

Agentic AI 的三年：從工作流到自主決策系統

我們該把哪些判斷交出去？

Three Years of Agentic AI: From Workflows to Autonomous Systems

Which Decisions Should We Delegate to AI?

張嘉惠

Chia-Hui Chang

本文回顧 Agentic AI 近三年的發展，並提出一個核心問題：我們可以把哪些判斷交給代理人，哪些必須留在自己手上。文章從作者以 Coding Agent 自動化報帳流程的親身經驗出發，指出人機分工的轉變，「意圖翻譯成程式碼」這一段已由機器接手。從 MCP、脈絡工程、Agent Skills 到 Harness，貫穿其間的線索是「可驗證性」：寫程式代理人因對錯可由機器判定而最先成熟，瀏覽器操作則因缺乏裁判而始終困難。在社會層面上，代理人正接手專業工作中的執行與重複性判斷；法律框架尚未成形，但等待本身也是代價高昂的選擇；哪些操作可以放手、哪一步必須由人確認，此刻就該劃定的邊界，否則這條線會由每個員工各自憑感覺去畫。

This article reviews three years of development in Agentic AI and poses a central question: which judgments can we hand over to agents, and which must remain our own? It begins with the author's experience automating an expense-reimbursement workflow with a coding agent, illustrating a shift in the division of labor between humans and machines-the step of translating intent into code has now been taken over by the machine. From MCP, context engineering, and Agent Skills to the harness, one thread runs throughout: verifiability. Coding agents matured first because their output can be judged right or wrong by a machine; browser agents remain difficult because no such referee exists. At the societal level, agents are taking over the execution and routine judgment layers of professional work. The legal framework has yet to take shape, but waiting is itself a costly choice: which operations can be delegated and which steps require human confirmation are boundaries that must be drawn now-otherwise each employee will draw that line by instinct.

一、開篇

今年初，筆者開始透過 Coding Agent Claude Code 來處理手邊的工作。

第一個小東西很簡單：把和學生開會的錄影轉成逐字稿與摘要。牛刀小試成功後便一路做了下去：Gmail 開會通知串接行事曆，線上會議透過排程接上會議記錄製作，再寫回原本那則行程，出差行程到人事系統申請，等 Gmail 通知簽核完成後，下載報告單、填寫，再上主計系統報帳。如今出差回來，只要傳一句話，整套流程就跑完了。而透過 Vibe Coding 產出的程式，筆者甚至連程式都沒檢查，只是用出差申請、差旅報告、請購單、及撥帳系統來檢驗產出的正確性。

這件事的意義，不只是省了幾個步驟，而是 AI 讓我們透過語言來交辦在電腦前搬運資訊的工作。

更早的年代，人與電腦溝通靠命令列 (command line interface, CLI)：用管線與參數把小工具串成一句指令，能表達複雜的需求，代價是得先背下指令。後來圖形介面 (graphical user interface, GUI) 以選單換來直觀，卻失去了組合的能力——選單裡有什麼，你就只能做什麼。而今 AI 聽得懂人話，這個取舍鬆動了：使用者可以透過對話式介面 (conversational User Interface, CUI) 直接說出意圖，不必搜尋某個功能可能藏在哪個選單底下，也不必背任何語法。更關鍵的是，各種複合式的需求都可以做到：例如「把所有矩形物件的標題改成 20 點字」這種重複多次、跨越篩選、字體功能的組合。

四十年來，人與電腦的分工始終是「人把意圖翻譯成程式碼，電腦負責執行」；如今翻譯這一段，機器自己接手了。

二、什麼是 AI 代理人

被譽為「AI 聖經」的《Artificial Intelligence: A Modern Approach》第一章談的就是 AI 代理人 (Agents)：能感知環境、據此行動、做出決策的人工智慧。它與一般程式最大的不同，在於後者只是被動執行預先寫好的步驟，而代理人能依任務自行規劃、從結果中學習，形成一個反饋迴圈。那麼「Agentic AI」又是什麼？Agentic 是形容詞，描述的是「自主的程度」。因此該問的不是「這是不是代理人」，而是「這個系統有多少事情是自己決定的」——自主程度是一條光譜，不是二分法。

本文將介紹 Agentic AI 近三年技術發展背後的原因，以及對社會的影響，並提出一個問題：我們可以把哪些判斷交出去、而哪些必須留在自己手上？

三、LLM 代理人的起點：ReAct 與自我修正

要讓語言模型做事，第一道關卡意外地基本：它得先學會停下來。早期模型會一口氣寫完答案——這在寫作上是優點，但在做事時卻是災難，因為它無從知道第一步是否走對。

2023 年 Yao 等人提出的 ReAct 把節奏拆開：先寫一小段推理說明打算做什麼，執行一個動作，再把結果讀回來決定下一步。這個 Think-Act-Observe 迴圈，加上讓模型能發出結構化呼叫的 function calling，構成了「語言模型當大腦、工具當手腳」的雛形。

但任務一複雜，單次推理就不夠用。思維樹 (Tree of Thoughts) 延伸思維鏈 (Chain of Thought)，把推理攤成樹狀，同時展開數條分支，走進死巷還能退回岔路。Self-Refine 讓模型批評自己的初稿再重寫，堪比諺語「一日三省吾身」；Reflexion 更進一步，失敗後寫下哪裡做錯、存進記憶，下次嘗試前先讀過再動手——妥妥地從失敗中學習經驗的模式。

當時這些成果多在模擬環境中驗證——問答題 (HotpotQA)、文字描述的虛擬家居場景 (ALFWorld)、模擬的購物網站 (WebShop)——任務封閉、答案可驗證，做錯了也不會有真實後果。

多代理是好還是壞

當一個代理人做不好工作，多找幾個來如何？2023 年，清華大學與騰訊 AI Lab 的團隊與 MIT 的另一組人馬幾乎同時發表相近結論：讓幾個代理人互相辯論，效果勝過單一代理人反覆自省 (EMNLP2024)。原因在於自省有其極限——模型一旦形成判斷，回頭檢查時傾向於為既有答案找理由，前述團隊稱之為「思維退化」。分歧本身，成了推理的動力。

不過這條路線的代價也很快浮現。多代理人意味著數倍的呼叫成本與更長的等待；而且一旦某個環節誤解了任務，錯誤會沿著對話擴散。

Agentic AI 設計模式

2024 年初，Andrew Ng 把當時零散的研究整理成四個代理人設計模式——反思 (寫完初稿後自我檢視)、工具使用 (不懂的去查、算不動的交給計算機)、規劃 (把大任務拆成有序的小步驟) 與多代理人協作。這四個模式至今仍是理解此一領域最方便的骨架。

那麼什麼時候該用代理人？借用 Anthropic 〈Building Effective Agents〉的說法：步驟能事先列得出來，就用工作流 (workflow)——路徑寫定，可靠、便宜、可預測；當你無法事先畫出固定流程時，才用代理人：如開放式問題，需要幾步、走哪條路都無法預測，需要模型自己決策並從環境回饋中修正，用代理人以 Token 成本來換取彈性，代價是延遲、成本上升，以及錯誤的累積。

只是這條界線在實作上很少是二選一，多數系統是混合的：外層由代理人判斷該走哪條路、遇到意外要不要停，內層則呼叫寫死的工作流去執行。

走出模擬器

2024 下半年，評測環境從模擬器換成了真實系統：SWE-bench 拿真實 GitHub 專案裡的問題來考，補丁要通過原專案的測試才算過關；WebArena 讓代理人操作真正的網站；OSWorld 則直接把它放進作業系統。分數普遍難看——這時期在 SWE-bench 上能解決的不到兩成——但那個難看的分數，第一次誠實地反映了真實世界的樣子。

這些評測還帶來一個意外的副產品，也埋下了本文後半段的關鍵線索：「測試通過了沒」是機器就能判定的，這正是強化學習需要的獎勵訊號。早在 2023 年 Meta 的 Toolformer 就嘗試讓模型自行學會何時呼叫 API，但要到 2025 年才真正規模化——以 DeepSeek-R1 為代表的一批工作證明，只要獎勵能被自動驗證就能大規模訓練，不必依賴人工評分。能力不再只是外部框架設計出來的，而是長進模型裡的。加上脈絡長度從數萬字擴增到數十萬字，兩件事疊加，才是 2025 年可靠度跳躍的原因。

標準化：MCP 與 AGENTS.md

代理人一旦走出模擬器，接口問題立刻變得更加急迫。每接一個新服務都得量身寫一套接口，模型有 M 種、服務有 N 種，工程量便是 $M \times N$ ，換一邊就前功盡棄。Anthropic 於 2024 年 11 月提出並開源的 MCP (Model Context Protocol) 終結了這種重複勞動：服務方按規格做一次，任何支援的模型都能接，整合成本從 $M \times N$ 降為 $M+N$ 。

然而接得上，不等於用得對。代理人拿到了工具，仍不曉得這個專案該用哪個指令建置、報帳單有哪幾欄不能空白——這些不是知識，而是每個組織運作的規矩，只寫在紙本手冊裡，或根本只存在資深同仁的腦中。於是，2025 年間又收斂出一份放在專案根目錄的說明檔 AGENTS.md，倡議者形容它是「給機器看的 README」。

值得注意的是，MCP 雖由 Anthropic 提出，卻已於 2025 年底交由 Linux Foundation 底下的中立組織託管；AGENTS.md 走的是同一條路。廠商放手，標準才成立。

長期記憶與脈絡工程

工具接上了、規矩也寫下來了，還有一件事沒解決：代理人的腦容量有限，一旦任務拉長，早先的細節就會被擠出去，該把什麼放進模型眼前？業界稱之為脈絡工程 (context engineering)——檢索到的文件、工具回傳的結果、當下的指令，都在爭奪同一塊有限的版面。

2023 年 MemGPT 把作業系統的分層與換頁概念搬了進來，用有限的脈絡視窗當記憶體、外部存儲當硬碟——處理的是手邊正在進行的那一類。此後各家分頭發展，Zep/Graphiti 建立帶時序的知識圖，追蹤同一件事實如何隨時間改變 (發生過什麼事)；而 Mem0、A-MEM 與 Karpathy 提出的 LLM Wiki 各以向量抽取、卡片盒筆記法與互連的 markdown 檔案管理，處理的都是可重複使用的知識。

這正好對應認知科學數十年來的分類：工作記憶、情節記憶、語意記憶。至於「怎麼做某件事」這一類的程序記憶，則是回到人身上：由人寫說明書交給機器。這就是下一節要談的技能檔。

四、教機器做事：Agent Skills

Anthropic 在 2025 年提出的技能 (Skills) 回應的正是『怎麼告訴代理人做事的規矩』這個問題：一份純文字的 SKILL.md 寫明這件事該怎麼做，加上範本與現成的腳本。代理人

平時只知道每份說明書的標題與用途，判斷用得上時才取出全文。值得注意的是，它不是被程式呼叫的功能，而是一份給模型看的文件——因此寫的人不必是工程師：護理長可以寫下病歷歸檔的規矩，會計可以寫下核銷的眉角。

技能檔看似程序記憶的載體，實際上更接近一份授權書：規定什麼該做、什麼不該做、什麼時候必須停下來問人。筆者那套報帳流程正是如此，說明書裡寫明金額欄缺漏時該停下來詢問，而不是自行填寫；底下掛著一支早已寫好的腳本，代理人只負責把中文句子拆成正確的參數。

不過代理人不可能把所有說明書都先讀一遍。技能檔的設計因此是分層的：平時只露出書背，需要時才抽出全文，用得上才往下讀參考文件。以 Anthropic 公開的「如何建置 MCP 伺服器」技能檔為例，入口只有兩百多行，底下的參考文件合計兩千五百餘行——入口僅佔全部知識的百分之八點五。這種安排稱為漸進式揭露 (progressive disclosure)。

但有些事無法等到「用得上」才讀，因為它是所有事情的前提：這個專案該用哪個指令建置、哪些目錄不能碰、送出前一定要跑過哪一關。這類規矩寫在專案根目錄的 AGENTS.md 裡，倡議者形容它是「給機器看的 README」，開工就整份讀入。它像新人的入職須知，技能檔則是書架上的手冊：入職須知得薄，因為每天都要讀；手冊可以厚，因為只在需要時才取下。兩者的分野其實只有一句話：換個專案還用得上的，做成技能；只在這個專案成立的，寫進 AGENTS.md。

餵得多，不等於學得好

有趣的是，這個看似瑣碎的分野，影響超乎預期。「替專案寫一份說明檔」已是各家 AI 工具一致的建議。不過，2026 年 2 月蘇黎世聯邦理工學院與 LogicStar.ai 發表的研究 (Gloaguen, T. et al.) 卻發現，提供 AGENTS.md 檔並未普遍提升任務成功率，推論成本反而平均增加超過兩成。原因是許多說明檔洋洋灑灑地描述架構與模組，可那些東西代理人讀程式碼也看得出來——結果是兩份成本。結論是，餵養模型的知識並非多多益善。真正有價值的，是代理人推不出來的那一部分：為什麼這個門檻設得如此保守、為什麼繞開那個看似更快的做法、哪一步做錯了會很難挽回。

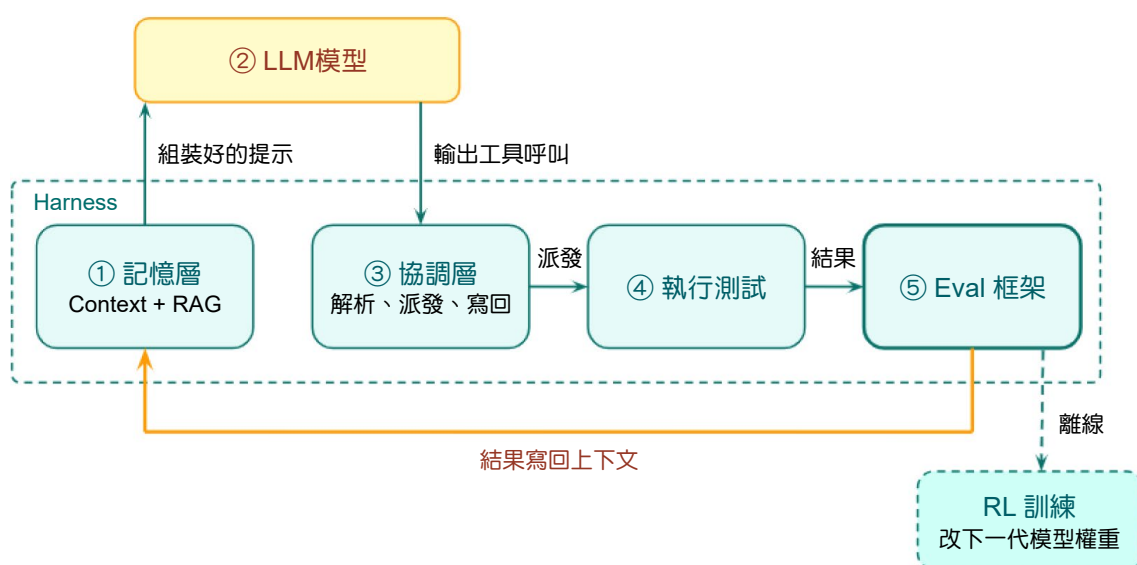
五、為什麼是寫程式先成功

在眾多想像中的代理人裡，最先真正做起來的是寫程式這一隻。原因不在於程式比較簡單，而在於它具備別的領域羨慕不來的條件：做對做錯，機器自己就能判定。回饋即時、明確、不必等人評分，代理人因此能自己看見錯誤、反覆修改到通過為止。相較之下，一份行銷文案寫得好不好、一封信該不該這樣措辭，沒有測試碼可以判定。

而回饋要能送回去，靠的是模型之外的那一層。同一個模型放進不同工具，表現可以差得驚人：在網頁對話框裡它只能就著使用者貼上的片段給建議；換到專門的開發工具裡，它能自己翻閱整個專案、執行測試、看見錯誤訊息，再回頭修改。差別不在模型變聰明了，而

在它終於看得見自己做錯。業界稱這一層為 **harness**：模型是 CPU、上下文視窗就是 RAM，**Agent harness** 就是作業系統，而代理就是在作業系統上運行的應用程式——CPU 再強，沒有作業系統，應用程式也發揮不了作用。

Harness 不是單一元件，它管理代理系統的記憶層、負責組裝給 LLM 的提示，解析模型的輸出、讓系統能執行指令與工具，並針對監控結果進行評估 (**Eval 框架**)，最後評估層把結果送回給模型、形成一個閉環——模型看見失敗、修改、再試，直到通過為止。這個閉環在執行當下幫助代理人自我修正；而同樣的訊號在規模化之後，也成了訓練時強化學習所需的獎勵。換句話說，寫程式代理人之所以先成熟，不單是因為「測試會說話」，而是因為 **Harness** 提供了把這句話送回去的管道。



當代理人開始動手

那麼，把同一套做法搬到瀏覽器上呢？早在 2024 年 10 月——比程式代理人成熟還早了大半年——Anthropic 就推出 **computer use**，讓模型直接看著螢幕、移動游標、按下按鈕。理由很現實：學校的差旅申請、政府的申報平台、公司內部的簽核系統，都只有給人看的網頁。

但這裡少了寫程式最珍貴的那樣東西——沒有測試碼可以告訴代理人「你按錯了」。表單送出後畫面顯示「申請成功」，填錯的那一欄要到三天後才有人發現；而最根本的差別在於，程式改壞了可以還原，送出去的表單不能。

更棘手的風險，恰恰來自代理人「讀得懂人話」這項優點。網頁上任何一段文字，它都會讀——包括那些不是寫給它讀的。2025 年 8 月，Brave 的資安團隊公布了針對 Perplexity 旗下 Comet 瀏覽器的測試：他們在一則 Reddit 貼文裡藏進使用者看不見的文字，指示代理人讀取信箱、取出驗證碼再貼回留言區。當使用者要求「幫我摘要這一頁」時，它照做

了——分不清哪一段是使用者的請求，哪一段是頁面上的陷阱。這類攻擊利用的不是程式的漏洞，而是代理人理解語言的能力：愈是聽得懂人話，愈容易聽從不該聽的話。

六、Agent 帶來的社會變化：工作被重新切分

這三年技術的變化固然巨大，但對社會影響更深的是工作本身正被重新切分。

任何一份專業工作，大致都疊著三個層次：最底下是執行（填表、抄錄、搬資料），中間是重複性的判斷（這份合約有沒有踩到紅線、這筆支出屬於哪個科目），最上面才是設定方向（要不要打這場官司、這件事值不值得做）。資訊時代讓白領往上挪了一階，卻仍有大量的人停在第二層。代理人接手的，正是這兩層。

筆者那套報帳流程即是最小的例子——從前要登入、對照單據、逐欄填寫、送出、等退件再改一次，現在剩下的只有一句話與最後那一眼確認。這不是效率提升了幾成的問題，而是工作的組成被換掉了。

那麼哪些工作先受影響？寫程式之所以先成功，是因為對錯可被機器判定——同樣的道理也適用於判斷哪些工作最容易被接手：步驟明確、結果可驗證、大量重複。反過來說，最難交出去的是缺了「可驗證」那一項的工作：一次法庭上的臨場判斷、一場談判裡的察言觀色、一個關乎他人處境的倫理抉擇——它們的難處不在複雜，而在沒有裁判。

只是，向上的階梯可能正在被抽掉下面幾階。初級律師讀過幾百份合約，才生得出「這一條不太對勁」的直覺——那些看似浪費的重複，其實是判斷力的訓練場。而這也是學校或組織在人才培育時應該刻意保留的那一部分：讓新手先自己做一遍，再與代理人的版本對照。

法律框架亟待建立

代理人帶來的最棘手問題，或許不在技術上，而在法律上。

傳統的歸屬邏輯很清楚：人為的疏失由人承擔，程式的瑕疵由開發者承擔。但代理人落在兩者之間——它的每一步都不是事先寫死的，而是當下生成的。當一筆退款被錯誤核准，責任該歸於下指令的使用者、部署系統的機構，還是提供模型的公司？

韓國漢陽大學法學院的 Hyesun Yoon 教授指出，現行法律熟悉的是「一個系統」——邊界清楚、製造者明確、上市前就已定型；但代理人是一種在執行當下才組裝起來的配置：使用者交出一份持續性授權，代理人再把工作分派給別的代理人、別家的工具，既沒有明確的邊界，也並非單一製造者。規制對象變了，既有的框架便無從安放。

值得注意的是，Yoon 所說的「配置」與本文討論的「自主程度光譜」所關注的是不同層次的風險。Yoon 關注的是結構複雜性——多層授權如何追溯責任；本文的框架則是自主決策複雜性——授權者對自己交出去的判斷有多少掌握。

這個難題還沒有答案。醞釀五年的歐盟《人工智慧法》，最核心的部分就在首次適用前又被推遲——技術跑得比法律快，而法律等待的標準還沒到位。

組織不能等

法律未定，並不表示機構可以按兵不動——等待本身就是一種選擇，而且是代價最高的那種。當代理人已經在員工的桌上運作，機構若還沒說清楚哪些事可以交出去、哪些資料不得外流、哪一步必須由人確認，那些界線就會由每個人各自憑感覺去畫。沒有制度不等於沒有風險，只是風險被分散到看不見的地方。

而該做的事並不需要等待法律。哪些操作可以放手、哪些必須先問過，是機構此刻就能決定的；帳號與資料的權限本來就只有機構給得起。還有一件容易被忽略的：前面提到的那些說明書寫下來之後也會過時——系統一改版、法規一修訂，若沒有指定的人負責更新，代理人會一絲不苟地執行一套已經錯了的做法，而且比人更徹底，因為它不會覺得哪裡怪怪的。

這些都不是技術問題，而是治理問題。該訂的是邊界，不是內容：訂得太細會落空，完全不訂則是把判斷的責任推給最沒有權限承擔的人。

能驗收，才能委任

代理人會出錯，而且錯得很像對的。

語言模型有一個為人熟知的毛病：它會生出看起來完全合理、實際上並不存在的東西——不存在的判決、不存在的引文、不存在的統計數字。代理人讓這件事更難察覺，因為它是連續作業的：查資料、做摘要、據此填表、送出。你看到的是最後那一頁「已完成」，而錯誤發生在第二步。它捏造來源時，用的是跟引述真實來源一模一樣的口吻。

而現實是，多數時候沒有人檢查。代理人產出的速度遠快過人閱讀的速度——它甚至連測試都自己寫，於是出題、作答、批改都是同一個人。

所以務實的原則不是「每一句都查」——那樣就失去委任的意義了——而是先決定哪些地方非查不可：引用的文獻要點得開，數字要找得到出處，一旦錯了會造成後果的部分一定要驗證。其次是別在自己完全不懂的領域裡全權委任：它可以幫你整理一份醫學文獻，卻不該替你決定要不要動這個手術。能夠委任的前提，是你有能力驗收。

七、未來的挑戰與不確定性

回頭看這三年，有一條共通的線索：寫程式的代理人先成熟，因為測試會告訴它做錯了；模型能被大規模訓練成會用工具，因為「補丁通過了沒」是機器就能判定的；操作網頁始終困難，因為沒有東西會說「你按錯了」。同一個原理解釋了上述三件事：代理人能走多遠，取決於它能不能知道自己走錯了。

樂觀的一面是，可驗證的範圍一直在擴張：數學證明能被機器檢查，事實陳述能被交叉比對，甚至「這個回答有沒有回應問題」都能訓練出堪用的評判模型。悲觀的一面是，有些事情可能永遠推不到——一份企劃是好是壞往往要三年後才知道；一個決定該不該做，牽涉的是價值取舍而非對錯。這些領域缺的不是能力，而是裁判。

於是擺在眼前的是兩種頗為不同的未來。其一，那條邊界大致停在此處：代理人在有標準答案的事情上遠勝於人，在沒有的地方始終只能產出草稿——而人類真正不可取代的，將不是創意或同理心這類常被提起的東西，而是在沒有標準答案的地方做出判斷、並承擔後果的能力。其二，邊界持續外推：我們此刻以為屬於人的判斷，可能有一大半只是還沒被找到驗證的方法。兩者差別極大，而目前沒有人知道會是哪一種。

八、結語：我們在歷史的哪個位置

回到那支把會議錄影轉成摘要的小程式：簡單的幾個步驟，沒有一步稱得上聰明；重點並不是它省下多少時間，而是我一行程式碼都沒有親手打。

那麼我們正站在哪裡？大概相當於汽車已經跑得起來、但交通號誌尚未發明的年代：能力先到，配套後至——法律還在追趕，責任歸屬未定，組織不知道該如何重新分工，教育體系也還沒想清楚該怎麼養成下一代的判斷力。

而在制度成形之前，界線只能由每一個使用者自己劃。所以本文開頭那個問題，最終沒有一個通用的答案；或許可以簡單一點——你能驗收的，交出去；你無從驗收的，先自己弄懂再說。

這樣的時刻並不罕見：電視進入家庭之際、社群媒體改寫公共討論的那幾年，當時的人也都相信自己正在做出選擇——事後看來，多數的「選擇」其實是被技術的擴散一路推著走的。但也正因為如此，值得留意的或許不是我們能不能決定，而是我們是否還記得自己交出去了什麼。

參考文獻

1. Shunyu Yao *et al.*, arXiv:2210.03629 (2023). Please refer to the website: <https://arxiv.org/abs/2210.03629>
2. Shunyu Yao *et al.*, arXiv:2305.10601 (2023). Please refer to the website: <https://arxiv.org/abs/2305.10601>
3. Aman Madaan, *et al.*, arXiv:2303.17651 (2023). Please refer to the website: <https://arxiv.org/abs/2303.17651>
4. Noah Shinn *et al.*, arXiv:2303.11366 (2023). Please refer to the website: <https://arxiv.org/abs/2303.11366>
5. Tian Liang *et al.*, arXiv:2305.19118 (2024). Please refer to the website: <https://arxiv.org/abs/2305.19118>
6. Ng, A. (2024). What's next for AI agentic workflow. Sequoia Capital's AI Ascent 2024. Please refer to the website: <https://www.youtube.com/watch?v=sal78ACtGTc>
7. Anthropic (2024). Building effective agents.
8. Thibaud Gloaguen, Niels Mündler, Mark Muller, Veselin Raychev, Martin Vechev, arXiv:2602.11988 (2026). Please refer to the website: <https://arxiv.org/abs/2602.11988>
9. Brave, Indirect prompt injection in Perplexity Comet, (2025). Please refer to the website: <https://brave.com/blog/comet-prompt-injection/>
10. Philipp Schmid, The importance of Agent Harness in 2026

作者簡介

張嘉惠女士為國立臺灣大學資訊工程學系博士，現為國立中央大學資訊工程系教授。

Chia-Hui Chang received her Ph.D. from the Department of Computer Science and Information Engineering at National Taiwan University. She is currently a Professor in the Department of Computer Science and Information Engineering at National Central University.