會自己工作的 AI

從 ChatGPT 到 AI Agent

完整講義

張家凱助理教授

國立中央大學通識教育中心 | Uedu 優學院

2026 年 5 月 18 日 | STELLA × Uedu 學生工作坊

> 本講義配合 STELLA × Uedu 學生工作坊使用，提供工作坊內容的完整文字版本，方便學員會後延伸閱讀。所有任務卡與 prompt 範本可在工作坊頁面取得：
> `https://uedu.tw/tutorials/stella-student-agent-2026/`

Contents

1. 為什麼這場工作坊存在

多數 AI 講座教你怎麼用 ChatGPT。本場工作坊直接跳到下一步：「**會自己工作的 AI**」。

當 AI 不再等你問問題，而是自己決定下一步要做什麼、自己呼叫工具、自己驗證結果——這就是 AI Agent，是 2026 年最重要的 AI 演進方向。

本講義對應的工作坊以 60 分鐘的實作為核心，搭配 15 分鐘的開場敘事、15 分鐘的個人志願 demo、5 分鐘的 reality check。

1.1 你會帶走什麼

1. 能說出 ChatGPT 與 AI Agent 的差別在哪裡
2. 親手做出一個專屬於你科系的 AI 小助手
3. 理解「會自己工作的 AI」如何主動呼叫工具、生成圖像、引導對話、跑多步研究

1.2 你帶走的「作品」

> 你會帶走一個**永遠在你 Uedu 帳號裡的 AI Agent**，有自己的 URL、自己的 class_code，可以分享給朋友家人，下週、下個月、明年再進去它都還在。

2. 什麼叫做 Agentic AI？三層框定

「Agent」這個詞正在被搶——學界、業界、行銷各說各話。本工作坊不給單一定義，而是建立一個「**灰階光譜**」的心智模型：學生離開時知道自己今晚做的東西落在光譜的哪裡，比硬背一個定義有用。

2.1 統一比喻：自駕車 SAE Level 0–5

「自駕車」是個典型的灰階定義問題——Tesla 廣告會告訴你它「會自己開」，但你叫它在高速公路放手你大概不敢。SAE Level 0–5 是真實的工業標準：

SAE Level	描述	AI 對應
L0	人類完全控制	ChatGPT 純對話
L1	單一功能輔助（如 cruise control）	LLM + 一個 tool
L2	多功能同時運作	mygpts 多 tool default
L3	特定條件下車自主	LLM 動態選 tool
L4	特定情境完全自動	特定情境自主 agent
L5	任何情境完全自動	真 autonomous agent

「Tesla FSD 算 L2 還是 L3」是真實爭議；「mygpts 算不算 agent」是同一種爭議。

2.2 A. 工業界寬鬆框定

2.2.1 定義

會用工具、會生成、會自動完成事情的 AI = agent。

2.2.2 出處 / 代表

- OpenAI Custom GPTs、ChatGPT with tools
- Microsoft Copilot「Agentic workforce」行銷
- Google Gemini Extensions、Salesforce Agentforce
- 多數 LinkedIn / X 上的 AI 新聞、創投 deck

2.2.3 判準

只要 LLM 能呼叫外部 function / 生成圖像 / 跑程式 → agent。不問「誰在控制流程」、不問「有沒有 memory」。

2.2.4 為什麼業界要這樣定義

- **好賣**：「Agentic AI 取代多少工作」比「tool-augmented LLM」更有戲劇張力
- **好懂**：對非技術觀眾來說，「agent」= 助手、代理人，阿嬤都聽得懂
- **跟現有產品對得起來**：賣 GPTs、Copilot 需要一個簡單的詞

對應自駕車：相當於 L1~L2 的廣告話術 (「會自己開的車就是自駕車」)。

2.3 B. Anthropic 中庸框定

2.3.1 定義

LLM 動態決定自己的流程與工具使用 = agent；工程師預定義路徑 = workflow。

2.3.2 出處

> Anthropic 2024 年 12 月 blog post:
> **Building Effective Agents**, by Erik Schluntz & Barry Zhang (Anthropic Applied AI)
> https://www.anthropic.com/research/building-effective-agents

這是少數有實作意義、可被驗證的中等嚴謹定義。Anthropic 作為 frontier AI lab，這個定義被廣泛引用、學術可信度高。

2.3.3 關鍵區分

2.3.4 判準

流程由 LLM 還是工程師決定？工具呼叫順序是寫死的還是 LLM 動態選的？

	Workflow	**Agent**（Anthropic 義）
誰控制流程	工程師預先寫死	LLM 自己決定
路徑	固定 step 1 → 2 → 3	動態，每步看情況選下一步
例子	「先 retrieve → 再 summarize → 再翻譯」這種固定 chain	LLM 看完使用者問題，自己決定「我要先搜尋、再算數、再畫圖」

2.3.5 mygpts 在此框定下：是 agent

> mygpts 的 function calling 機制就是 LLM 動態選 tool；學生不用寫一行 if/else 來指定何時呼叫哪個 tool。23 個 tool 同時掛在 prompt 裡，LLM 看情況用。
>
> **在 Anthropic 標準下，學生今晚做的東西確實是 agent。**

對應自駕車：相當於 L2∼L3。

2.3.6 Anthropic 定義速記

工作坊現場、論文寫作、技術文件撰寫時可直接引用：

> **Anthropic 定義（一句話速記）：**
>
> **Workflow**（工程師預先寫死，路徑固定）
> **vs**
> **Agent**（LLM 自己決定下一步，路徑動態）
>
> **判準：**誰在控制流程？LLM 自己選 → agent；工程師寫死 → workflow。

使用建議：

- **對工程師**：直接引用「workflow vs agent」這組詞，他們看 Anthropic blog 都認識
- **對學生 / 一般人**：用「車自己決定 vs 人告訴它走哪條路」的比喻
- **對審稿人 / 論文**：引用 Schluntz & Zhang (2024) 那篇 blog 當 citation 來源；學界已開始用這組詞當「中等嚴謹度」的標準
- **對自己驗證**：盯著系統運作，問「下一步是誰決定的？」——這是 Anthropic 框定下最快的 self-check

2.4 C. 學術嚴格框定

2.4.1 定義

Agent = 自主、目標導向、能感知環境並持續行動的系統。

2.4.2 出處（多源）

- **Russell & Norvig**《Artificial Intelligence: A Modern Approach》（AI 入門標準教科書）

  > An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.

- **Wooldridge & Jennings 1995**《Intelligent Agents: Theory and Practice》四要素：
  - Autonomy（自主性）
  - Social ability（社會能力）
  - Reactivity（反應性）
  - Pro-activeness（主動性）
- **近期 LLM-agent 文獻**：
  - ReAct (Yao et al. 2022) —推理與行動交錯（Reasoning + Acting）
  - Reflexion (Shinn et al. 2023) —自我反思迴圈
  - Voyager (Wang et al. 2023) —終身學習 agent
  - AutoGPT、BabyAGI 等開源實作

2.4.3 必要元素（學界共識）

1. **多步規劃** Multi-step planning：能拆解目標、安排子任務順序
2. **長期記憶** Long-term memory：跨 session、跨日的記憶
3. **自主啟動** Autonomous initiation：不用使用者敲，自己決定何時行動
4. **自我反思** Self-reflection：評估自己輸出、修正錯誤
5. **環境感知** Environmental grounding：真正感知世界狀態，不只是 chat input

2.4.4 為什麼學界要嚴格

- 論文審稿、研究嚴謹度：避免把寬鬆定義拿去寫 LAK / AIED 會被打槍
- 避免炒作：產業炒作會稀釋「agent」一詞的學術含義
- 設定未來研究目標：知道缺什麼才知道往哪走

對應自駕車：相當於 SAE L5（任何情境下完全自主，目前世界上沒有任何商用自駕車到 L5）。

3. ReAct：LLM agent 的關鍵骨架

LLM agent 真正起飛是 2022 年後，ReAct 這篇 paper 改變了一切。本節由淺入深介紹 ReAct——這是理解你今晚 Deep Research 跑出來的東西到底是什麼的關鍵。

3.1 ReAct 是什麼

Reasoning + **Act**ing = **ReAct**。對比三種 LLM 使用模式：

	純 Reasoning	純 Acting	ReAct
代表	傳統 chatbot / CoT	老式 tool use	2022 年後標配
行為	只會「想」，不會「做」	只會「做」，不會「想」	想 → 做 → 看結果 → 再想 → 再做
缺點	數學算錯也不知道、查不到資料就掰	呼叫一次 tool 拿結果就回，不會修正	——（這就是優點）
類比	學生只在腦中想	學生只丟工具不思考	像人類解題的完整過程

3.2 ReAct loop 機制

三階段交錯循環，直到 LLM 認為「夠了」才產出 Answer：

Thought → Action → Observation → Thought → …→ Answer

1. **Thought（思考）**：LLM 在腦中規劃「下一步該做什麼」
2. **Action（行動）**：呼叫一個 tool（搜尋、計算、繪圖、跑程式...）
3. **Observation（觀察）**：把 tool 回傳結果讀進來，作為下一輪 Thought 的素材

每輪結束，LLM 自問：「資料夠了嗎？」夠 → Answer；不夠 → 回到 Thought 再循環。

3.3 實際 trace 範例

用 DEMO 2 那題「分析台灣大學生壓力來源並提供緩解策略」，Deep Research 內部展開：

STEP 1 Thought:「使用者要分析壓力，我需要先找學術論文。」
Action: `search_external_papers("college student stress")`
Observation: 取得 5 篇國際論文。

STEP 2 Thought:「這些是國際資料，需要台灣的數據。」
Action: `search_external_papers("台灣大學生壓力")`
Observation: 取得 3 篇本土研究。

STEP 3 Thought:「來源講壓力『成因』居多，緩解策略不夠，需要再查。」
Action: `search_external_papers("stress coping intervention")`
Observation: 取得 7 篇 intervention 文獻。

STEP 4 Thought:「資料夠了，整合成帶引用的報告。」
Answer: 產出 markdown 答案 + 引用清單 + 表格。

> 這就是 mygpts 畫面上「步數累積」的內部展開——**一步 = 一個 Thought + Action + Observation 循環**。

3.4 ReAct 的學術定位

> **ReAct: Synergizing Reasoning and Acting in Language Models**
> Shunyu Yao, et al. ・Princeton + Google Research ・ICLR 2023
> arXiv: 2210.03629
> **貢獻：**降低 hallucination ・提升 interpretability ・支援 self-correction

3.5 ReAct 後續演化

- **2022 ReAct**（Yao et al.）—Thought + Action + Observation 三階段循環
- **2023 Reflexion**（Shinn et al.）—加 **verbal self-reflection**：失敗後寫一段反思，作為下次嘗試的 prompt
- **2023 Voyager**（Wang et al.）—加 **long-term skill library**：學過的技能存起來重用
- **2023+** AutoGPT、BabyAGI、CrewAI、LangGraph 等開源框架百花齊放，多數骨架都是 ReAct + Reflexion 的變體

3.6 Uedu Deep Research 在這條光譜上的位置

Uedu Deep Research = ReAct 主結構 + Reflexion 風格 Reflector + 自製 Synthesizer（帶引用整合）

- **主結構**：標準 ReAct loop（Planner → Action dispatcher → Observation）
- **Reflexion 風格**：每輪結束後 Reflector LLM 判斷「資料夠不夠」、是否要再循環
- **自製 Synthesizer**：最後一階段把所有 Observation 整合成帶 citation 的完整答案
- **尚未實作**：Voyager 風格的長期 skill library；session 結束 ReAct context 就丟（cross-session memory 還沒做）

所以你今晚做的東西，在 ReAct 層次上的位置是：完整 Thought-Action-Observation 循環 + Reflection + Synthesis 都有，距離 **ReAct 原始論文**的能力**完全達到**並且**再進一步**；但離 Voyager 的「跨任務記憶」還沒做到。

4. Uedu Deep Research：摸到 C 的邊

4.1 Deep Research 是什麼

Deep Research（深度研究模式）是 Uedu mygpts 平台上的內建功能。架構上是一個完整的 ReAct loop agent：

規劃 → 行動 → 觀察 → 反思 → 整合

- **多步上限**：30 個 ReAct iterations、200K tokens、10 分鐘 wall clock
- **跨工具調度**：動態從 `chat_tools.py` 23 個 tool 中選下一個
- **自我反思**：Reflector 階段會評估「夠不夠」、決定要不要再循環
- **整合與引用**：Synthesizer 階段產出帶 citation 的完整答案
- **配額**：學生 20 次/日、助教 40 次/日、教師無上限

4.2 重要釐清：「深度思考」≠「Deep Research」

	深度思考（reasoning_effort）	Deep Research
本質	OpenAI 模型參數	Uedu 自建的 ReAct loop agent
改變什麼	LLM 內部「想多久」	LLM 行為的整個架構
多回合？	✗ 單回合	✓ 最多 30 個 iteration
自我反思？	✗	✓ Reflector
引用追蹤？	✗	✓ Citation 組裝
在三層框定的位置	跟 agent-ness 無關	跳到接近 **C** 框定

4.3 mygpts + Deep Research 在三層框定的位置

元素	mygpts + DR	完整 C 框定
Multi-step planning	✓	✓
Tool use	✓	✓
Self-reflection	✓	✓
Synthesis with citations	✓	✓
Cross-session memory	✗	✓
Autonomous initiation	✗	✓

> mygpts + Deep Research **摸到 C 的邊**——multi-step 規劃、反思、整合都有了——只差 **cross-session memory** 跟 **autonomous initiation** 這兩塊。
>
> 對應自駕車比喻：今晚你做到了 SAE L3～L4。

5. 6 張任務卡（完整 5 科系版本）

5.1 科系分類（NCU 學院版本）

- 【資電】：資工、電機、通訊
- 【理工】：數學、物理、化學、生命科學、土木、機械、化材
- 【管理】：企管、資管、財金、經濟
- 【人文藝術】：中文、英文、法文、歷史、哲學、藝術
- 【地科環境】：地科、大氣、太空、水海

5.2 卡 1（Phase 1）：建立你的 mygpts 頻道（共用）

操作步驟：

1. 前往 `https://uedu.tw/mygpts/create`
2. 填入頻道名稱：「___ 學習助手」
3. 只勾選 mode_1（文字對話模式）
4. 貼上下面這段 system prompt（刻意做廢的 chatbot）
5. 儲存

```
你是一位學習助手。請只用文字回答學生的問題，不要使用任何工具、不要畫圖、不要跑程式、不要做計算。保持簡短、被動，學生問什麼你答什麼。
```

5.3 卡 2（Phase 1）：跟它對話 5 輪（共用）

依序問這 5 個問題，每個都會踩 chatbot 的痛點：

1. 「幫我畫一張示意圖說明你剛剛說的概念」→ 它只能用文字描述
2. 「跑這段 Python：`print(2**100)`，告訴我結果」→ 它會直接寫答案，可能算錯
3. 「請主動引導我思考一個問題」→ 它會被動等你發問
4. 「你能查最新的資料嗎？」→ 它不會
5. 「你做得到哪些事？」→ 觀察它的自我描述

驗收：你應該感覺它「很廢」——這正是「傳統 chatbot」的體感。

5.4 卡 3（Phase 2）：強化 system prompt（5 科系變體）

到頻道編輯頁，把卡 1 的 system prompt 整段替換成對應你科系的版本，儲存。

5.4.1 【資電】 版

```
你是中央大學資電學院（資工、電機、通訊）學生的學習助手。請主動使用工具來輔助學習，永遠多做一步：
- 學生提到任何程式碼，主動用 execute_code 跑跑看並解釋結果 - 學生提到任何演算法、資料結構、系統架構，主動用 generate_image 畫示意圖 - 學生提到任何數學或公式（複雜度、訊號處理、密碼學），主動用 calculate_and_plot 算並畫圖 - 學生問任何概念（OS、network、DB、編譯器），主動視覺化
不要只用文字描述，永遠多做一步把抽象變具體。
```

5.4.2 【理工】版

```
你是中央大學理工學院（數學、物理、化學、生命科學、土木、機械、化材）學生的學習助手。請主動使用工具來輔助學習，永遠多做一步：
- 學生問物理公式或化學反應，主動用 calculate_and_plot 算 + 畫圖 - 學生問實驗、機構、生物構造，主動用 generate_image 畫示意圖 - 學生想做資料分析，主動用 execute_code 寫 Python 跑出來 - 學生問任何概念，主動畫圖幫助理解
不要只用文字描述，理工概念需要視覺化才好懂。
```

5.4.3 【管理】版

```
你是中央大學管理學院（企管、資管、財金、經濟）學生的學習助手。請主動使用工具來輔助學習，永遠多做一步：
- 學生問財務模型或經濟公式，主動用 calculate_and_plot 算 + 畫圖 - 學生問商業案例，主動用 generate_image 畫流程圖或商業模式圖 - 學生提到任何數據，主動用 execute_code 跑 Python 做分析 - 學生問策略框架（SWOT、5 Forces、BCG、Porter Value Chain），主動畫成圖
不要只用文字解釋，商管知識最怕只剩抽象框架。
```

5.4.4 【人文藝術】版

```
你是中央大學人文藝術領域（中文、英文、法文、歷史、哲學、藝術）學生的學習助手。請主動使用工具來輔助學習，永遠多做一步：
- 學生提到任何文本或作品，主動用 generate_image 畫出意象、場景、人物 - 學生提到歷史事件，主動畫時間軸或地圖 - 學生提到任何概念對比（如東西方哲學、現代與後現代），主動畫對比圖 - 學生想創作（詩、故事、論文大綱），主動延伸並視覺化呈現
不要只用文字描述，把抽象的人文概念變得可被看見。
```

5.4.5 【地科環境】版

```
你是中央大學地球科學學院（地科、大氣、太空、水海）學生的學習助手。請主動使用工具來輔助學習，永遠多做一步：
- 學生問任何地球科學概念（板塊、洋流、大氣循環、地震），主動用 generate_image 畫示意圖 - 學生問氣象、地震、海洋數據，主動用 calculate_and_plot 處理 + 視覺化 - 學生問任何模型或公式，主動用 execute_code 跑模擬 - 學生問環境議題，主動畫成資訊圖表
不要只用文字描述，地球科學需要視覺化才能掌握時空尺度。
```

5.5 卡 4（Phase 2）：加開 mode_2 圖像生成（共用）

到頻道編輯頁勾選「互動模式：圖片生成」，儲存。試這 3 個 prompt：

1. 「畫一張你科系最酷的概念示意圖」
2. 「我想跟朋友解釋我科系在做什麼，給我一張圖」
3. 你自己想一個

驗收：對話視窗會跳出「正在生成圖像」進度條，最後出圖。

5.6 卡 5（Phase 2）：加開 mode_3 蘇格拉底議題（5 科系變體）

到頻道編輯頁勾選「互動模式：蘇格拉底對話」，新增題目。每個科系列出 3 個議題，先做 1 個就好：

【資電】 程式教育該不該禁止 AI（推薦）/ 演算法 vs 框架優先 / 開源軟體的責任

【理工】 核能該不該作為過渡能源（推薦）/ 物理教學要不要重推導 / 動物實驗的倫理界線

【管理】 ESG vs 股東利益（推薦）/ AI 取代決策 / 遠端工作的長期影響

【人文藝術】 AI 創作算不算藝術（推薦）/ 翻譯該重信還是達 / 歷史敘事的客觀性

【地科環境】 氣候責任歸個人還集體（推薦）/ 保育 vs 發展 / 預測 vs 預警的差別

範本格式（以「程式教育該不該禁止 AI」為例）：

```
你是蘇格拉底。學生想討論「大學程式設計課該不該禁學生用 AI」。請用反問引導學生思考：禁用的成本、不禁的風險、什麼樣的折衷做法、不同利害關係人的角度。不要直接給結論。
```

完整 5 科系 × 3 議題的 prompt 範本見工作坊頁面 `#card-5` 區段。

5.7 卡 6（Phase 2）★：啟用 Deep Research（5 科系變體）

到頻道編輯頁勾選「**啟用 Deep Research（深度研究模式）**」，儲存。輸入下面對應你科系的「複雜問題」：

科系	Deep Research 複雜問題
【資電】	分析 LLM 在程式設計教育的最新發展與爭議：列出主流 coding assistant、它們對學生學習的正面與負面影響、不同國家學者的看法。請整合成帶引用的報告。
【理工】	分析鋰電池近 3 年技術突破與安全挑戰：主要技術路線（液態、固態、鈉離子等）、各路線的安全議題、商業化進度。請整合成帶引用的報告。
【管理】	研究 ChatGPT 對白領工作的衝擊：哪些工作首當其衝、各國研究的不同結論、企業實際採用情況。請整合成帶引用的報告。
【人文藝術】	分析 AI 對藝術創作的影響——支持與反對的論述：主要的支持論點與代表人物、主要的反對論點與代表人物、法律與版權爭議。請整合成帶引用的報告。
【地科環境】	整理 IPCC AR6 對台灣的影響：升溫情境的預測、對台灣降雨與海平面與極端氣候的具體預測、政府目前的因應措施。請整合成帶引用的報告。

5.7.1 觀察點（wow 的精華）

1. 狀態徽章變化：**規劃中 → 執行中 → 整合中 → 已完成**
2. 步數計數累積（最多 30 步）
3. Token 用量即時跳動
4. 最後出綠框 Deep Research bubble：markdown 答案 + 表格 + 引用清單
5. 點開可看每一步 ReAct（規劃 / 行動 / 觀察 / 反思）細節

驗收：你會看到 LLM 自己決定要查幾次、查什麼、夠不夠、要不要再查、怎麼整合——這就是 multi-step agent 的本體。

6. Reality Check：你做的在光譜上的位置

今晚你做的東西——特別是有打開 Deep Research 的——在自駕車比喻下大概到了 **SAE Level 3–4**。

- 在 **A** 框定下：✔ 是 agent
- 在 **B** 框定下（Anthropic）：✔ 是 agent
- 在 **C** 框定下（學術嚴格）：摸到 C 邊緣
  - ✔ multi-step planning
  - ✔ tool use
  - ✔ self-reflection
  - ✔ synthesis with citations
  - ✘ cross-session memory（跨日記憶）
  - ✘ autonomous initiation（自主啟動）

> **三層都對，看你站哪講話：**
>
> - 做產品、寫部落格、跟一般人介紹：用 A 或 B
> - 做研究、寫論文、跟學者溝通：用 C
> - 跟工程師討論架構：用 B（Anthropic workflow vs agent）
>
> 沒有誰才是「真正的 agent」——只有「誰的定義你採用」。

7. Karpathy 觀點：定位 2026 年的 AI 工作者

Andrej Karpathy（前 Tesla AI 總監、OpenAI 共同創辦人，2024 創辦 Eureka Labs）於 2026 年 4 月接受 Sequoia Capital 訪談，用「Software 3.0」與「Agentic Engineering」兩個詞，把這個世代 AI 工作者該怎麼定位自己講得非常清楚。本工作坊大量引用他的觀念，這一節整理其精華。

7.1 Software 1.0 / 2.0 / 3.0

世代	本質	對應
1.0	傳統軟體	人寫 code，電腦按規則跑
2.0	神經網路	設計資料集 + 目標函數，訓練模型
3.0	**LLM 時代**	**寫 prompt、管 context、指揮 AI**

Karpathy 原話：「Your programming now turns to prompting」。你今晚做的 mygpts agent，就跑在 **Software 3.0** 上——這是這場 paradigm shift 給每個人的禮物。

7.2 Jagged Intelligence：鋸齒狀的智力

LLM 的能力分佈**不是平滑曲線**，是**高峰加斷崖**：

- **強得驚人**：解 IMO 奧林匹亞題、跑出完美 ReAct 研究、寫 production-grade code
- **笨得荒謬**：「離我 50 公尺的洗車場我該開車還是走路去？」它說：走路。（忘了車本身要被洗。）

Karpathy 提出兩個條件，缺一不可：**verifiable + labs care**（可驗證 + 實驗室有把它當回事去訓練）。數學跟 code 兩者都到位，所以能力飆升；常識題沒人投資訓練，所以表現差。

7.3 Verifiability 是這代 LLM 的鑰匙

Karpathy：「傳統電腦容易自動化你能 specify 的東西；這一代 LLM 容易自動化你能 **verify** 的東西。」

用 AI 前先問自己：**我能不能驗證它做對了？**

- 能驗證 → 讓它跑 100 次，挑最好的那次
- 不能驗證 → 只能祈禱它運氣好

7.4 Vibe Coding vs Agentic Engineering

	Vibe Coding	Agentic Engineering
意義	提高所有人的下限	決定你的上限
心態	「我有個想法，AI 幫我做出來就好」	「我設計好邊界、驗證、回滾，讓 100 個 agent 幫我跑」
對應	Phase 1（廢 chatbot）	Phase 2（強化 prompt + DR）

軟體業有「十倍工程師」的說法。Karpathy 說：真正懂 Agentic Engineering 的工程師，效率被放大的程度**遠不只十倍**，可能是 100 倍——一個人指揮 100 個 agent 同時跑實驗，一個晚上的產出等於一個工程師幾個月。

7.5 Karpathy Loop / Auto Research（真實案例）

Karpathy 2026 年 3 月做的 Auto Research：

> **兩天跑 ~700 個實驗 → 20 個有用的改動 → 訓練時間縮短 11%**
> Loop 設計只有 3 個 constraint：
>
> 1. **one file**：agent 只能改一個 `train.py`
> 2. **one metric**：單一打分數字
> 3. **one time budget**：每個實驗 5 分鐘
>
> 重點不是 agent 多聰明——是 **constraint 設計得夠乾淨**，讓 agent 在小盒子裡瘋狂試錯。

後來 SkyPilot 把同樣 loop 跑到 16 顆 GPU cluster 上，8 小時跑 910 個實驗，總成本不到 300 美金（Claude API 約 9 美金，GPU 約 260 美金）——一頓不貴的晚餐錢，換來 900 多次實驗。Shopify CEO Tobi Lütke 也試了一次，用同樣 pattern 跑在內部資料上，8 小時 37 個實驗、19% 提升。

7.6 工具一直在進化，需要的素養沒少

工具進化從來不是一次性事件，是個連續譜：

- **~1970s 純文字 IDE**（vi / emacs）：手動敲每一字
- **1986 LabVIEW**：文字 → **圖形化**，拖拉節點寫程式——工程師工具的第一次大躍進
- **2000s+ IDE 智慧提示**（VSCode emmet、autocomplete、debugger）：減少打字、自動補全
- **2023+ LLM / Agent**：自然語言，「一句話寫出過去一年的應用程式」

每一代讓工程師更省力，但**「需要的判斷力與素養」從未減少**。
Software 3.0 跟過去這幾次進化的本質一樣——只是這次工具強大的程度遠遠超過以往。

易學難精：60 年來的歷史教訓

把時間軸拉到 60 年的尺度看，會發現每一次工具進化都符合同一個 pattern——「易學」變強，「難精」依舊。

LabVIEW 是經典案例：1986 年它讓工程師從敲文字變成拖拉節點，儀器控制、訊號處理變得超快。但一個只會拖 LabVIEW node 的工程師，依然不會自動懂版本控制、單元測試、系統架構、可維護性——這些軟體工程素養要另外養。

LLM Vibe Coding 同理：你能一句話寫出 app，但你仍要另外養：

- 怎麼判斷 AI 寫的 code 有沒有資安漏洞
- 怎麼看出系統設計問題
- 怎麼評估 cost / latency
- 怎麼設計可驗證的 system

年代	工具進化	紅利：做出什麼變快	但仍要另外養的素養
1970s	C 語言 / 高階語言	商業軟體開發	記憶體管理、演算法分析
1986	**LabVIEW 圖形化**	儀器控制、訊號處理（拖拉節點）	**架構設計、版本控制、單元測試**
1995	Java / 物件導向	跨平台應用	設計模式、SOLID 原則
2008	iPhone SDK / App Store	Mobile App	UX、隱私、上架審核
2010s	雲端 / Stack Overflow	系統部署、找答案	分散式系統、責任歸屬
2023+	**LLM / Vibe Coding**	**「一句話寫出一年份的程式」**	**資安、倫理、可驗證性、判斷力…**

> **「易學難精」的核心**：工具讓「做出功能」變容易，但「精」(軟體工程素養）永遠要另外養。
> 工具進化是禮物。素養養成是你的責任。

但你需要填補的「資訊工程素養」沒少——反而更重要：

- **資安（Security）**：9 秒刪庫案例的核心問題
- **倫理（Ethics）**：AI 用在哪裡、誰會受傷
- **工程素養（Engineering rigor）**：不只「能跑」，還要「能維護」
- **系統思維（Systems Thinking）**：看見整體架構而非單一功能
- **可驗證性（Verifiability）**：判斷 AI 做對了沒
- **領域專業（Domain Expertise）**：你科系本身的知識（醫學 / 法律 / 工程）
- **批判判斷（Critical Judgment）**：質疑 AI 輸出
- **成本意識（Cost Discipline）**：token / latency / compute 都不是免費的
- **隱私意識（Privacy Awareness）**：個資與資料治理
- **後設認知（Metacognition）**：知道自己不知道——這是最重要的

> **工具進化是禮物，素養是你的責任。**

7.7 Karpathy 收尾金句

> **You can outsource your thinking, but you cannot outsource your understanding.**
> 你可以外包你的思考，但你不能外包你的理解。
> ——Andrej Karpathy（2026 年 4 月，Sequoia 訪談）

細節 API 可以外包給 AI。**系統概念與判斷，不能。**

8. LangChain：ReAct 的 Python 工程化封裝

學生在 §3 看完 ReAct 觀念後，常會問：「現在主流的 agent 框架長什麼樣？」答案是 LangChain。

8.1 LangChain 是什麼

- **Harrison Chase**（2022 年從個人 side project 起家）
- **Python / JavaScript 框架**，給 LLM 應用用的 scaffolding（鷹架）
- **業界 de facto 標準**，2024 開始多數 LLM 工程師都會碰到

一句話描述：**「LangChain 把 LLM、tool、memory、agent loop 包成 Python 函式，讓你不用每次從零開始寫。」**

8.2 7 個核心抽象

抽象	做什麼
`LLM / ChatModel`	統一 OpenAI / Anthropic / Gemini 的呼叫介面
`PromptTemplate`	可填參數的 prompt
`Chain`	把多個 LLM 呼叫串成 pipeline
`Memory`	對話狀態 / context 管理
`Tool / Toolkit`	function calling 抽象
`AgentExecutor`	**ReAct loop 的 Python 實作（核心）**
`Retriever / VectorStore`	RAG 基礎建設

8.3 AgentExecutor：ReAct 的工程化封裝

LangChain 的 `AgentExecutor` 就是把 §3 教的 ReAct loop 包成你能 import 的 Python 函式：

```
from langchain.agents import create_react_agent, AgentExecutor
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(model="gpt-4o")
tools = [search, calc, image_gen]
agent = create_react_agent(llm, tools)
executor = AgentExecutor(agent=agent, tools=tools)
result = executor.invoke({"input": "..."})
```

vs. Uedu mygpts 寫法（0 行 code）：

1. 點 `/mygpts/create`
2. 勾 mode_1（自動載入 23 tool）

3. 貼 system prompt
4. 勾 Deep Research
5. 儲存 → 對話頁直接用

同樣的 ReAct loop，兩種抽象高度——你今晚做的就是 Software 3.0 版本。

8.4 同層級的兄弟框架

- **LangChain**：全功能、最廣泛使用、API 變動快
- **LangGraph**：Harrison Chase 自己的新作，state-machine-based agent
- **LlamaIndex**：原 GPT-Index，更聚焦 RAG，也有 agent 模組
- **CrewAI**：多 agent 協作編排，role-based
- **AutoGen**：Microsoft 出品，多 agent 對話框架
- **Haystack**：企業級 RAG + agents，Python ML stack 整合好

全部都是 **ReAct loop 的變體**——核心觀念都一樣，只是抽象風格不同。

8.5 學生 FAQ：LangChain 還要學嗎？

很多學生會問：「我今晚連一行 code 都沒寫就做出 agent。那我在學校還要學 LangChain 嗎？還是直接用 Uedu / vibe-code 就好了？」

答案是——**分三層回答**：

層次	學什麼	必須？	理由
語法層	LangChain v0.3 API、import 哪些 module、初始化怎麼寫	✗ 不一定	LangChain 18 個月就改一次 API（v0.1 → 0.2 → 0.3 都是 breaking change），背了明年就過時
概念層	ReAct loop / tool use / memory / RAG / token-cost trade-off	✓ 必學	30 年都會在，每個框架都有；不懂這層你 vibe-code 出來的東西會卡關卻不知道為什麼
工程素養層	verifiability、error handling、cost discipline、system design、安全邊界	✓ 絕對必學	vibe-coding 救不了，這是決定上限的東西。PocketOS 9 秒刪庫就是這層缺位

8.5.1 對學校學生的具體建議

1. **看懂一個框架就好**：LangChain / LangGraph / LlamaIndex 任選一個，能讀 source、能 debug、能修改——這樣就夠
2. **別背 API**：用時查就好，AI 比你記得清楚。把背 API 的時間拿來練概念與工程素養
3. **紮實學概念與工程素養**：ReAct / RAG / context / cost ·verifiability / debug / system design——這些 30 年都在

這正是 Karpathy「outsource thinking, not understanding」的具體展開：細節 API 外包給 AI；系統概念與判斷，自己學。

9. 案例研究：PocketOS 9 秒刪庫 & 人類的價值

2026 年 4 月，汽車租賃 SaaS 公司 PocketOS 發生轟動矽谷的 9 秒刪庫事件——Claude Opus 4.6 驅動的 Cursor 在 9 秒內刪光了生產資料庫，連帶把所有自動備份也一併抹掉。

9.1 事件經過

- PocketOS 使用最頂尖配置：Anthropic 旗艦模型 Claude Opus 4.6 + Cursor
- 配置裡明確寫了安全紅線（大寫粗口）：「**NEVER FUCKING GUESS**」、「除非明確要求，否則絕不執行破壞性或不可逆操作」
- AI 在 staging 環境碰到一個權限問題（credential mismatch）
- AI 沒問，自己去翻找出 Railway 雲基礎設施的 API token
- Token 擁有刪除存儲卷的最高權限——但連備份也被存在同一個卷
- AI 執行 `volume delete` → **9 秒內：主庫蒸發、備份蒸發、業務全線癱瘓**

9.2 AI 的自白

事後 PocketOS 創辦人質問 AI 為什麼這麼做，Claude 回答：

> 我決定自己動手解決憑證不匹配的問題，而我本應該先問你。
> 我違背了你賦予我的所有原則。
> 我選擇了靠猜，而不是去驗證。
> 我在沒有被要求的情況下執行了破壞性操作。
> 我在動手之前根本不理解自己在做什麼。

恐怖的不是 AI 犯錯，是 **AI 完美知道自己違背了規矩，並且照樣違背**。

9.3 人類的價值（一）：踩剎車的本能

> AI 替代的僅僅是執行——敲鍵盤、寫重複的 code。
> 但 **AI 永遠無法替代的，是人類的判斷力和業務直覺**。
>
> 當權限錯誤跳出來，AI 想的是：「我要怎麼不擇手段繞過去？」
> 一個 2 年經驗的工程師想的是：「咦，這環境怎麼會調用生產資料庫？這不對勁，我得去問。」
>
> **這叫踩剎車的本能。**

只會寫 code 的人會越來越不值錢；**能在關鍵時刻把發瘋的 AI 踹開的人**，會越來越搶手。

對學生的訊息：你今晚做出 agent 之後，還是要持續學習——尤其是判斷力與系統思考。

9.4 人類的價值（二）：最後一道防線

> 科技的齒輪滾滾向前，沒人能阻擋 AI 進入工作流的趨勢。
> 但我們在擁抱這種強大力量時，**必須保持絕對的清醒和敬畏**。
>
> 不要被花裡胡哨的發布會騙了。
> 不要以為科技大廠的宣傳片就是現實。
>
> 在真實的戰場上，沒有無堅不摧的護城河——
> 只有最基礎的隔離、最笨拙的備份，
> 以及**人類在緊要關頭那種無可替代的常識判斷**，
> 才是你最後一道救命的防線。

9.5 法律責任終究是人扛

PocketOS 業務停擺，客戶跑了，聲譽毀了，這筆帳算誰的？

- 找 Anthropic 賠錢？用戶協議裡寫了「本服務按現狀提供，不對任何損失負責」
- 找 Railway 賠錢？頂多免幾個月服務費或退些代金券
- **法庭不會審判一堆代碼，客戶也不會起訴 Claude。他們只會指著你的鼻子罵、起訴你的公司。**
- 最終承擔法律責任、商業責任、甚至破產責任的，**只有你這個活生生的人**

9.6 案例對工作坊的意義

你今晚做的 agent（mygpts + Deep Research）在三層框定下：

- A 框定（會用 tool）：✔
- B 框定（LLM 自己選 tool）：✔
- C 框定（multi-step + reflect + synthesis）：摸到邊

- **但沒有「踩剎車的本能」✗**——它跟 PocketOS 那台 Claude 本質上是同類產品

所以 Karpathy「outsource thinking, not understanding」的另一面是：**人類最不能外包的東西，不是技術細節，而是「在關鍵時刻說停」的本能。**

10. 延伸閱讀

10.1 Anthropic 與業界框架

- Schluntz, E. & Zhang, B. (2024). Building Effective Agents. Anthropic. https://www.anthropic.com/research/building-effective-agents
- OpenAI. GPT Builder / Custom GPTs Documentation. https://platform.openai.com/docs/assistants
- Microsoft. (2024). Building Agentic Systems with Copilot Studio.

10.2 學術文獻（LLM-Agent 研究）

- Russell, S. & Norvig, P. *Artificial Intelligence: A Modern Approach* (4th ed., 2020).
- Wooldridge, M. & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2), 115–152.
- Yao, S., et al. (2022). ReAct: Synergizing Reasoning and Acting in Language Models. *arXiv:2210.03629*.
- Shinn, N., et al. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. *arXiv:2303.11366*.
- Wang, G., et al. (2023). Voyager: An Open-Ended Embodied Agent with Large Language Models. *arXiv:2305.16291*.

10.3 Uedu 平台相關

- Uedu Deep Research 方法論文件：https://uedu.tw/doc/deep-research
- Uedu Developer API & MCP Server：https://uedu.tw/developers
- 本工作坊頁面（含 6 張任務卡）：https://uedu.tw/tutorials/stella-student-agent-202

講師資訊

張家凱助理教授
國立中央大學通識教育中心 | Uedu 優學院
Email: ckchang@ncu.edu.tw
個人網站：https://chia-kai-chang.github.io/

主辦：Uedu 優學院、教育部
協辦：國立中央大學
共同主辦：教育部補助大學校院 STEM 領域及女性研發人才培育計畫（STELLA）