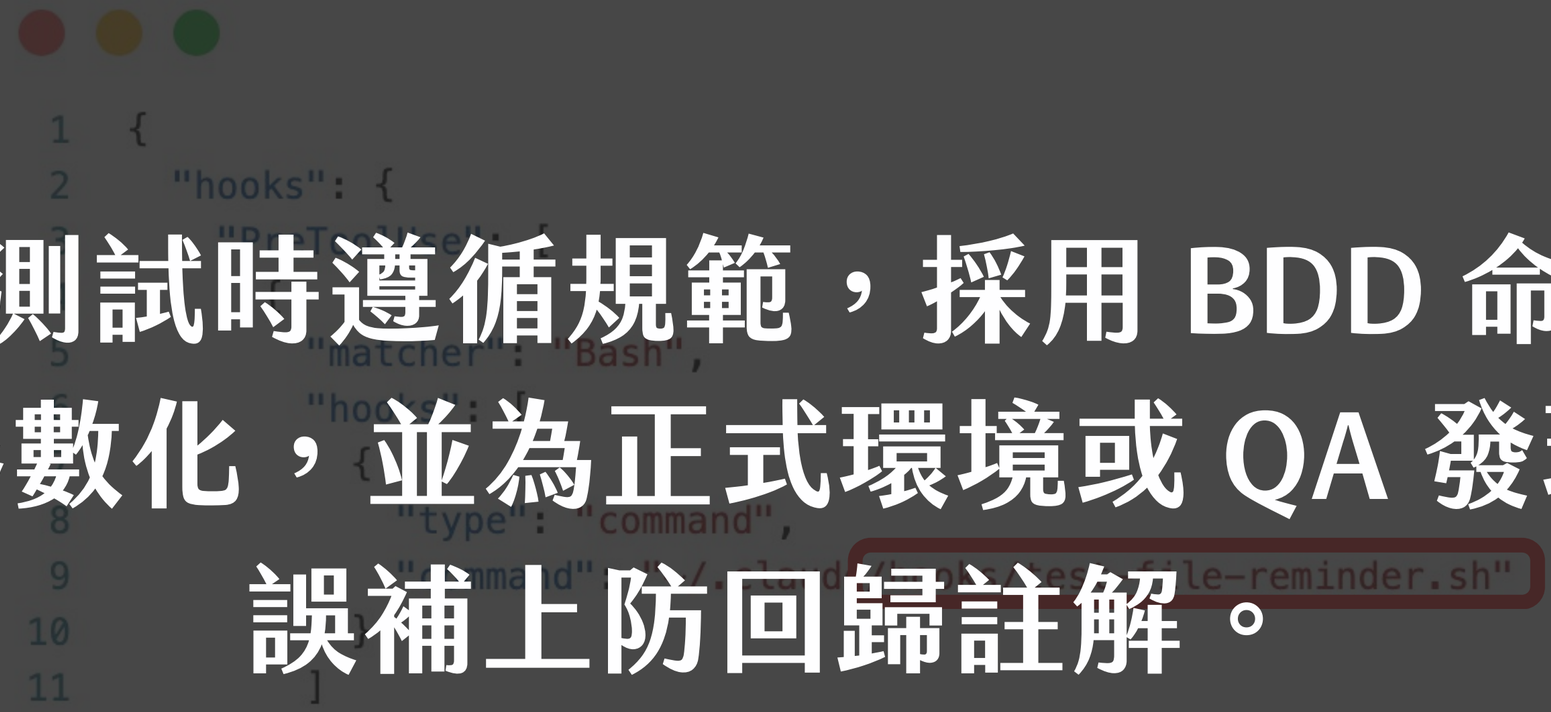
Hooks?

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "if": "Bash(rm *)",
            "command": "${CLAUDE_PROJECT_DIR}/.claude/hooks/block-rm.sh",
            "args": []
          }
        ]
      }
    ]
  }
}
```

Hooks?

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "~/.claude/hooks/test-file-reminder.sh"
          }
        ]
      }
    ]
  }
}
```

Hooks?



撰寫測試時遵循規範，採用 BDD 命名、適度參數化，並為正式環境或 QA 發現的錯誤補上防回歸註解。

小結

- 利用 Skills 的漸進式揭露管理上下文

小結

- 利用 Skills 的漸進式揭露管理上下文
- Subagents 處理「只重結果，不在意過程」的任務

小結

- 利用 Skills 的漸進式揭露管理上下文
- Subagents 處理「只重結果，不在意過程」的任務
- Skills 結合 Subagents 能讓給 Subagents 的指令每次都相同

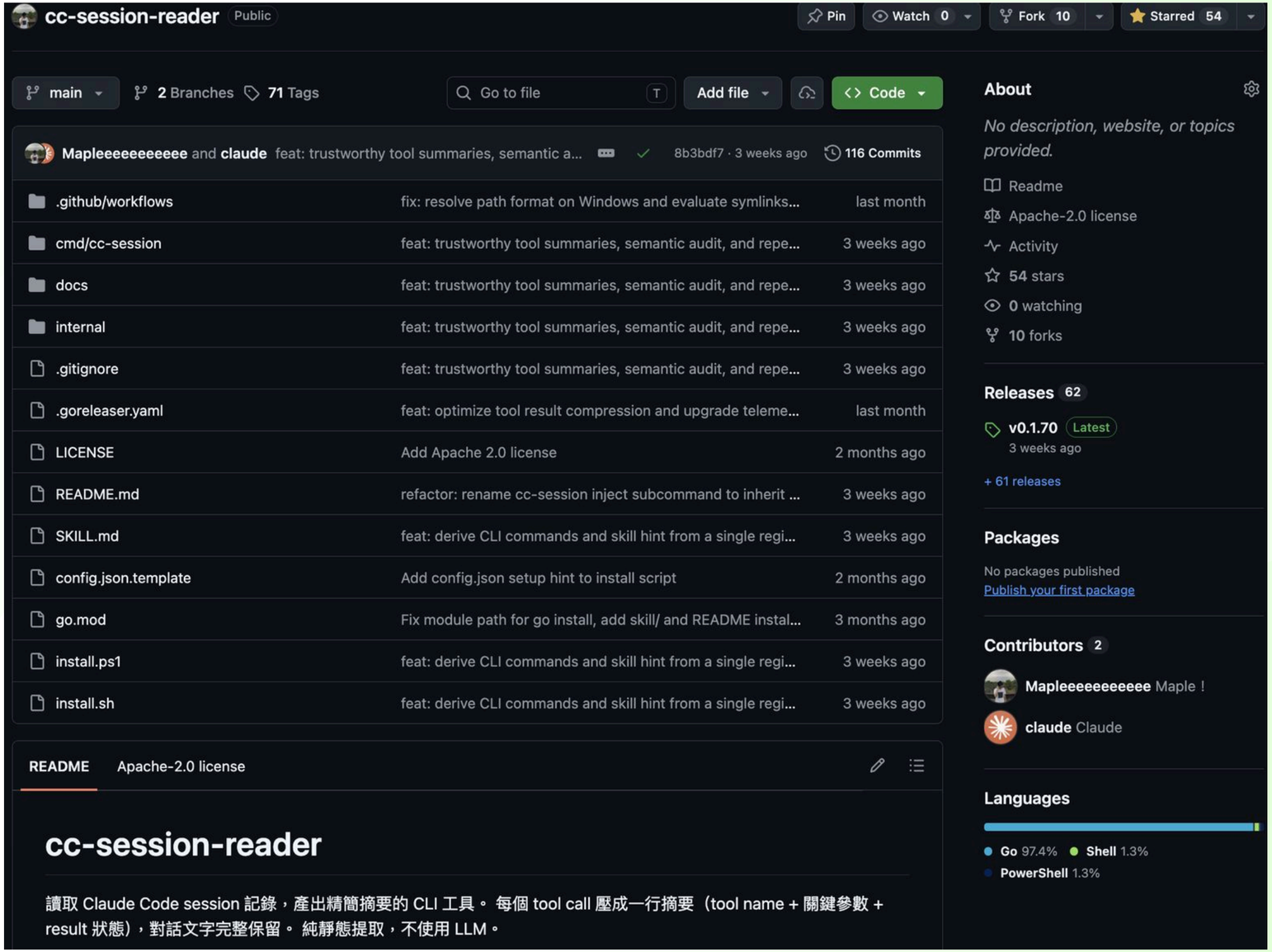
小結

- 利用 Skills 的漸進式揭露管理上下文
- Subagents 處理「只重結果，不在意過程」的任務
- Skills 結合 Subagents 能讓給 Subagents 的指令每次都相同
- Hooks 的初衷是副作用，讓模型能夠得到機械式的回饋

小結

- 利用 Skills 的漸進式揭露管理上下文
- Subagents 處理「只重結果，不在意過程」的任務
- Skills 結合 Subagents 能讓給 Subagents 的指令每次都相同
- Hooks 的初衷是副作用，讓模型能夠得到機械式的回饋
- 可以用來在特定場景自動注入 Prompt

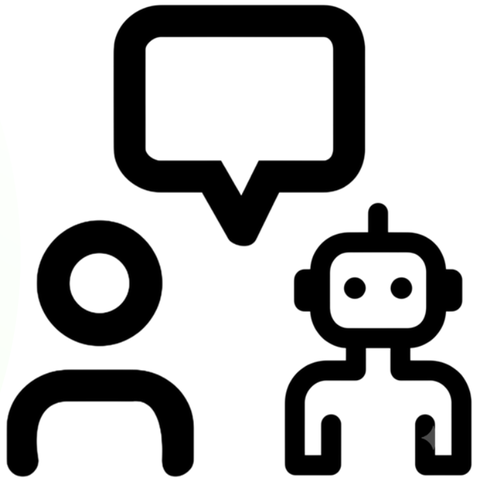
透過自製 CLI tools 做 Context Engineering



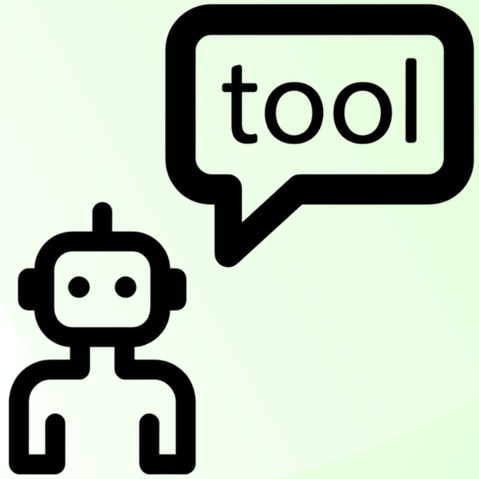
初衷

- 讓 Claude 回顧過去的歷史記錄，找到我可以改善的地方
- 每次 Claude 都在抱怨很多噪音，自己寫程式過濾 JSONL

設計邏輯

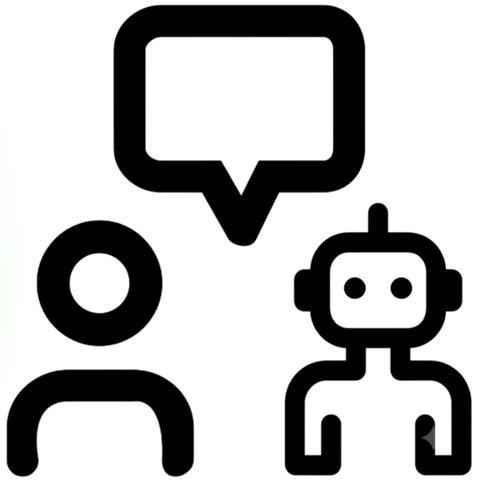


保留 User 及 AI 訊息



壓縮工具細節

設計邏輯



保留 User 及 AI 訊息



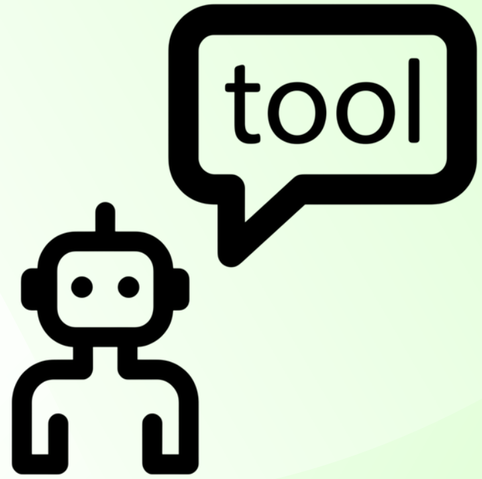
大部分資訊量
都存在於對話

壓縮工具細節

設計邏輯

工具的結果
大多可以重複執行

保留 User 及 AI 訊息



壓縮工具細節

實際效果

```
❯ /cc-session inherit 4df3f890-ef50-44d7-9300-e12982c13919

● Bash(cc-session inherit 4df3f890-ef50-44d7-9300-e12982c13919)
  └ [page 1/4 | lines 1-361 of 971]
    [07-15 02:03] user:
    我想要調整一版，讓 skills 可以 codegen CLI 提供的指令 我就不用手打 inherit 可以 auto complete

    [07-15 02:03] assistant:
    我先了解一下這個專案的結構，特別是 `cc-session` CLI 和對應的 skill 是怎麼組織的，再來釐清你要的方向。

      [Agent(Explore)#GHba] Explore cc-session CLI and skill -> ok

    [07-15 02:03] assistant:
    了解專案結構的同時，我想先確認一下你要的方向。等探索結果回來我會更清楚，但你講的「codegen」和「auto
    complete」其實可能是兩件不同的事，我先理解你的痛點。

      [AskUserQuestion#Einm] Q1: 你說「不用手打 inherit、可以 auto complete」，最主要想在哪裡自動補全 / 少打字?

    [07-15 02:04] user (answer):
    Your questions have been answered: "你說「不用手打 inherit、可以 auto complete」，最主要想在哪裡自動補全 /
    少打字?"="你去了解一下 skills 的機制". You can now continue with these answers in mind.

    [07-15 02:04] assistant:
    你要我先搞懂 skills 的機制再回來談。我來讀 cc-session 這個 skill 本身，並理解 SKILL.md 的運作方式。

      [Bash#xsSn] Locate cc-session SKILL.md files -> ok: === 專案內的 skill 檔案 ===
      [Read#cfCA] cc-session/SKILL.md -> ok

    [07-15 02:04] assistant:
    SKILL.md 我讀完了，關鍵在第 30-44 行那張「子命令速查」表—它是**手寫**的。CLI 一旦改指令名（像
    `inject`→`inherit`），得有人手動改這張表；忘了改，skill 就過時，我讀了會叫你去跑一個已經不存在的指令。這正是你要解決的痛點。

    我等一下 CLI 結構的探索結果，好知道指令是怎麼定義的、能不能自動抽出來。先讓那個 agent 跑完。

      [Bash#ECTy] Peek at project root and language -> ok: === CLI 語言/結構 ===

    [07-15 02:04] assistant:
    Go 專案。等 explore agent 回報指令定義結構。我先看一眼 `cmd/` 怎麼組織的。

      [Bash#7Qop] Inspect cmd structure and CLI framework -> ok: === cmd/ ===
      [Bash#NVut] Check CLI dispatch style in main.go 1 new message (click) ↓
```

實際效果

```
❯ /cc-session inherit 4df3f890-ef50-44d7-9300-e12982c13919

● Bash(cc-session inherit 4df3f890-ef50-44d7-9300-e12982c13919)
  └ [page 1/4 | lines 1-361 of 971]
    [07-15 02:03] user:
    我想要調整一版，讓 skills 可以 codegen CLI 提供的指令 我就不用手打 inherit 可以 auto complete

    [07-15 02:03] assistant:
    我先了解一下這個專案的結構，特別是 `cc-session` CLI 和對應的 skill 是怎麼組織的，再來釐清你要的方向。

      [Agent(Explore)#GHba] Explore cc-session CLI and skill -> ok

    [07-15 02:03] assistant:
    了解專案結構的同時，我想先確認一下你要的方向。等探索結果回來我會更清楚，但你講的「codegen」和「auto
    complete」其實可能是兩件不同的事，我先理解你的痛點。

      [AskUserQuestion#Einm] Q1: 你說「不用手打 inherit、可以 auto complete」，最主要想在哪裡自動補全 / 少打字?

    [07-15 02:04] user (answer):
    Your questions have been answered: "你說「不用手打 inherit、可以 auto complete」，最主要想在哪裡自動補全 /
    少打字?"="你去了解一下 skills 的機制". You can now continue with these answers in mind.

    [07-15 02:04] assistant:
    你要我先搞懂 skills 的機制再回來談。我來讀 cc-session 這個 skill 本身，並理解 SKILL.md 的運作方式。

      [Bash#xsSn] Locate cc-session SKILL.md files -> ok: === 專案內的 skill 檔案 ===
      [Read#cfCA] cc-session/SKILL.md -> ok

    [07-15 02:04] assistant:
    SKILL.md 我讀完了，關鍵在第 30-44 行那張「子命令速查」表—它是**手寫**的。CLI 一旦改指令名（像
    `inject`→`inherit`），得有人手動改這張表；忘了改，skill 就過時，我讀了會叫你去跑一個已經不存在的指令。這正是你要解決的痛點。

    我等一下 CLI 結構的探索結果

      [Bash#ECTy] Peek at project

    [07-15 02:04] assistant:
    Go 專案。等 explore agent 回報指令定義結構。我先看一眼 `cmd/` 怎麼組織的。

      [Bash#7Qop] Inspect cmd structure and CLI framework -> ok: === cmd/ ===
      [Bash#NVut] Check CLI dispatch style in main.go 1 new message (click) ↓
```

完整保留對話歷史

實際效果

```
/cc-session inherit 4df3f890-ef50-44d7-9300-e12982c13919

● Bash(cc-session inherit 4df3f890-ef50-44d7-9300-e12982c13919)
  └ [page 1/4 | lines 1-361 of 971]
    [07-15 02:03] user:
    我想要調整一版，讓 skills 可以 codegen CLI 提供的指令 我就不用手打 inherit 可以 auto complete

    [07-15 02:03] assistant:
    我先了解一下這個專案的結構，特別是 `cc-session` CLI 和對應的 skill 是怎麼組織的，再來釐清你要的方向。

      [Agent(Explore)#GHba] Explore cc-session CLI and skill -> ok

    [07-15 02:03] assistant:
    了解專案結構的同時，我想先確認一下你要的方向。等探索結果回來我會更清楚，但你講的「codegen」和「auto
    complete」其實可能是兩件不同的事，我先理解你的痛點。

      [AskUserQuestion#Einm] Q1: 你說「不用手打 inherit」，是指 auto complete 要補全 / 少打字?

    [07-15 02:04] user (answer):
    Your questions have been answered: 你說「不用手打 inherit」是指 auto complete 要補全 /
    少打字?"="你去了解一下 skills 的機制". You can now continue with these answers in mind.

    [07-15 02:04] assistant:
    你要我先搞懂 skills 的機制再回來談。我來讀 cc-session 這個 skill 本身，並理解 SKILL.md 的運作方式。

      [Bash#xsSn] Locate cc-session SKILL.md files -> ok: === 專案內的 skill 檔案 ===
      [Read#cfCA] cc-session/SKILL.md -> ok

    [07-15 02:04] assistant:
    SKILL.md 我讀完了，關鍵在第 30-44 行那張「子命令速查」表—它是**手寫**的。CLI 一旦改指令名（像
    `inject`→`inherit`），得有人手動改這張表；忘了改，skill 就過時，我讀了會叫你去跑一個已經不存在的指令。這正是你要解決的痛點。

    我等一下 CLI 結構的探索結果，好知道指令是怎麼定義的、能不能自動抽出來。先讓那個 agent 跑完。

      [Bash#ECTy] Peek at project root and language -> ok: === CLI 語言/結構 ===

    [07-15 02:04] assistant:
    Go 專案。等 explore agent 回報指令定義結構。我先看一眼 `cmd/` 怎麼組織的。

      [Bash#7Qop] Inspect cmd structure and CLI framework -> ok: === cmd/ ===
      [Bash#NVut] Check CLI dispatch style in main.go 1 new message (click) ↓
```

工具呼叫記錄壓縮

取代 Auto Compact?



既然有壓縮效果，能不能取代自動壓縮呢

Auto Compact 的原理



Source: "Using Claude Code: session management and 1M context"

量化分析

Session	Context	NewCtx	Context 省	K	回本	10-turn 省	100-turn 省
b470eacd	415K	134K	67.7%	3.3	turn 3	45%	45%
acecda6c	410K	87K	78.8%	4.8	turn 1	63%	41%
c2323694	406K	79K	80.6%	3.8	turn 2	64%	42%
897479c9	406K	126K	68.8%	4.7	turn 2	52%	42%
eabbdd84	404K	106K	73.8%	5.9	turn 2	60%	43%
e61060b1	403K	169K	58.2%	4.5	turn 4	36%	42%
0b81c6d8	402K	118K	70.5%	4.5	turn 2	52%	37%
33e631ce	401K	85K	78.7%	4.2	turn 2	56%	29%
2bee938d	398K	115K	71.1%	7.5	turn 1	57%	36%
4e820fe3	390K	82K	79.1%	6.1	turn 1	67%	44%
8671d55e	387K	64K	83.4%	10.7	turn 1	65%	27%
e1f0d957	383K	73K	80.8%	3.5	turn 2	64%	45%
977b7360	381K	89K	76.6%	5.5	turn 1	62%	42%
270e2323	367K	73K	80.2%	7.4	turn 1	67%	39%
c913bf77	354K	107K	69.7%	7.9	turn 2	51%	24%
中位數			**76.6%**		**turn 2**	**60%**	**42%**

量化分析

Session	Context	NewCtx	Context 省	K	回本	10-turn 省	100-turn 省
b470eacd	415K	134K	67.7%	3.3	turn 3	45%	45%
acecda6c	410K	87K	78.8%	4.8	turn 1	63%	41%
c2323694	406K	79K	80.6%	3.8	turn 2	64%	42%
897479c9	406K	126K	68.8%	4.7	turn 2	52%	42%
eabbdd84	404K	106K	73.8%	5.9	turn 2	60%	43%
e61060b1	403K	169K	58.2%	4.5	turn 4	36%	42%
0b81c6d8	402K	118K	70.5%	4.5	turn 2	52%	37%
33e631ce	401K	85K	78.7%	4.2	turn 2	56%	29%
2bee938d	398K	115K	71.1%	7.5	turn 1	57%	36%
4e820fe3	390K	82K	79.1%	6.1	turn 1	67%	44%
8671d55e	387K	64K	83.4%	10.7	turn 1	65%	27%
e1f0d957	383K	73K	80.8%	3.5	turn 2	64%	45%
977b7360	381K	89K	76.6%	5.5	turn 1	62%	42%
270e2323	367K	73K	80.2%	7.4	turn 1	67%	39%
c913bf77	354K	107K	69.7%	7.9	turn 2	51%	24%
中位數			**76.6%**		**turn 2**	**60%**	**42%**

量化分析

Session	Context	NewCtx	Context 省	K	回本	10-turn 省	100-turn 省
b470eacd	415K	134K	67.7%	3.3	turn 3	45%	45%
acecda6c	410K	87K	78.8%	4.8	turn 1	63%	41%
c2323694	406K	79K	80.6%	3.8	turn 2	64%	42%
897479c9	406K	126K	68.8%	4.7	turn 2	52%	42%
eabbdd84	404K	106K	73.8%	5.9	turn 2	60%	43%
e61060b1	403K	169K	58.2%	4.5	turn 4	36%	42%
0b81c6d8	402K	118K	70.5%	4.5	turn 2	52%	37%
33e631ce	401K	85K	78.7%	4.2	turn 2	56%	29%
2bee938d	398K	115K	71.1%	7.5	turn 1	57%	36%
4e820fe3	390K	82K	79.1%	6.1	turn 1	67%	44%
8671d55e	387K	64K	83.4%	10.7	turn 1	65%	27%
e1f0d957	383K	73K	80.8%	3.5	turn 2	64%	45%
977b7360	381K	89K	76.6%	5.5	turn 1	62%	42%
270e2323	367K	73K	80.2%	7.4	turn 1	67%	39%
c913bf77	354K	107K	69.7%	7.9	turn 2	51%	24%
中位數			**76.6%**		**turn 2**	**60%**	**42%**

量化分析

Session	Context	NewCtx	Context 省	K	回本	10-turn 省	100-turn 省
b470eacd	415K	134K	67.7%	3.3	turn 3	45%	45%
acecda6c	410K	87K	78.8%	4.8	turn 1	63%	41%
c2323694	406K	79K	80.6%	3.8	turn 2	64%	42%
897479c9	406K	126K	68.8%	4.7	turn 2	52%	42%
eabbdd84	404K	106K	73.8%	5.9	turn 2	60%	43%
[illegible]	[illegible]	[illegible]	58.2%	4.5	turn 4	36%	42%
0b81c6d8	402K	118K	70.5%	4.5	turn 2	52%	37%
[illegible]	[illegible]	[illegible]	[illegible]	4.2	turn 2	56%	29%
[illegible]	[illegible]	[illegible]	[illegible]	7.5	turn 1	57%	36%
4e820fe3	390K	82K	79.1%	6.1	turn 1	67%	44%
8671d55e	387K	64K	83.4%	10.7	turn 1	65%	27%
e1f0d957	383K	73K	80.8%	3.5	turn 2	64%	45%
977b7360	381K	89K	76.6%	5.5	turn 1	62%	42%
270e2323	367K	73K	80.2%	7.4	turn 1	67%	39%
c913bf77	354K	107K	69.7%	7.9	turn 2	51%	24%
中位數			76.6%		turn 2	60%	42%

K = 每次下 prompt
模型呼叫工具的平均次數

量化分析

Session	Context	NewCtx	Context 省	K	回本	10-turn 省	100-turn 省
b470eacd	415K	134K	67.7%	3.3	turn 3	45%	45%
acecda6c	410K	87K	78.8%	4.8	turn 1	63%	41%
c2323694	406K	79K	80.6%	3.8	turn 2	64%	42%
897479c9	406K	126K	68.8%	4.7	turn 2	52%	42%
eabbdd84	404K	106K	73.8%	5.9	turn 2	60%	43%
e61060b1	403K	169K	58.2%	4.5	turn 4	36%	42%
0b81c6d8	402K	118K	70.5%	4.5	turn 2	52%	37%
33e631ce	401K	85K	78.7%	4.2	turn 2	56%	29%
2bee938d	398K	115K	71.1%	7.5	turn 1	57%	36%
4e820fe3	390K	82K	79.1%	6.1	turn 1	67%	44%
8671d55e	387K	64K	83.4%	10.7	turn 1	65%	27%
e1f0d957	383K	73K	80.8%	3.5	turn 2	64%	45%
977b7360	381K	89K	76.6%	5.5	turn 1	62%	42%
270e2323	367K	73K	80.2%	7.4	turn 1	67%	39%
c913bf77	354K	107K	69.7%	7.9	turn 2	51%	24%
中位數			76.6%		turn 2	60%	42%

量化分析

e61060b1：Warm old vs NewCtx 成本曲線

成本為從決策點開始的累積 input/cache 美元，不含 output，也不含已發生的歷史成本。ctx 累積欄是同一成本模型推估的下一輪 prefix tokens。共同續聊從 turn 1 開始。

Context 403K | NewCtx 169K | Context 省 58.2% | K 4.5 | Inject pages 16 | Setup $1.86

決策點 / 共同續聊 turns	Old ctx 累積	NewCtx 累積	Warm old	NewCtx	省下	省錢比率
決策點（turn 0）	403K	169K	既有 ctx cache	Setup $1.86	不含 sunk cost	-
1	406K	171K	$0.93	$2.25	-$1.32	-143.1%
2	408K	173K	$1.86	$2.65	-$0.80	-42.8%
3	411K	176K	$2.80	$3.06	-$0.27	-9.6%
4	414K	179K	$3.74	$3.48	$0.26	7.0%
5	416K	181K	$4.69	$3.90	$0.79	16.9%
10	430K	195K	$9.53	$6.09	$3.43	36.0%
20	456K	221K	$19.65	$10.93	$8.72	44.4%
50	535K	300K	$53.58	$29.00	$24.58	45.9%
100	668K	433K	$122.04	$71.02	$51.02	41.8%
200	932K	697K	$303.60	$199.72	$103.88	34.2%

量化分析

e61060b1：Warm old vs NewCtx 成本曲線

成本為從決策點開始的累積 input/cache 美元，不含 output，也不含已發生的歷史成本。ctx 累積欄是同一成本模型推估的下一輪 prefix tokens。共同續聊從 turn 1 開始。

Context 403K | NewCtx 169K | Context 省 58.2% | K 4.5 | Inject pages 16 | Setup $1.86

決策點 / 共同續聊 turns	Old ctx 累積	NewCtx 累積	Warm old	NewCtx	省下	省錢比率
決策點（turn 0）	403K	169K	既有 ctx cache	Setup $1.86	不含 sunk cost	-
1	406K	171K	$0.93	$2.25	-$1.32	-143.1%
2	408K	173K	$1.86	$2.65	-$0.80	-42.8%
3	411K	176K	$2.80	$3.06	-$0.27	-9.6%
4	414K	179K	$3.74	$3.48	$0.26	7.0%
5	416K	181K	$4.69	$3.90	$0.79	16.9%
10	430K	195K	$9.53	$6.09	$3.43	36.0%
20	456K	221K	$19.65	$10.93	$8.72	44.4%
50	535K	300K	$53.58	$29.00	$24.58	45.9%
100	668K	433K	$122.04	$71.02	$51.02	41.8%
200	932K	697K	$303.60	$199.72	$103.88	34.2%

量化分析

e61060b1：Warm old vs NewCtx 成本曲線

成本為從決策點開始的累積 input/cache 美元，不含 output，也不含已發生的歷史成本。ctx 累積欄是同一成本模型推估的下一輪 prefix tokens。共同續聊從 turn 1 開始。

Context 403K | NewCtx 169K | Context 省 58.2% | K 4.5 | Inject pages 16 | Setup $1.86

決策點 / 共同續聊 turns	Old ctx 累積	NewCtx 累積	Warm old	NewCtx	省下	省錢比率
決策點（turn 0）	403K	169K	既有 ctx cache	Setup $1.86	不含 sunk cost	-
1	406K	171K	$0.93	$2.25	-$1.32	-143.1%
2	408K	173K	$1.86	$2.65	-$0.80	-42.8%
3	411K	176K	$2.80	$3.06	-$0.27	-9.6%
4	414K	179K	$3.74	$3.48	$0.26	7.0%
5	416K	181K	$4.69	$3.90	$0.79	16.9%
10	430K	195K	$9.53	$6.09	$3.43	36.0%
20	456K	221K	$19.65	$10.93	$8.72	44.4%
50	535K	300K	$53.58	$29.00	$24.58	45.9%
100	668K	433K	$122.04	$71.02	$51.02	41.8%
200	932K	697K	$303.60	$199.72	$103.88	34.2%

量化分析

e61060b1：Warm old vs NewCtx 成本曲線

成本為從決策點開始的累積 input/cache 美元，不含 output，也不含已發生的歷史成本。ctx 累積欄是同一成本模型推估的下一輪 prefix tokens。共同續聊從 turn 1 開始。

Context 403K | NewCtx 169K | Context 省 58.2% | K 4.5 | Inject pages 16 | Setup $1.86

決策點 / 共同續聊 turns	Old ctx 累積	NewCtx 累積	Warm old	NewCtx	省下	省錢比率
決策點（turn 0）	403K	169K	既有 ctx cache	Setup $1.86	不含 sunk cost	-
1	406K	171K	$0.93	$2.25	-$1.32	-143.1%
2	408K	173K	$1.86	$2.65	-$0.80	-42.8%
3	411K	176K	$2.80	$3.06	-$0.27	-9.6%
4	414K	179K	$3.74	$3.48	$0.26	7.0%
5	416K	181K	$4.69	$3.90	$0.79	16.9%
10	430K	195K	$9.53	$6.09	$3.43	36.0%
20	456K	221K	$19.65	$10.93	$8.72	44.4%
50	535K	300K	$53.58	$29.00	$24.58	45.9%
100	668K	433K	$122.04	$71.02	$51.02	41.8%
200	932K	697K	$303.60	$199.72	$103.88	34.2%

總結

- Context Engineering 仍不過時

總結

- Context Engineering 仍不過時
- 各種名詞本質是怎麼讓模型有更好的輸入，以獲得更好的輸出

總結

- Context Engineering 仍不過時
- 各種名詞本質是怎麼讓模型有更好的輸入，以獲得更好的輸出
- 善用 Skills, Subagents, Hooks 等機制來達到上下文管理

附件

Claude Taiwan

cc-session reader



Maple Kuo

AI 時代出身的軟體工程師

感謝你！

contact.maple.kuo@gmail.com