Harness Engineering

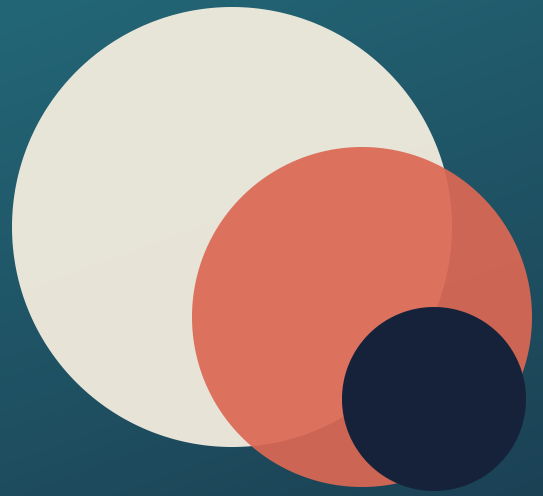
快速理解
Harness Engineering

這份教學要講清楚一件事：要讓 AI 在團隊裡穩定做事，靠的不只是 Prompt，而是整套運行機制。

Prompt：一句指令

Context：一張地圖

Harness：整套系統



今天先用一個簡單比喻來記：模型像馬，工程師像騎手，Harness 是把力量導向工作的馬具與車體。

AIU

@Akiryou0206

iiaiuii

一張圖理解：Prompt、Context、Harness 的差別

Prompt 是一句話，Context 是資訊範圍，Harness 是整套工作機制。

Prompt

回答時怎麼說

像是「向右轉」

Context

模型看到了什麼

像是「一張地圖」

Harness

模型如何被安排去工作

像是「整輛車 + 方向盤 + 煞車」

如果只記一句話：Harness Engineering 不是在教模型「怎麼回答」，而是在設計模型「怎麼工作」。

為什麼 2026 大家都在談 Harness Engineering？

對團隊來說，真正的差距不是模型夠不夠聰明，而是流程能不能讓它穩定完成長任務。

36%

20 步任務，每步 95% 成功率

整體仍只剩約三分之一成功。沒有檢查點、回滾點和驗證層，裸模型很容易在長鏈任務裡失手。

天花板效應

模型本身很強，但沒有流程、環境與約束，能力很難穩定釋放。

裸模型陷阱

做大工程時容易忘上下文、誤判完成、沒驗證就交卷。

壁壘改變

模型能力趨同後，真正的護城河變成系統設計與運行質量。

AIU

@Akiryou0206

iiaiuii

為什麼一般人用 AI，常常會覺得它「很降智」？

很多時候不是模型突然變笨，而是大家直接拿「裸模型」去做複雜、多步、跨上下文的任務。

1. 裸模型沒有流程

單輪問答時看起來很聰明，但一碰到複雜任務，模型本身不會自帶拆解、檢查、回滾與交接，所以很容易越做越亂。

2. 太想一步到位

一般使用者常直接丟大題目，結果模型容易上下文耗盡，或只看到局部進展就提前宣布完成，使用體感就會變得很差。

3. 長鏈路誤差累積

單步成功率就算有 95%，只要任務要連做 20 步，端到端成功率也只剩大約 36%，所以高單點準確率不等於整體可靠。

大家感受到的「AI 降智」，很多時候不是模型太差，而是少了 Harness：任務拆解、檢查點、驗證與回滾。

Harness 最常見的 6 個核心模組

跟同事討論 Agent 設計時，先把這六塊講清楚，後面很多問題都會更容易對齊。

上下文工程

決定要給模型看什麼、隔離什麼、何時壓縮。

工具編排

工具少而精，權限清楚，只開真正需要的能力。

驗證機制

用 Linter、測試、審查循環，避免自我感覺良好。

狀態管理

把待辦、進度、決策與 Git 檢查點留在外部記憶。

可觀測性

失敗時能回頭追原因，知道是哪一環出了錯。

人類接管

高風險動作先暫停，讓人做最後拍板。

記法：上下文、工具、驗證、狀態、觀測、人類接管。

一個穩定 Agent 的基本工作流

團隊裡比較實用的做法，不是一口氣跑完，而是每個階段都可驗證、可恢復、可交接。

1

拆任務

把大目標拆成小步

2

配上下文

只給相關資訊

3

叫工具

在受控權限內操作

4

先驗再過

先測再提交

5

寫狀態

更新待辦與 Git

6

該停就停

高風險交回人

實務上它不像一般固定腳本，而是讓模型在每一段都能被約束、被觀測、被中斷。

AIU

@Akiryou0206

iiaiuii

大廠實戰其實在收斂到同一個答案

這頁不是要背案例，而是想讓大家看到：做法不同，但共通點都指向分工、外部記憶、獨立驗證。

Anthropic

- 雙 Agent 進化成 Planner / Generator / Evaluator。
- 進度檔、清單、Git 歷史成為外部記憶。
- 把生成者與評估者拆開，降低自我偏誤。

OpenAI

- 靠上下文工程、架構約束、垃圾回收守住品質。
- 大量程式碼落地，依賴規範與驗證硬門檻。
- Agent 遇阻時先問：我缺的是什麼能力？

Google

- 生成器、驗證器、修正器三段式高度工程化。
- 重視跨步驟的狀態找回與驗證。
- 高可靠 Agent 幾乎都離不開驗證器。

我們可以直接借鏡的共同模式：生成與評估分離 + 外部持久化記憶 + 確定性驗證層

落地不要一口氣做完，先走這 3 步

如果團隊要開始落地，建議不要一次做滿，先從低成本、最容易見效的地方開始。

01

先寫 agent.md

先記錯誤、規則、交付標準。最快見效。

02

建立驗證層

把 Linter、測試、Hook 變成硬門檻。

03

做可拆的模組

讓記憶、工具、驗證能獨立替換，方便日後拆補丁。

如果要帶團隊開始做，最實用的順序是：先規則化，再驗證化，最後系統化。

最後帶走這 5 個重點

如果今天只帶走一件事，那就是：Harness 不是更長的 Prompt，而是更完整的工程系統。

- Prompt 管一句話，Context 管看到什麼，Harness 管怎麼把事做完。
- 團隊要的不是偶爾做對，而是可以反覆做對。
- 讓模型可靠的關鍵，是驗證、狀態與人類接管節點。
- 外部持久化記憶，比讓模型硬記更重要。
- 好的 Harness 要能長，也要能拆，方便下一代模型接手。

一句收尾

對團隊來說，未來真正的差異化，不只是誰用到更強的模型，而是誰能把模型放進一套可靠、可觀測、可回滾、可交接的工作系統裡。

這就是我們今天談 Harness Engineering 的原因。

AIU

@Akiryou0206

iiaiuii