

> Hello World Conf

看懂 AI 開發工具的計費結構

如何省得到錢

Duran Hsieh - Microsoft MVP • GitHub Star



ABOUT ME · 講師介紹

Duran Hsieh (謝政廷)



開發 · 測試 · 效能調教

GitHub Copilot

DevOps

Azure

專長於開發、測試、效能調教、GitHub、GitHub Copilot、DevOps、Azure 開發相關技術，曾擔任技術顧問，提供國內外企業技術諮詢與數位轉型。台灣第一位 GitHub Star、Study4TW 社群核心成員、GDG Taichung 共同創辦人、微軟最有價值專家、書籍作家、大學業界講師、技術部落客。



Duran Blogger

<https://dog0416.blogspot.com>



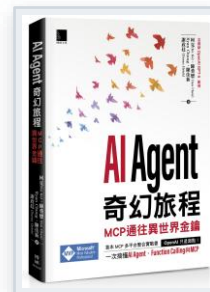
YouTube · FB

Duran 技術冶煉廠

認證 · 社群



著作
BOOKS



實驗 · 同一句話，送兩次

在 VS Code 送出 Hello World

第一次

6.64

個計價單位

第二次

< 1

個計價單位

同樣 11 個字元，價格差了 6 倍以上。差在哪裡？

拆解 · 那份請求長什麼樣

你打的字，只佔請求的一小角



← 工具在背後自動組出來的

你打的 Hello World ↑

常駐指令檔

copilot-instructions.md、
AGENTS.md、Skills 描述，每一輪都帶

工具與 MCP schema

每個工具的名稱、說明、參數格式，
開越多越長

對話歷史

之前所有的問答、讀過的檔案、工具
結果

比例為示意。跨工具佐證：Claude Code 只回一句「OK」前就先送出約 33,000 token (Systima 2026/07 側錄)

帳單算的是 token，而且有四種價格

文字先被切成 token 再送進模型。英文常見詞多半一個 token，程式碼符號和中文切得更碎。

Hello _ World

修 好 失敗 的 測 試

切法為示意，實際依各模型 tokenizer 而定

輸入

你送進去的整份請求，基準價

快取輸入

開頭沒變的部分，約一折

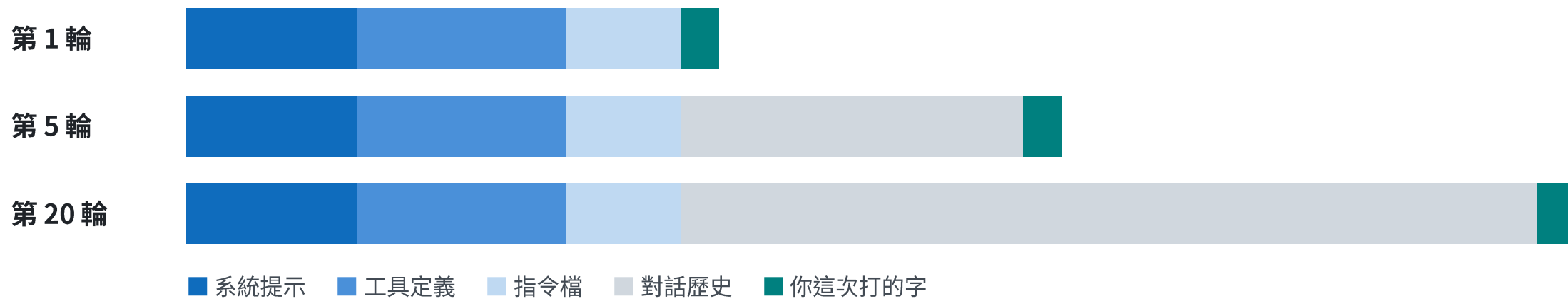
輸出

模型寫出來的字，常是輸入的 3-5 倍

推理

看不到的思考，照輸出價計費

模型沒有記憶：每一輪都把全部重送一次



你這次打的字往往是最小的一段。前三段每輪固定重送，歷史則越長越貴，整段對話的累計成本會隨輪數**近似平方成長**。

快取：前綴一字不差，才能便宜重用

模型處理每個 token 時會算出中間結果（KV cache）。下一次請求的開頭若完全相同，供應商就直接重用，這段以約一折計價、回應也更快。

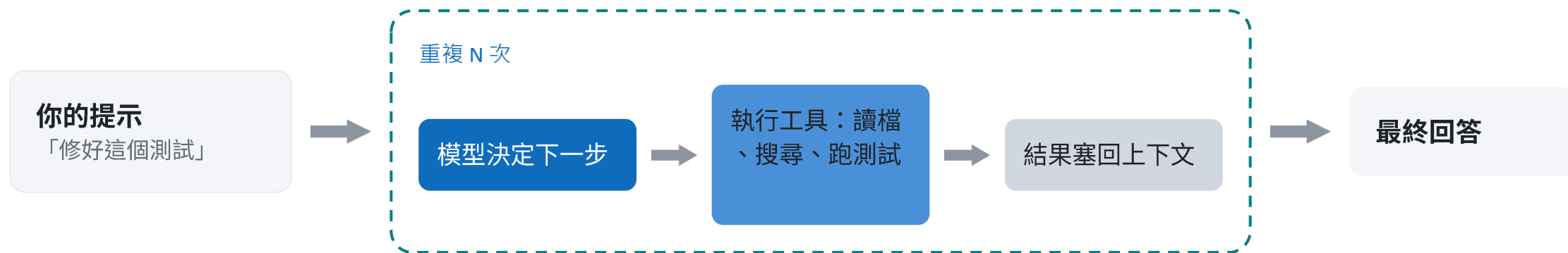
好的順序	系統提示+工具+指令（命中，約一折）	舊歷史（命中）	新訊息
開頭放時間戳	10:42	後面全部對不上 → 全部原價	

不變的放前面，會變的放最後

對話中別換模型、工具組、指令檔

有時效：常見 5-10 分鐘沒用就失效

一個任務，其實是十幾次 API 呼叫



每一圈都是完整請求

原理 2 的整條長條，每次工具呼叫都要再送一遍

工具輸出會累積

讀過的檔案、測試 log 都留在上下文，後面每圈都帶著

走錯路最貴

每多一圈探索或重試，成本就疊一層

一個簡單任務，兩分鐘燒掉 240 萬 token

2.4M

VS Code Agent 模式，起始提示只有約 3,000 token。兩分鐘後，累積的輸入 token 已經到 240 萬。

原因：每次工具呼叫都重送整包系統提示與歷史，agent 還讀了大量用不到的檔案。

200K

第 10 次工具呼叫時，單一請求的大小。
每一步都比上一步更貴

為什麼帳單會爆炸

單價一直跌，帳單卻一直漲

5–30×

Agent 工作流程每個任務耗用的 token，是一般聊天的 5–30 倍；一個任務可觸發 10–20 次模型呼叫。

62%

Agent 帳單花在重送系統提示、工具定義與歷史：付錢讓模型重讀它看過的內容

30%↓

輸入越長品質越差：中段位置的準確率可掉 30%，花更多錢還換來重試

4 月

Uber 2026 全年 AI 預算在 4 月中就用完。省錢的關鍵在架構，不在等降價

優先順序

省錢的正確順序

1

先控 output

單價最高、不能快取，而且輸出會變成下一輪的歷史，等於付兩次

2

再瘦結構性 input

指令檔、工具 schema、歷史每一輪都重送；即使快取打一折，乘上輪數仍是大頭

3

養成快取友善習慣

結構瘦了之後，讓剩下的前綴盡量走一折；避開會讓快取整段失效的動作

讓模型少寫、少想、只回該回的

指令檔預設簡短

寫進 copilot-instructions.md：只給 diff 與一句說明，不貼完整檔案、不寫總結報告

推理強度用預設

例行修改不需要高強度思考；難題再調高，推理 token 照輸出價收費

提示裡給格式與長度

「三點說明」「只回程式碼」「表格不超過五列」

問問題用 Ask 模式

只是要解釋，不需要 Agent 去讀檔、跑工具、多繞幾圈

每一輪都要帶的東西，越小越好

結構	怎麼瘦	在 VS Code 哪裡改
常駐指令檔	只留每次都用得到的規則；流程搬進 Skill；用 applyTo 限定路徑	.github/copilot-instructions.md 與 *.instructions.md
工具與 MCP	只開這次任務用得到的工具；常用流程做成精簡的自訂 agent	聊天面板 Configure Tools、.vscode/mcp.json
對話歷史	換主題就開新對話；同主題岔出去用 Fork	Ctrl+N、/fork、/compact
搜尋範圍	log、大型 JSON、產出物不讓 agent 搜	search.exclude、.gitignore

官方實驗：VS Code 啟用 tool search 後，中位數用戶整個 session 的 token 降 18%

修剪工具、改用 CLI，真實流程省 43–62%

修剪沒用到的 MCP 工具

40 個工具的 schema 每回合佔 10–15 KB，即使只用 2 個。比對「註冊 vs 實際呼叫」後修剪，每回合省 8–12 KB

固定讀取交給 CLI

PR diff、檔案內容用 gh CLI 在 agent 啟動前先下載，移出推理迴圈。最省的呼叫是不打的那通

每日 Token Auditor

自動揪出用量異常的工作流程，開 Issue 建議修法

一條錯誤設定，64 回合失控

一條錯誤的 allowlist 曾讓 agent 跑了 64 回合迴圈。沒有量測，就看不到

哪些日常動作，會讓快取整段重算？

對話中換模型

不同模型不共用快取

中途改推理強度

VS Code 規劃將它標為會推高成本的動作

開關工具或 MCP

工具 schema 在前綴裡，一變後面全部對不上

改指令檔

對話進行中編輯 copilot-instructions.md

隔太久才回來

快取有存活時間，午餐後接續長對話最貴

壓縮對話

/compact 後歷史換成摘要，下一輪前綴改變

對策：要換設定就開新對話；固定的放前面；長對話別中途改裝備

把會變的東西搬到最後，命中率 7% → 84%

他們怎麼做

分三層快取斷點：系統提示、靜態工具定義、對話滑動視窗。再把每一步都會變的 working memory 移出前綴、改放訊息尾端，命中率一夜從 7% 跳到 74%。

快取殺手：別放進前綴

即時時間戳、session 或 request ID、動態註冊的 MCP 工具、每次都變的個人化資料。系統提示開頭一行「現在時間」，就能讓一整天 0 命中、成本多 10 倍。

快取命中率

7% → 84%

整體成本

-59%

快取規模

9.8B tokens

每做一個動作，再送一次 Hello World

#	動作	依機制預期	實測 credits
0	新對話，第一次送出	原價 + 寫入快取	[__]
1	同對話，再送一次	命中，約一折	[__]
2	開新對話，同模型同工具	固定前綴仍命中	[__]
3	對話中換模型	失效	[__]
4	Configure Tools 關掉一個 MCP	失效	[__]
5	指令檔加一行	失效	[__]
6	/compact 後再送	歷史段失效	[__]

台灣開發者容易忽略的

五個在地情境

繁中寫的常駐指令檔

同內容繁中約英文 2.1 倍 token，又每輪重送。指令檔用英文，對話照說中文

「改用中文提示比較省」

研究實測不成立，有模型反而多 1.28 倍，失敗重試更貴

直接拖 Word、Excel、PDF 規格書

先用 Markdown 轉成 Markdown，只給需要的章節

企業方案的促銷額度已結束

Business、Enterprise 6–8 月每人 \$30、\$70，9 月起回到 \$19、\$39

沒設預算上限

在 GitHub 帳單設定 budgets；Copilot code review 另耗 Actions 分鐘

跨工具 · 機制相同 · 按鈕不同

換了工具，先找這四個地方

要找的	GitHub Copilot (VS Code)	Claude Code
常駐指令檔	copilot-instructions.md、AGENTS.md	CLAUDE.md
看用量	回應上 hover、Agent Debug Logs	/context、/cost
管工具	Configure Tools、mcp.json	/mcp、settings
管歷史	Ctrl+N、/fork、/compact	/clear、/compact

判斷新工具時問三個問題：它每輪幫我送了什麼？我看得到用量嗎？快取怎麼計價？

今天下班前就能做的三件事

1

改指令檔

刪到只剩每次都用得到的規則，加一行「只回 diff 與一句說明」

2

關掉用不到的工具

Configure Tools 只留這個專案真的會用的 MCP server

3

換設定就開新對話

換主題、換模型、換工具組時 Ctrl+N，不在長對話中途改

做完後，用同一句 Hello World 量一次前後差異

帶走這份清單

個人

- ☐ 指令檔精簡，預設簡短輸出
- ☐ 推理強度用預設，難題才調高
- ☐ 問問題用 Ask，動手才用 Agent
- ☐ 只開需要的工具與 MCP
- ☐ 指定檔案與錯誤訊息，不說「看一下」
- ☐ 換設定就開新對話
- ☐ 大型 log 與資料加入 search.exclude

團隊

- ☐ 共用精簡指令檔範本，納入 code review
- ☐ MCP server 白名單
- ☐ 設定預算上限與每月用量報表
- ☐ 盤點 9 月後的方案內含額度
- ☐ KPI 看每個成功任務的成本
- ☐ 用示範 repo 做一次內訓實測

總結

四句話帶回去

問題在 context，不在 prompt

你打的字很少，貴的是每輪重送的指令、工具與歷史

順序：output → input → 快取

先控最貴的輸出，再瘦結構，最後別打斷快取

對的模型做對的事

六到七成的呼叫用小模型就夠，旗艦模型留給難題

量測決定一切

看每個完成任務的成本，不只看 token 數

砍成本沒有品質把關，只是把 API 帳單換成客訴帳單。

APPENDIX

附錄：省 token 技巧詳解

每個技巧的原理、做法、效益數字與來源，以及同一任務四種設定的完整示範

所有技巧，都在拉這三根槓桿

呼叫次數 × (每次的 token 數 × 每個 token 的單價)

少呼叫

精準提示、先規劃、斷路器、交給確定性工具、一次做對不重試

Tip 09 · 22 · 23 · 26

少送

瘦身常駐內容、清掉舊歷史、過濾工具輸出、只取相關片段

Tip 06–08 · 11–17 · 21 · 25

便宜送

快取命中、批次半價、選對模型與推理強度、自架模型

Tip 01–05 · 19 · 20

WHY · 問題在哪

Token 不是花在你的問題，而是花在「每輪重送」

LLM 沒有記憶：每一輪都要重送系統提示、工具定義、指令檔與完整對話歷史。真正的成本大頭是這些常駐內容的累積。

31×

第 30 則訊息的成本可能是第 1 則的 31 倍：歷史越長，每輪重讀越貴

55K

接上 GitHub、Slack、Sentry、Grafana、Splunk 五個 MCP，工具定義就先吃掉約 55,000 token

30–50%

Coding agent 每輪中，系統提示、Skills 與 MCP 清單的開機開銷占比

來源：YouTube 〈18 Claude Code Token Hacks〉摘要、Anthropic（經 WorkOS 整理）、ComputingForGeeks

五層加補充

計價槓桿

- Prompt Caching
- 模型路由
- 推理與輸出預算

上下文工程

- 指令檔瘦身
- /compact 與交接檔
- 先規劃再動手
- 子代理分工

工具與 MCP

- Tool Search

資料壓縮

TOON 格式
語言與 tokenizer

自架與補充

- 劃與執行分離
- Copilot 專屬設定

Prompt Caching：重複的前綴只付一成

原理

把不變的前綴（系統提示、工具定義、長文件）算好的 KV 結果存起來，下次前綴完全一致就以折扣價計費。前綴只要變一個字，後面全部失效。

怎麼做

- 靜態內容放前面、動態內容（使用者輸入）放最後
- Anthropic：cache_control 斷點；OpenAI：自動前綴比對
- Gemini：建立具名 cache 物件，可自訂 TTL
- 系統提示納入版控，避免無謂變動；監看快取命中率

效益

-90%

快取命中的輸入 token 成本；長提示延遲最多 -85%

快取試算：為什麼不是 -90%

呼叫 100 次，每次輸入 10,000 token，其中 8,000 是固定前綴；快取讀取以一折計價。

輸入計費單位	不用快取	使用快取
第 1 次	10,000	10,000
第 2-100 次（每次）	10,000	2,800
合計	1,000,000	287,200

每次 2,800 = 變動的 2,000 + 前綴 8,000 × 0.1。未計快取寫入溢價、輸出與推理 token。

示例試算，非特定模型報價；實際費率請依各家官方價格重算

輸入費用

-71.3%

一折的折扣只作用在可快取的那 80%

模型路由：簡單的事交給便宜的模型

原理

大多數請求不需要旗艦模型。路由器先判斷難度：簡單的派給小模型或內部開源模型，困難的才送旗艦，品質幾乎不變、成本大降。

怎麼做

- RouteLLM (lm-sys)：開源路由框架，可調成本門檻
- Cascade (FrugalGPT)：先跑便宜模型，評分不過才升級
- Coding agent：日常用 Sonnet / Haiku，難題才切 Opus
- 內部：格式化、摘要等雜務派給自架開源模型

效益

-85%

MT Bench 成本，同時保有 GPT-4 約 95% 的表現

指令檔瘦身，流程搬進 Skills 按需載入

原理

CLAUDE.md、AGENTS.md、copilot-instructions.md 每一輪都會重送。Skill 平常只有一行描述常駐，內文在觸發時才載入。

怎麼做

- 只放 80% 的 session 都用得到的規則，控制在 200 行內
- 超過 2KB 的流程（migration、PR review）改寫成 Skill
- 次要文件用 @docs/xxx.md 需要時再引用
- 寫成查找表，不要寫成長篇背景說明

試算

1M

5,000 token 的 CLAUDE.md ×
200 輪對話，未計快取折扣

換任務就 /clear，同任務用 /compact

原理

長對話每輪都重讀全部歷史。把狀態寫進檔案而不是留在對話裡，定期壓縮或重開，讓每輪只帶必要的內容。

怎麼做

- 主動壓縮並給保留提示：/compact Keep: 架構決策、待辦
- 結束前寫 session-handoff.md，新 session 讀它接手
- 用 /context、/usage 監看上下文組成與用量
- 取捨：壓縮會改變前綴、讓快取失效，別每幾輪就壓縮

案例

98.5%

一段 100+ 則訊息的對話裡，花在重讀舊歷史的 token 比例

先規劃再動手，給 AI 一張 repo 地圖

原理

走錯方向的 token 全部白花，還要再花一輪修正。先確認方案，再告訴 AI 該先讀哪些檔，避免它掃描整個專案找實作。

怎麼做

- Claude Code Plan mode、Copilot Plan agent 先產出方案
- docs/repo_map.md：模組職責、入口檔、先讀哪些檔
- 提示寫明「只讀與 token 驗證相關的檔案」
- 直接給檔案路徑或貼出函式，而非「看一下 auth」

效益

**少讀錯檔
少修正迴圈**

錯誤路徑越早攔下，浪費的 token 越少

Tool Search：工具定義延遲載入

原理

每個 MCP 工具的 schema 每輪都進上下文，用不到也照付。改成只放一個搜尋工具，模型需要時再載入那 3-5 個工具的完整定義。

怎麼做

- 工具 10 個以上或定義超過 10K token 就該延遲載入
- Claude Code 在工具定義超過上下文 10% 時自動啟用
- 關掉當下用不到的 MCP server（每個可達上萬 token）
- 工具超過 30-50 個，選擇準確度會下降

效益

-85%

工具定義 token，且工具選擇準確度不降反升

TOON：表格資料不要用 JSON 餵

原理

JSON 每筆資料都重複鍵名、引號與括號。TOON 用縮排取代括號，陣列只宣告一次欄位，之後以類 CSV 的列輸出。

怎麼做

- 儲存與 API 照用 JSON，送進 LLM 前才轉 TOON
- 效益最大：欄位一致的物件陣列（日誌、指標、清單）
- 深層巢狀或欄位不一時，compact JSON 可能更省
- 有 TypeScript、.NET、Rust 等實作與 CLI

效益

30–60%

均一陣列的 token 減少；基準測試準確率 73.9%（JSON 69.7%）

常駐內容用英文寫，對話照樣說中文

原理

多數 tokenizer 以英文為主訓練，同樣內容的中文常需要更多 token。但研究也發現「整段改用中文提示比較省」並不成立。

怎麼做

- 每輪都重送的系統提示、指令檔、工具描述用英文寫
- 一次性的對話與需求說明用中文即可
- 用目標模型的 tokenizer 實測（tiktoken、HF tokenizer）
- Qwen、GLM 等開源模型對中文較友善，要個別量測

實測

2.1×

同一篇文章的繁體中文 token 數
是英文的 2.1 倍（一份多語部落格
實測）

規劃與執行分離，走偏就 Undo

原理

規劃需要大量上下文與強模型；執行只需要計畫本身。把兩者拆到不同會話，執行端乾淨又便宜。走偏時回到乾淨狀態，比在壞掉的上下文上疊修正更省。

怎麼做

- 強模型寫 plan.md 或 issue，再開新會話用較便宜模型執行
- 每個里程碑 commit，當作可回復的檢查點
- 走偏時 Undo 或還原檔案、Fork 對話，不要連續追加修正
- 同一錯誤糾正兩次以上，就寫進 AGENTS.md 或 Skill

Google 的比喻

大象規劃 金魚執行

長上下文會話產出計畫，低 token
的新會話照計畫做

Copilot 專屬設定：模式、範圍、穩定

原理

Agent 模式會帶上工具清單、讀檔、多輪迴圈；簡單問題用不到這些。指令只在相關檔案載入，對話中保持設定不變，快取才不會失效。

怎麼做

- 簡單問題用 Ask 模式，多步驟任務才用 Agent 模式
- instructions 加 applyTo 路徑，只在符合的檔案載入
- 對話中不換模型、推理強度、工具組與順序；要換就開新的
- DOCX、PDF、PPTX 先用 MarkItDown 轉成 Markdown 再給

社群估計

60–90%

簡單問題改用 Ask 模式可省下的
token

別人量到的

公開實驗：同樣的趨勢

實驗	前 → 後	性質
VS Code 官方：開啟 tool search	中位數用戶整個 session token -18%	官方 A/B 實驗
Scott Spence：關掉多餘 MCP	MCP 工具 82.0K → 5.7K token	作者實測
Aetherius：加一份 CLAUDE.md	100K → 70K token，10-12 → 7-8 分鐘	同一 iOS 任務
Zilliz：語意程式碼檢索	73K → 44K token，F1 相同	30 題 SWE-bench
ComputingForGeeks：10 種工具	最佳 -43%；RTK 在此任務 0%	同一提示各跑一次

來源：VS Code Blog 2026/06/17、scottspence.com、aetherius-solutions.com、Zilliz claude-context、computingforgeeks.com

沒有量測就沒有優化：四步驟落地

01

量測

/context、/usage、Copilot 用量報表、API 回應的 cached tokens 欄位，找出前幾名的消耗來源

02

砍常駐

指令檔、MCP 工具、Skills 清單逐項檢視；共用範本放進 repo，全團隊一致

03

開折扣

確認快取命中率，非即時工作改批次，依難度路由到合適模型

04

設護欄

預算告警與用量上限；每週檢視耗用最高的 session，回饋成指令檔與工具調整