

# 一天內專題Demo



[Demo07c] AI代理設計模式\_員瑛式思考生成器Two\_Stage\_CoT版.ipynb  
檔案

這個是我的參考專題，請以他的概念延伸，我能做一些什麼有趣的專題Demo，並且是要在1天之內能做完

下面我會根據你提供的 「AI代理設計模式」 「員瑛式思考生成器」 「Two-Stage CoT」 等概念，延伸出 **你在1天內可以做完的、有趣又可展示的專題 Demo**。

這些 Demo 都是「可在一天內完成、可展示、可重複利用、可延伸成更大的專題」的形式。

## 你的參考專題核心概念（簡化後）

你給的 Notebook 主要展示：

- Two-Stage CoT（先產生思考，再根據思考產生答案）
- 代理模式（Agentic Pattern）
- 特定思考框架（員瑛式思考）
- 自動化生成推理 + 行動結果的流程化 Demo

## 我為你設計的「一天內可完成、有趣的 Demo」

### ① AI 推理代理：自動分析線上文章的「多階段思考摘要器」

概念延伸：使用 Two-Stage CoT 讓模型先思考、再生成結果。

功能

1. 輸入任意文章（你貼網址或貼文字）

2. 代理流程：

- Step1：模型生成「深度思考框架」：主題、觀點、隱含結論
- Step2：模型依據思考結果生成：
  - 150字摘要
  - 三大洞察
  - 一句話結論

3. UI 可用 Streamlit（1小時內搞定）

📌 為什麼有趣？

很像「文章超級理解助手」，老師與學生都會喜歡。

🕒 所需時間：4-6 小時

## ② AI 會議紀錄助理（使用 Two-Stage CoT 提升邏輯）

1. 貼上會議文字稿

2. Two-Stage CoT：

- Stage1：分析涉眾、主議題、衝突點
- Stage2：輸出會議結論、待辦事項、清單化

✨ 可展示：

- 「我貼一段混亂的會議稿，AI 自動整理得像專業秘書」

🕒 4 小時內完成

## ③ “錯誤推理偵測器” — AI 自動找出人類邏輯錯誤

利用你的 Two-Stage CoT 代理，讓模型先分析推理性結構，再標出謬誤。

範例：

輸入：一段學生作文或論證

輸出：

- 推理步驟（模型產生）
- 哪裡有跳步？
- 什麼謬誤（稻草人／非黑即白／訴諸情緒）？
- 重寫版本

✨ 展示時超加分：

你給一段超亂的文章，AI 做自動「修正推理」。

🕒 約 6 小時

## ④ “AI 角色化 CST（Context-Specific Thinking）代理”

讓 AI 依照角色不同，產生不同邏輯風格的回答。

例如：

角色	思考方式（由模型產生）
工程師	Debug flow, causal chain
商管教授	SWOT、5 Force

角色	思考方式（由模型產生）
心理師	CBT + 情緒標記
作家	故事化 CoT

展示方式：

在 UI 中選角色 → AI 自動套用該角色專屬 CoT 思考流程

這是你「員瑛式思考」概念的延伸：

AI 會根據角色改變推理架構 → 這是很酷的 Demo

⌚ 6 小時內可完成

---

## ⑤ “AI 研究助理”——自動生成 Relevance Graph + 思考鏈

如果你常寫論文，這超好展示：

1. Input：研究題目的描述
2. Stage1：Two-Stage CoT 建立關鍵因素圖（GraphRAG 精簡版）
3. Stage2：生成
  - 研究缺口
  - 潛在變數
  - 問卷題項草稿

你可以在 1 天內做一個「迷你 GraphRAG」。

不需要 Neo4j，用 Python list + dict 就好。

⌚ 8 小時內搞定

---

## ⑥ “AI 任務分解器”——自動把任何需求拆成行動計畫

例如輸入：

我要做一個論文自動整理系統

AI 會輸出：

1. Stage1（推理）
  - 需求拆解
  - 隱含依賴性
  - 風險、障礙
2. Stage2（產出）
  - 甘特圖文字版
  - 步驟 1-20
  - 可直接 copy 到 Notion 的版面

 展示超實用，討論系統架構時很好用。

 3 小時內完成

---

## 我最推薦你做的（一天內最穩、最亮眼）

### 「AI 會議紀錄 + 推理決策助手」

為什麼？

- Demo 非常直觀
  - 老師與業界都覺得“有價值”
  - Two-Stage CoT 用得剛剛好
  - 很容易展示：
    - 貼上雜亂文字 → AI 輸出完美報告
  - 3-5 小時就能做出可展示版本
- 

## 若你願意，我可以幫你：

- ✓ 幫你選定一個題目
  - ✓ 幫你寫完整架構
  - ✓ 幫你做 1 天版的開發計畫
  - ✓ 我還可以幫你生成 Streamlit UI prototype
  - ✓ 甚至直接把 Python 主程式寫給你
- 

## 要我幫你把哪一個 Demo 「做成可實作版本」？

- 1 多階段文章摘要器
- 2 AI 會議紀錄助理
- 3 推理錯誤偵測器
- 4 角色化思考代理
- 5 AI 研究助理
- 6 任務分解器

你說一個，我就直接開始幫你寫完整可執行版本（含程式碼）。



我想法是有老子式、孔子式、管仲式思考生成器可以供使用者選擇

太棒的主題！

「老子式、孔子式、管仲式思考生成器」

這是一個非常亮眼、超有特色、又能在 **1 天內完成** 的 Demo。

下面我幫你整理：

---

## ✓ 你要做的 Demo：

### 《古代思想家風格的 AI 思考生成器》

使用者可以選擇：

- 老子式（道家）
- 孔子式（儒家）
- 管仲式（法家/務實治國）

AI 會依據不同思想家，套用不同的「思考框架」，再使用 Two-Stage CoT 生成答案。

---

## 系統工作流程（簡單 + 帥氣 + 容易做）

這是你 Demo 的標準流程：

---

### STEP 1 — 使用者輸入問題

例如：

「我要如何讓我的新團隊快速運作？」

---

### STEP 2 — 使用者選擇思想家風格

選項：

- 老子
  - 孔子
  - 管仲
- 

### STEP 3 — Two-Stage CoT 執行

#### 📌 Stage 1：生成該思想家的「思考框架」

例如：

#### ● 老子式（道家）

- 無爭而治
- 柔弱勝剛強

- 道法自然
- 反者道之動
- 去欲、去控制  
→ 生成一段「推理鏈」：以自然、順勢、反向觀點出發

### ● 孔子式（儒家）

- 仁義禮智信
- 以德服人
- 修身齊家治國
- 名正言順  
→ 推理鏈會偏向倫理、秩序、人際角色

### ● 管仲式（務實治國/法家）

- 效率、制度、利害
- 富國強兵
- 奬懲分明
- 用人、分工、制度化  
→ 推理鏈偏向制度設計、資源配置

## STEP 4 — Stage 2 根據思想家的推理鏈輸出答案

例如問題：

「我要如何讓我的新團隊快速運作？」

### ● 老子式回答（哲學、大局、自然、反向思維）

你越急，團隊越慢。

領導者先退半步，讓團隊自行找出節奏。

「無為」不是不做，而是不妨礙。

讓流程自然流動，反而運作更快。

### ● 孔子式回答（道德、秩序、角色）

領導者必須先修身，讓團隊有所依循。

明確角色、確立規範，以禮使人。

團隊才能心齊，而後行速。

### ● 管仲式回答（務實、制度、效率）

先建立制度、分工、獎懲。  
資訊透明、責任清楚、資源充足。  
團隊自然高速運轉。

---

## 🔥 這個 Demo 的亮點

- ✓ 一天內能做完
  - ✓ 超容易展示（問任何問題 → 三位古人回答）
  - ✓ 可靈活擴充（孟子、荀子、孫子兵法、墨子...）
  - ✓ 展現你「思考架構 + Agent + Two-Stage CoT」的功力
  - ✓ 老師、評審、觀眾都會覺得超有創意
- 

## 🛠 我可以立刻幫你做的三件事

如果你願意，我可以直接給你：

---

① 完整的 Prompt 設計（含三套思考鏈模板 + Two-Stage-CoT Workflow）

---

② Streamlit UI 程式碼（可以直接跑 Demo）

---

③ 完整 Python 主程式（1 檔案即可 Demo）

---

## 🎯 你想讓我先完成哪一部分？

- 1 完整 Prompt？
- 2 Python 程式碼？
- 3 Streamlit UI？
- 4 還是把三個都整合成一個「一天版完整專題」？

你告訴我，我馬上開始寫。



請先給我一個完整程式規劃書，我使用groq的api

太好了 — 我會直接給你一份可立刻執行、以 Groq API 為推理後端的「老子式 / 孔子式 / 管仲式思考生成器」完整程式規劃書。這份規劃書包含系統架構、API 呼叫範例（Python/JS）、Two-Stage CoT 的 Prompt 模板

(可直接貼入請求 body)、前端/後端最小實作清單、1 天開發時間表與測試檢查表。

我在關鍵處註明 Groq API 的使用方式與來源（官方文件），你可以直接用這些範例去串接 Groq 的 Responses/Completions 端點。[console.groq.com+2](https://console.groq.com/)

---

## 一頁總覽（目標）

在 1 天內完成的 Demo：

**古代思想家風格的 AI 思考生成器** — 使用者輸入問題並選擇風格（老子 / 孔子 / 管仲），系統以 Two-Stage Chain-of-Thought (Stage1：產生該思想家的「思考框架/推理鏈」；Stage2：根據推理鏈生成最終建議) 回應。後端使用 Groq API 作為 LLM 推理引擎。[console.groq.com+1](https://console.groq.com/)

---

## 系統架構（簡潔）

1. 前端：單頁應用（Streamlit 或簡易 React + Vercel 部署）
2. 後端：輕量 API（可用 Flask / FastAPI 或直接由前端呼叫 Groq，如果要藏 API key 則放後端）
3. LLM 推理：Groq Responses API（OpenAI-compatible base URL: <https://api.groq.com/openai/v1>，用 API key 驗證）。[console.groq.com+1](https://console.groq.com/)

資料流程：

User → Frontend UI (問題 + 選擇思想家) → Backend (構造 Two-Stage prompts) → Groq Responses API → Backend 處理/format → Frontend 顯示（「推理鏈」與「最終答案」兩個分段）

---

## 功能需求（最小可行產品 MVP）

必須功能（1 天內完成）：

- 輸入框：問題（文字）
- 選單：三種風格（老子 / 孔子 / 管仲）
- 「產生」按鈕 → 顯示：
  - Stage 1：思想家式的**推理鏈（CoT）**（條列式）
  - Stage 2：根據推理鏈的**最終建議（短文）**
- 顯示 token/latency（可選）
- 可複製的 Prompt 與可編輯（方便 Demo）

Nice-to-have（若時間允許）：

- 替代風格：孟子 / 荀子 / 孫子（擴充）
- 儲存歷史紀錄（localStorage）
- share link（把 question+style encode 在 URL）

## Two-Stage CoT 設計（關鍵：prompt templates）

我們用 **兩個連續請求** 或 **一次請求內兩段 system/instruction** (Groq 支援 OpenAI-compatible Chat/Responses) 來實作 Stage1→Stage2。建議採用「先要求思考輸出 (explicit chain)」，再要求答案且要引用該 chain」。

設計原則：Stage1 要求可讀的推理步驟/觀點清單（不超過 300 tokens），Stage2 要求根據 Stage1 的點逐條產出建議（150–250 tokens）。

下面是可直接拿去呼叫的三套 Prompt (可放到 request body) ——把 <> 替換成使用者問題。

## 共用 Stage1 模板（會根據思想家替換「思考架構」段落）

```
pgsql
```

**System**: 你是模擬古代思想家的智慧助手。請嚴格模擬其思考風格與核心觀念。不要直接給最終建議，先產生「推理鏈」(Chain-of-Thought)，以條列方式列出 4–6 個推理步驟或視角，並標明每一步的核心論點。

**User**: 思考風格: <>。問題: "<>"

**Instruction**: 根據 <> 的哲學/實務觀點，寫出 4–6 條推理步驟（每條 1–2 句），條列時在前面加上 Step 1/Step 2 等標籤。不要做總結，也不要直接執行 Stage2 的生成。

### 三個 <> 範例（可直接插入到 System 或 User 內）

- **老子式（道家）** : 聚焦「無為、順勢、柔弱勝剛強、反者道之動」。以大局和反向思維為主。
- **孔子式（儒家）** : 聚焦「仁義禮智信、修身齊家、以德服人」。以倫理、秩序、人際為主。
- **管仲式（務實/法家風格）** : 聚焦「制度、分工、效率、獎懲」。以制度化、利害衡量為主。

(提示：你可以直接在 Stage1 的 system 中放入這三段關鍵詞來引導 Groq。)

---

## Stage2 模板（依賴 Stage1 輸出）

```
pgsql
```

**System**: 你現在是根據前面「推理鏈」生成實用建議的執行者。請嚴格根據 Stage1 的 Step 1..N 的內容，對應每一條推理給出具體建議（每條 1–2 句），最後產出 1 段 3–5 句的總結建議，語氣友善、可操作。

**User**: 下面是 Stage1 的推理鏈：

<>

**Instruction**: 逐條對應上面的 Step，為每條提供 1–2 句可執行建議；最後寫一段 3–5 句的總結（如給領導者的 30 秒建議）。

實作方式：

- Option A (簡單) : 發兩次請求，第一次呼叫 Stage1，拿回 stage1\_text，再把 stage1\_text 當作 input 發第二次請求 Stage2。
- Option B (一請求雙段式) : 把 Stage1 的 instruction 和 Stage2 的 instruction 一併放在 messages 中 (先 system 要求產出 chain，然後 assistant -> user 模擬回傳 chain，再 assistant 再被提示生成答案)。若 Groq 的 Responses 支援工具/多段流，這會更節省延遲。[console.groq.com/+1](https://console.groq.com/+1)

小提醒：為穩定可解釋性，建議採 **兩次請求**（更容易在 UI 顯示「推理鏈」），除非你熟悉一次請求內 multi-turn 模擬。

## Groq API 使用要點（官方要點）

- Base URL (OpenAI-compatible) : <https://api.groq.com/openai/v1>。可用 OpenAI-compatible client libs，但要設定 `baseURL` 與 `apiKey`。[console.groq.com](https://console.groq.com/)
- 官方 quickstart 建議可用 JavaScript 的 `ai` SDK + `@ai-sdk/groq`，也可直接用 Python/requests。  
[console.groq.com](https://console.groq.com/)
- Responses API 支援 chat/completions 功能（choices、usage 等），你可以拿回 model、tokens、status 等。[console.groq.com](https://console.groq.com/)

## 實作細節：程式碼範例

下面提供**最小可跑的** Python 與 JavaScript 範例（採兩次請求 Stage1→Stage2）。

你可以直接把 `GROQ_API_KEY` 設為環境變數；若前端直接呼叫，務必使用後端中介來保護金鑰。

## Python (requests) 簡單範例

```
python

import os, requests

API_KEY = os.environ.get("GROQ_API_KEY")
BASE = "https://api.groq.com/openai/v1"

headers = {
    "Authorization": f"Bearer {API_KEY}",
    "Content-Type": "application/json"
}

def groq_chat(messages, model="gpt-4o-mini", max_tokens=400, temperature=0.6):
    payload = {
        "model": model,
        "messages": messages,
        "max_tokens": max_tokens,
        "temperature": temperature
    }
    resp = requests.post(f"{BASE}/chat/completions", headers=headers, json=payload)
    resp.raise_for_status()
    return resp.json()

# Stage1
messages_stage1 = [
    {"role": "system", "content": "你是模擬古代思想家的智慧助手。... (使用上面的 Stage1 template)" },
    {"role": "user", "content": '風格：老子式；問題："我要如何讓我的新團隊快速運作?"'}
]
r1 = groq_chat(messages_stage1)
stage1_output = r1["choices"][0]["message"]["content"]

# Stage2
messages_stage2 = [
    {"role": "system", "content": "你現在是根據前面「推理鏈」生成實用建議的執行者..." },
    {"role": "user", "content": stage1_output}
]
r2 = groq_chat(messages_stage2)
```

```
stage2_output = r2["choices"][0]["message"]["content"]

print("== Stage1 ==\n", stage1_output)
print("== Stage2 ==\n", stage2_output)
```

範例參考 Groq Responses API 的 chat endpoint 格式（OpenAI-compatible）。 [console.groq.com](https://console.groq.com)

## JavaScript (fetch, 前端呼叫應透過後端)

```
javascript

const base = "https://api.groq.com/openai/v1";
async function groqChat(messages) {
  const res = await fetch(`${base}/chat/completions`, {
    method: "POST",
    headers: {
      "Authorization": `Bearer ${process.env.GROQ_API_KEY}`,
      "Content-Type": "application/json"
    },
    body: JSON.stringify({
      model: "gpt-4o-mini",
      messages,
      max_tokens: 400
    })
  });
  return await res.json();
}
```

(若你使用 Vercel AI SDK / @ai-sdk/groq，可以參考 quickstart 範例來更少 boilerplate。) [console.groq.com+1](https://console.groq.com+1)

## 前端建議 (Streamlit 最快)

Streamlit 是 1 天 MVP 的最佳選擇（簡單、可快速展示）：

- input\_text\_area (問題)
- selectbox (老子/孔子/管仲)
- button (產生) → 呼叫後端函式 (或直接呼叫 Groq)
- 顯示 Stage1 與 Stage2 (兩欄或折疊面板)
- 顯示「Copy Stage1 Prompt」按鈕方便展示

若你想更有互動性，可做成 React (Next.js) 搭配 Vercel + Groq (官方有 Vercel 範例整合)。 [console.groq.com](https://console.groq.com)

## 部署與安全 (重要)

- **不要把 API Key 放前端。** 若將前端直接呼叫 Groq，必須使用短期 token 或 proxy server。官方示例建議在後端設定 GROQ\_API\_KEY。 [console.groq.com+1](https://console.groq.com+1)
- Rate limits / model 選擇：開發時用較小模型或較低 max\_tokens 省成本 (Groq Console 有 rate limit 與 model list)。可參考 Groq docs。 [console.groq.com](https://console.groq.com)

## 1 天開發時間表 (實際可在 6-8 小時內完成 MVP)

- 0.5 小時：建立 repo、安裝套件（streamlit / requests 或 Next.js）、設定環境變數
  - 1 小時：實作後端簡單路由（/api/generate），包含 Stage1 → Stage2 兩次呼叫邏輯
  - 1.5 小時：做前端 UI（Streamlit/React）輸入欄 + 風格選單 + 顯示 panel
  - 1 小時：整理 prompt 模板、微調 prompt（測試 3-5 次）
  - 0.5 小時：加入錯誤處理 / loading indicator / copy 按鈕
  - 0.5 小時：最後測試、準備 Demo 範例題目（3 個示例問題）
  - 0.5 小時：部署（Vercel / Streamlit share）與拍攝短 demo（或錄音）
- 

## 測試檢查表（Demo 前務必跑）

- ■ Stage1 會穩定回傳 4-6 條條列推理
  - ■ Stage2 會根據 Stage1 的內容逐條給出建議（檢查是否有遺漏 Step）
  - ■ 三種風格在語氣與角度上有明顯差異（老子偏大局/逆向、孔子偏倫理/秩序、管仲偏制度/效率）
  - ■ API Key 無洩漏（前端 console/network 確認）
  - ■ latency 與 token 使用量在可接受範圍（調整 max\_tokens）
  - ■ 準備 2-3 個「示範題目」方便現場演示
- 

## 範例示範題（Demo 用）

1. 「如何在兩週內把新團隊拉起來？」
2. 「面對團隊內部衝突，我應該怎麼處理？」
3. 「在有限預算下，如何提升產品上線速度？」

在演示時，先顯示 Stage1（推論鏈），再展示 Stage2（逐條建議），讓評委看到 Two-Stage CoT 的可解釋性。

---

## 延伸/升級建議（日後）

- 加入「風格強度」滑桿（影響 prompt 的保守/激進程度）
  - 加入「例子」庫（每種風格提供古文或名言參考）
  - 導出 PPT/Notion 摘要
  - 支援語音輸入/播放（用 ElevenLabs 等 TTS）
- 

## 重要參考（我用到的官方文件）

- Groq API Reference / Responses (OpenAI-compatible) — chat/completions endpoints & schema. [console.groq.com/+1](https://console.groq.com/+1)
- Groq Quickstart & ai SDK (@ai-sdk/groq) — 若用 JS SDK，此為官方入門。 [console.groq.com](https://console.groq.com)

- Groq 官方首頁 / 服務說明（模型與低延遲解說）。 [Groq](#)
  - Vercel + Groq AI SDK 範例（若你用 Next.js / Vercel 會很方便）。 [console.groq.com](#)
- 

如果你要我立刻把整個專案的程式碼（包含 Streamlit 前端 + Flask 後端）寫出來，或是我直接給你 Stage1/Stage2 的完整 prompt 文字（含三種風格的精確引導語），我可以馬上開始把它 寫成可執行的 repo 範本（含 README 與部署指令）。你想先要哪一項？

1. 完整 repo (Streamlit + Flask + prompts + README)
2. 只要完整 prompt 套件（可貼入 Groq 請求）與 demo 問題集
3. 只要後端 API 範例（Python）與部署步驟

選一個，我立刻把對應的可執行程式碼給你（含可直接貼到 Groq 的 request body）。