書中，我也會平衡「深入理解理論」和「實務操作技能」兩者。我會在每一章使用大量的比喻來簡化複雜的事情，佐以小段的程式碼來讓概念更具體。本質上，我想讓這本書成為你的 LLM 講師 + 助教，以簡化並揭示這片迷人的領域，而不是展示一大堆術語。我想讓你在看完每一章時都能夠清楚地理解主題，並知道如何將它應用在現實場景中。

如何閱讀本書

如果你有一些機器學習經驗，你的旅程將比沒有任何經驗的人更輕鬆一些。但是，只要你會寫 Python，並做好學習的準備，你就可以在這條道路上邁進。在閱讀本書時，你可以根據你的背景、目標和時間，以不同的程度來參與。你可以深入研究實踐的小節，試著編寫程式與調整模型，或閱讀理論的部分，以深入瞭解 LLM 如何運作，而不編寫任何一行程式碼。選擇權完全在你手上。

在瀏覽這本書時，別忘了，一般來說，每一章都以之前的內容為基礎，你在每一節學到的知識和技能可以幫助你閱讀接下來的內容。你將面臨的挑戰都是學習過程的一部分。有時你可能感到疑惑、挫折，甚至完全卡住。當我為這本書開發視覺問答（visual question-answering，VQA）系統時，模型曾經不斷輸出無意義的內容，反覆產生相同的短句，使我經歷一次又一次的失敗。但是，在經過無數次的反覆改進之後，它開始產生有意義的輸出，當我沉浸在成功和突破的喜悅之中時，我認為之前遭遇的所有失敗都值得了。這本書將為你提供類似的挑戰，進而創造類似的成就。

本書概要

本書分為四個部分。

第一部分：大型語言模型簡介

第一部分是 LLM 的簡介。這部分提供快速入門 LLM 所需的基本知識，包括提示工程、Transformer 架構的底層注意力機制、到 RAG（檢索增強生成）及 agent。

第 1 章：大型語言模型概述

本章廣泛地介紹 LLM 世界，涵蓋一些基本知識，包括 LLM 是什麼、它們如何工作，以及它們為何重要。本章將幫助你理解本書的其餘部分，在你看完時，為你奠定紮實的基礎。

第 2 章：使用 LLM 來進行語意搜尋

第 2 章在第 1 章的基礎上，深入探討如何用 LLM 來執它最有影響力的任務：語意搜尋。我們將製作一個能夠理解查詢詞條的含義，而非僅僅比對關鍵字的搜尋系統。

第 3 章：踏出提示工程的第一步

寫出有效的提示詞既是一門藝術，也是一門科學，它是充分發揮 LLM 威力的關鍵。第 3 章提供提示工程的實用介紹，以及充分運用 LLM 的指南和技術。

第 4 章：AI 生態系統：整合所有組件

第 4 章深入討論兩個案例研究：建立 RAG 流水線，以及利用前幾章的知識來建立一個 agent。

第二部分：榨出 LLM 的所有潛力

第二部分帶你更上一層樓，目標是協助你微調 LLM 並 embed 模型，以充分發揮 AI 系統的效能。

第 5 章：使用自訂的微調來優化 LLM

在 LLM 的領域裡沒有一體適用的解決方案。第 5 章介紹如何使用你自己的資料集來微調 LLM，並提供實際的範例和練習，讓你能夠迅速自訂模型。

第 6 章：進階提示工程

我們將在第 6 章深入研究提示工程的世界。本章探討進階的策略和技術，它們可以幫助你更充分地利用 LLM，例如輸出驗證，和語意 few-shot（少量範例）學習。

第 7 章：自訂 embedding 與模型架構

我們將在第 7 章討論更多 LLM 的技術面，介紹如何修改模型架構和 embedding，讓模型更適合你的使用情境及需求。我們也會調整 LLM 架構以滿足需求，並微調一個優於 OpenAI 模型的推薦引擎。

第 8 章：AI 對齊：第一原則

本章將後退一步，檢視既有基礎流程，讓 AI 系統更實用、造成更少負面影響，並且更容易操作。本章的目的是剖析「對齊」這個概念，以便突顯不同機構創造的 LLM 之異同。

第三部分：LLM 進階用法

第三部分繼續設計與評估自訂的 LLM 架構，我們將使用 RLHF 來從零開始訓練「指導式對齊的」（instruction-aligned）聊天機器人，並量化與提煉 LLM，讓它在生產環境中具備最佳效能。

第 9 章：超越基礎模型

第 9 章探討一些次世代模型及架構，它們正在擴大 LLM 的可能性。在本章，我們將結合多個 LLM，並使用 PyTorch 來建立一個框架以自訂 LLM 架構。本章也介紹如何利用回饋來進行強化學習，讓 LLM 更符合需求。

第 10 章：微調進階的開源 LLM

第 10 章提供微調進階開源 LLM 的指南和範例，並把焦點放在實作上。我們不僅透過一般的語言建模來微調 LLM，也使用回饋強化學習等進階方法，以 Meta 的 Llama-3 模型為基礎，建立我們自己的 instruction-aligned LLM——我們稱它為 SAWYER。

第 11 章：將 LLM 投入生產

本章探討在生產環境中部署 LLM 的現實問題，我們將介紹如何擴展模型，處理即時請求、確保模型穩健可靠，並優化執行速度及記憶體使用量。

第 12 章：評估 LLM

顧名思義，最後一章的目的是藉著探討效能評測、模型探查、模型校準等主題，來鞏固 LLM 評估的流程和框架，以輸出更值得信賴的 AI 預測結果。

第四部分：附錄

接下來的三個附錄包含常見問題、詞彙表，以及 LLM 原型參考。

附錄 A：LLM FAQ

作為一位顧問、工程師和教師，我每天都會收到許多關於 LLM 的問題，本附錄整理一些比較有影響力的問題。

附錄 B：LLM 詞彙表

在這個詞彙表裡，有本書主要術語的進階參考。

附錄 C：LLM 應用程式原型

在本書中，我們使用 LLM 來建構許多應用程式，想要自行建構應用程式的讀者可以把附錄 C 當成起點。這個附錄列出在常見的 LLM 應用領域裡，應關注的 LLM 有哪些、你可能需要哪些資料，以及可能面臨哪些問題及如何解決它們。

本書特點

「本書與其他書籍有什麼差異？」，我聽到你的疑問了。首先，本書融入我的各種經驗，它們來自我的理論數學背景，以及擔任創業者、大學講師、企業家、機器學習工程師，和風險投資顧問時的經歷。這些經歷塑造了我的 LLM 知識，我將所有知識都載入此書。

你將會發現本書有一項特點：實際應用概念。我說的「真實世界」是真的：這本書充滿真正的實作經驗，可幫助你瞭解使用 LLM 的實況。

此外，本書並非只帶你瞭解領域的現況，正如我常說的，LLM 的世界隨時都在變化。即使如此，有些基本知識是持久不變的，我將在書中強調它們，讓你不僅為當下做好準備，也為將來未雨綢繆。

案例研究 1：檢索增強生成

在使用 LLM 時，我們會立刻遇到它很容易產生**幻覺**這個問題，簡而言之，就是模型憑空捏造看似正確的內容。這種行為該不該稱為「幻覺」是值得討論的有趣議題，但我想把這個話題留到另一本書再談（希望如願）。針對這種幻覺，有一種廣受歡迎的解決方案是建立 **RAG（檢索增強生成）**系統，結合 T5、GPT、Llama 等生成模型與 BERT 等檢索模型，將檢索模型取得的資訊填入生成模型。圖 4.4 是 2020 年發表的原始論文「Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.」裡的一張示意圖[2]。

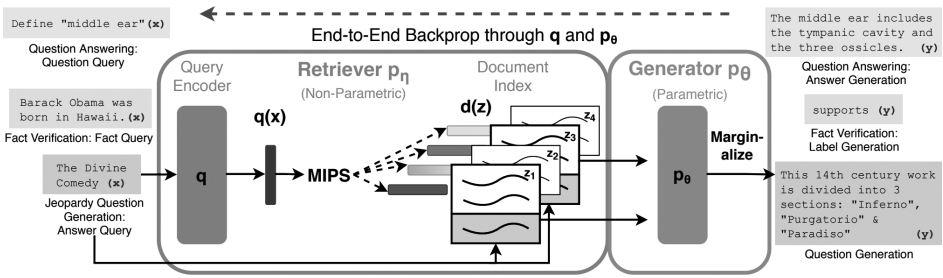

圖 4.4　原始的 RAG 論文也介紹了微調 RAG 效能的其他進階訓練方法。資料來源：Lewis, P. et al. “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.” *Advances in Neural Information Processing Systems* 33 (2020): 9459-74。摘自 https://arxiv.org/abs/2005.11401。

我們將使用 GPT-4 和第 2 章建立的語意檢索系統來建構一個非常簡單的 RAG 應用程式。

各個零件之總合：retriever 與 generator

我們的 RAG 系統包含以下兩部分：

- **retriever（檢索器）：**它會將真實（ground-truth）知識放入資料庫，讓收到查詢的 LLM 檢索它們。我們將使用第 2 章的語意搜尋 API 來進行檢索。

2　https://arxiv.org/abs/2005.11401

- **generator（生成器）：**這是一個 LLM，它將推理用戶的查詢詞，並結合檢索出來的知識，以即時提供對話式回應。我們將使用 GPT-4。

我們有一個語意搜尋 API 端點可以根據自然查詢，從資料集檢索文件。我們只要完成以下的四個步驟，即可啟動 RAG 系統：

1. 為 GPT-4 設計系統提示詞，透過 few-shot 學習和思維鏈提示詞來展示所需的對話結構。
2. 當人類向機器人詢問問題時，查詢我們的語意搜尋系統。在第 2 章，我們已經完成大部分的資料分段、向量化，和索引製作工作了。現在我們僅利用該系統的最初用途：即時存取上下文。
3. 將取自 DB（資料庫）的任何上下文直接注入 GPT-4 的系統提示詞。
4. 讓 GPT-4 執行它的工作，並回答問題。

你可以在圖 4.5 看到這些高階步驟。圖 4.6 是提示詞層面的詳細步驟。

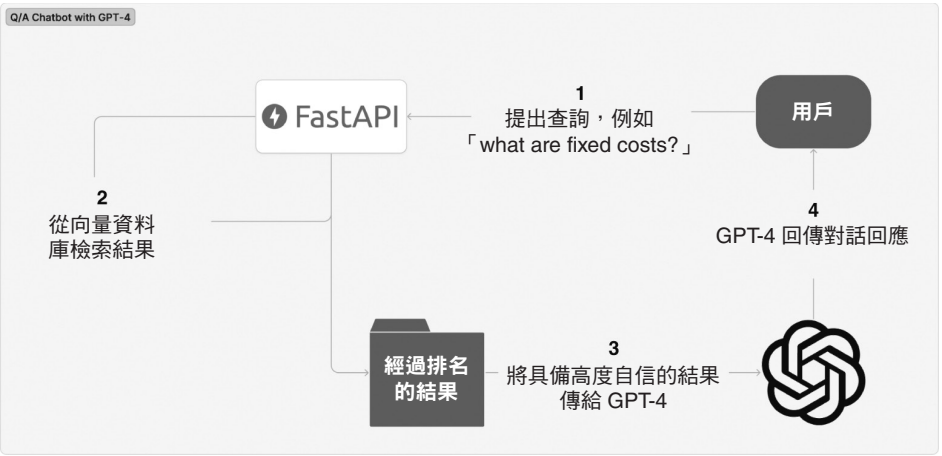

圖 4.5 RAG 聊天機器人的鳥瞰圖。這個機器人使用 GPT-4 作為語意搜尋 API 之前的對話介面。

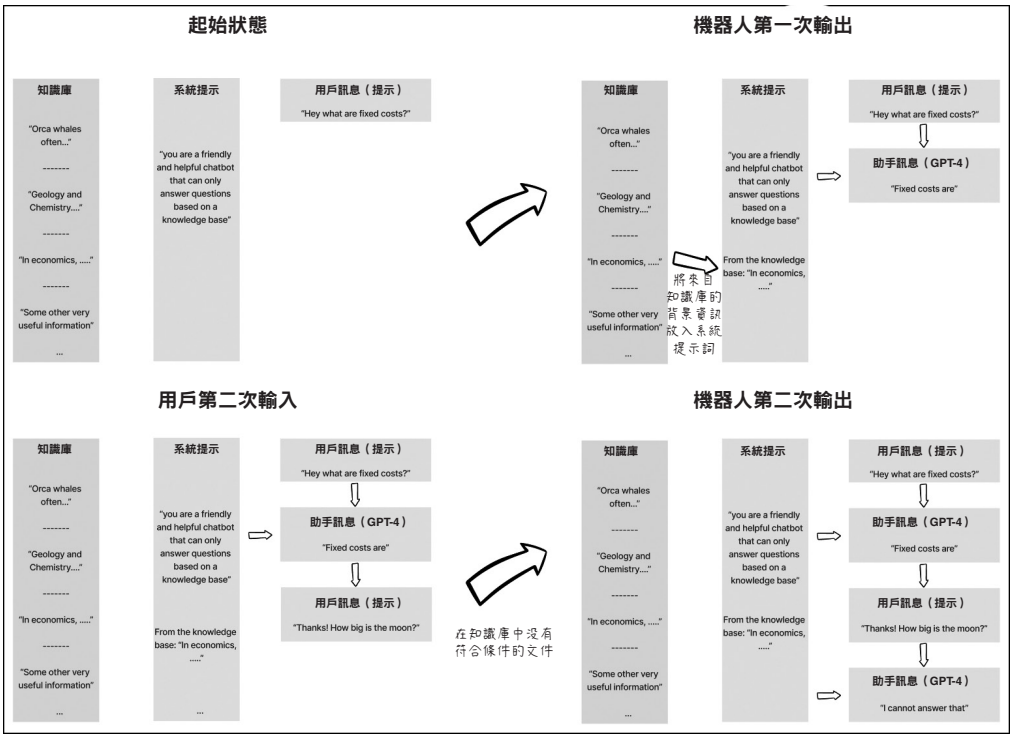

圖 4.6　從左上角開始，由左到右的四個狀態，代表機器人的內在架構。每當用戶輸入的內容可從知識庫取出有信心的文件時，就直接將該文件插入系統提示詞。我們用系統提示詞來要求 GPT-4 只使用來自知識庫的文件。

我們將這些邏輯包成一個 Python 類別，範例 4.1 是它的結構。

範例 4.1　GPT-4 RAG 機器人

```
client = OpenAI(api_key=userdata.get('OPENAI_API_KEY'))

class ChatLLM(BaseModel):
    model: str = 'gpt-3.5-turbo'
    temperature: float = 0.0

    def generate(self, prompt: str, stop: List[str] = None):
        response = client.chat.completions.create(
            model=self.model,
            messages=[{"role": "user", "content": prompt}],
            temperature=self.temperature,
```

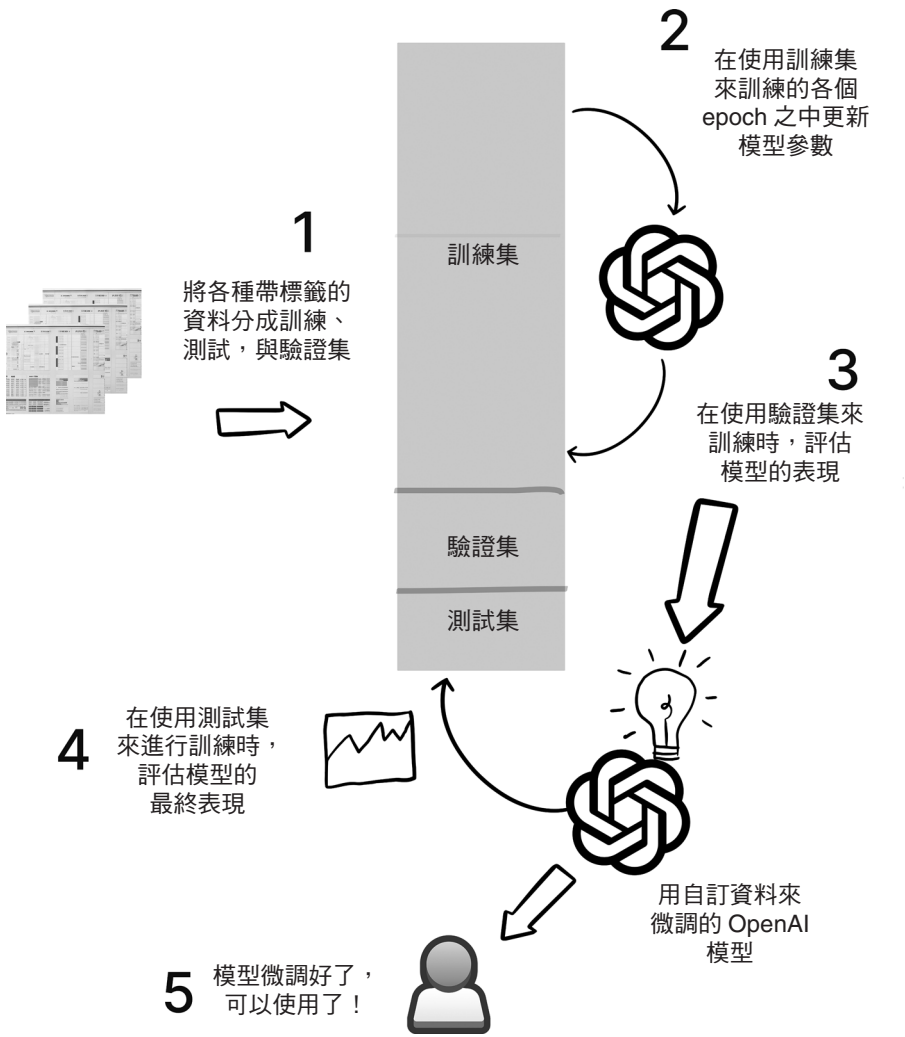

圖 5.2 微調程序。我們將資料集分成訓練、驗證，和測試集，用訓練集來更新模型的權重和評估模型，用驗證集在訓練過程中評估模型。然後用測試集來檢測最終模型，並使用一套標準來評估它。如果模型通過所有測試，那就將它投入生產，並監視它，以進行後續的迭代。

你可以採取幾個策略來避免提示充塞問題。首先且最重要的是，提示詞應簡潔具體，只包含 LLM 一定需要的資訊，讓 LLM 把注意力放在眼前的特定任務，並產生較準確的結果，以解決有待處理的所有問題。此外，你可以使用提示鏈，將多任務工作流程分解成多個提示詞（如圖 6.8 所示）。例如，你可以用一個提示詞來產生行銷計畫，然後使用該計畫作為輸入，要求 LLM 指出關鍵人物…等。

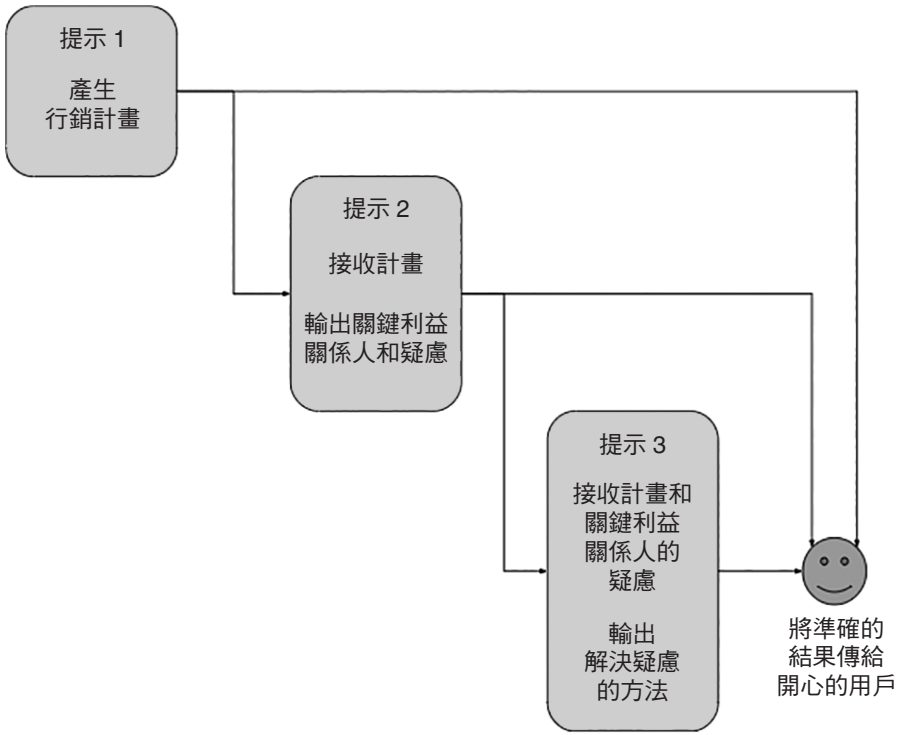

圖 6.8　可行的提示鏈流程。我們用一個提示詞來產生計畫，用另一個提示詞來產生利益關係人和疑慮，並且用最後一個提示詞來找出處理疑慮的方法。

提示充塞也可能對 GPT 的效能和效率造成負面影響，因為模型可能要花更久的時間來處理紊亂或過於複雜的提示詞，以及產生輸出。提供簡潔且條理分明的提示詞可以幫助 GPT 更有效率地執行任務。

範例：使用多模態 LLM 和提示鏈來防禦攻擊

假設我們想要建立一個 311 風格（譯註：311 是指美國 311 公共服務熱線）的系統，讓人們可以送出照片來報告他們社區的問題。我們可以串接多個 LLM，讓其中的每一個 LLM 扮演特定的角色，以建立一個全面性的解決方案：

- **LLM-1（圖像標題生成）：**用這個多模態模型為照片產生準確的標題。這種模型可以處理圖像，並提供其內容的文字敘述。
- **LLM-2（分類）：**讓這個純文字模型接收 LLM-1 產生的標題，並將問題分類為幾個預先定義的選項之一，例如「路面坑洞」、「路燈故障」，或「塗鴉」。
- **LLM-3（跟進問題）：**LLM-3（純文字的 LLM）根據 LLM-2 決定的類別來產生相關的跟進問題，以蒐集關於問題的更多資訊，並確保適當的行動被執行。
- **LLM-4（視覺問題回答）：**這個多模態模型與 LLM-3 合作，使用收到的圖像來回答跟進問題。它將圖像的視覺資訊與 LLM-3 的文字輸入結合起來，為每一個答案提供準確的答案及信心分數，讓系統優先考慮需要立即處理的問題，或將信心分數較低的問題提出來，供作業員進一步評估。

圖 6.9 將這個範例視覺化。你可以在本書的程式碼版本庫中找到這個範例的完整程式。

談到提示鏈，在現實世界裡，沒有任何單一提示技術可以決定提示詞的成敗，最佳成果通常要結合多種技術才能獲得。為此，在接下來的案例研究中，我們要結合你在介紹提示工程的各章學到的知識。

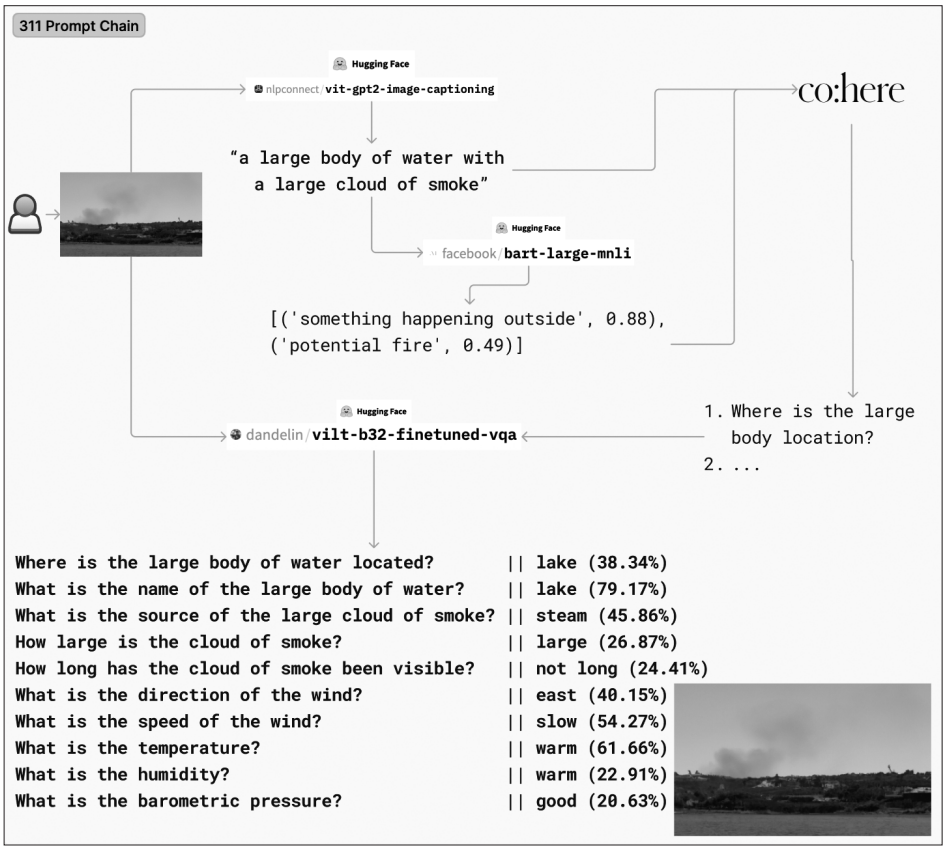

圖 6.9　多模態提示鏈，從左上方用戶提交圖像處看起，多模態提示鏈用四個 LLM（三個開源模型和 Cohere）來接收圖片、取得標題、分類、產生跟進問題，並回答它們與提供信心分數。

案例研究：AI 的數學能力有多強？

我們來重新探討 few-shot 學習和思維鏈提示這兩種技術，這兩種技術可以讓 LLM 在幾乎沒有訓練資料的情況下，快速適應新任務。我們已經在第 3 章看過使用這些技術的提示詞範例了。隨著 Transformer-based LLM 技術不斷進步，以及它被越來越多人用於各自的架構中，few-shot 學習和思維鏈提示已成為發揮這些尖端模型最大效能的關鍵方法，可讓 LLM 有效地學習，並執行比原始的 LLM

回想之前的語意搜尋範例，我們知道，許多現成的 embedding 模型都能夠保留語意相似性，包括 OpenAI 的閉源模型，以及開源模型。在本章中，我們將進一步探討這個概念，深入研究如何訓練自編碼的 LLM（比起「寫」，更擅長「讀」的模型），以有效地在 embedding 空間中捕捉其他的商業用例。藉此，我們將瞭解自訂 embedding 和模型架構的潛力，以便建立更強大、多功能的 LLM 應用程式。

案例研究：建立推薦系統

本章的大部分內容將探討 embedding 和 LLM 架構在設計推薦引擎時發揮的作用，並使用一個真實的資料集作為研究案例。我們將使用 OpenAI 的 embedding 模型以及開源模型。接下來要討論的 embedding 模型都是由 LLM 驅動的（但是並非所有 embedding 架構都是由 LLM 驅動的，本章會展示它們）。本章的目標是證明自訂 embedding 和模型架構對於實現更好的效能，以及針對特定使用情境量身打造結果時的重要性。

設定問題和資料

為了展示自訂 embedding 的威力，我們將使用 MyAnimeList 2020 資料集，該資料集可以在 Kaggle 上取得，它包含關於動畫標題、評分（從 1 到 10），和用戶偏好的資訊，提供了建立推薦引擎所需的豐富資料。圖 7.1 展示在 Kaggle 網頁上的資料集。

為了公平地評估推薦引擎，我們將資料集分成訓練集和測試集。和之前的章節一樣，接下來的結果都是讓模型處理保留下來的測試集產生的。透過這個步驟，我們可以使用一份資料來訓練模型，然後用模型未曾見過的另一份資料來評估它的效能，從而公正地評估模型是否有效。範例 7.1 是載入動畫標題，並將它分為最初的訓練部分和測試部分的程式。我們將訓練部分進一步分成訓練集和驗證集。

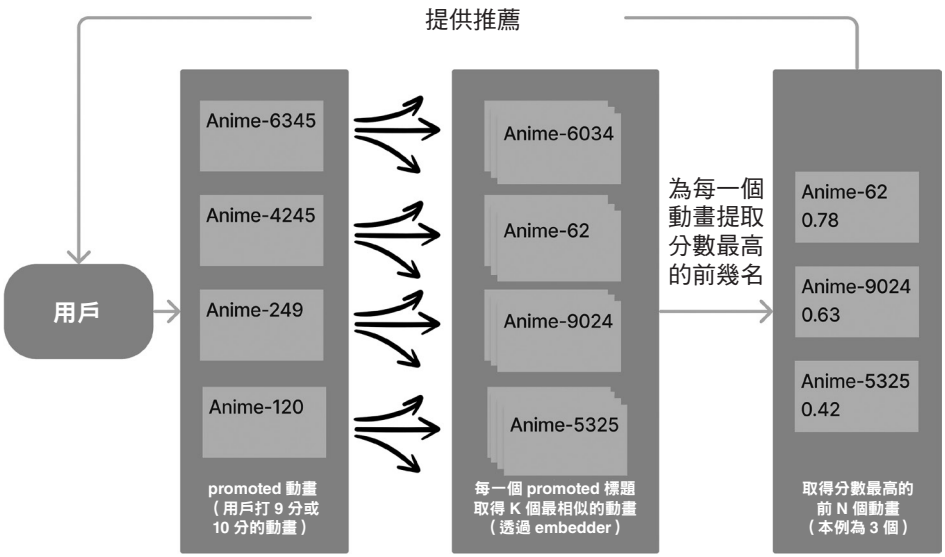

圖 7.3 第 2 個步驟接收用戶，並為用戶 promoted（打 9 或 10 分的）的每一部動畫找到 k 部動畫。例如，若用戶 promoted 4 部動畫（6345、4245、249 和 120），且 k = 3，系統會先找出 12 部語意相似的動畫（為每部 promoted 動畫找出 3 部，可重複），若動畫多次出現，則稍微提高該動畫的原始餘弦分數來排除重複。接著考慮 promoted 動畫的餘弦分數，以及它們在原始的 12 部動畫列表中出現的頻率，來選擇前 k 部不同的推薦動畫標題。

3. **為相關的動畫評分：**如果在步驟 2 中找出來的動畫不在測試集中，那就忽略它。如果在測試集裡有用戶給那部動畫打分數，則根據以下這些源自 NPS 的規則來為系統推薦的動畫指定分數：

   - 如果在測試集中，用戶將推薦的動畫評為 9 或 10 分，則將該動畫視為「promoter」，系統獲得 +1 分。
   - 若評分是 7 或 8，將該動畫視為「passive」，並獲得 0 分。
   - 若評分介於 1 到 6 之間，將該動畫視為「detractor」，並且獲得 -1 分。

這個推薦引擎的最終輸出是一個排行榜，裡面是用戶最可能喜歡的前 *N* 部（取決於我們想展示幾部給用戶看）動畫，以及系統處理事實（ground-truth）測試集的表現分數。圖 7.4 以高層的角度來展示整個過程。

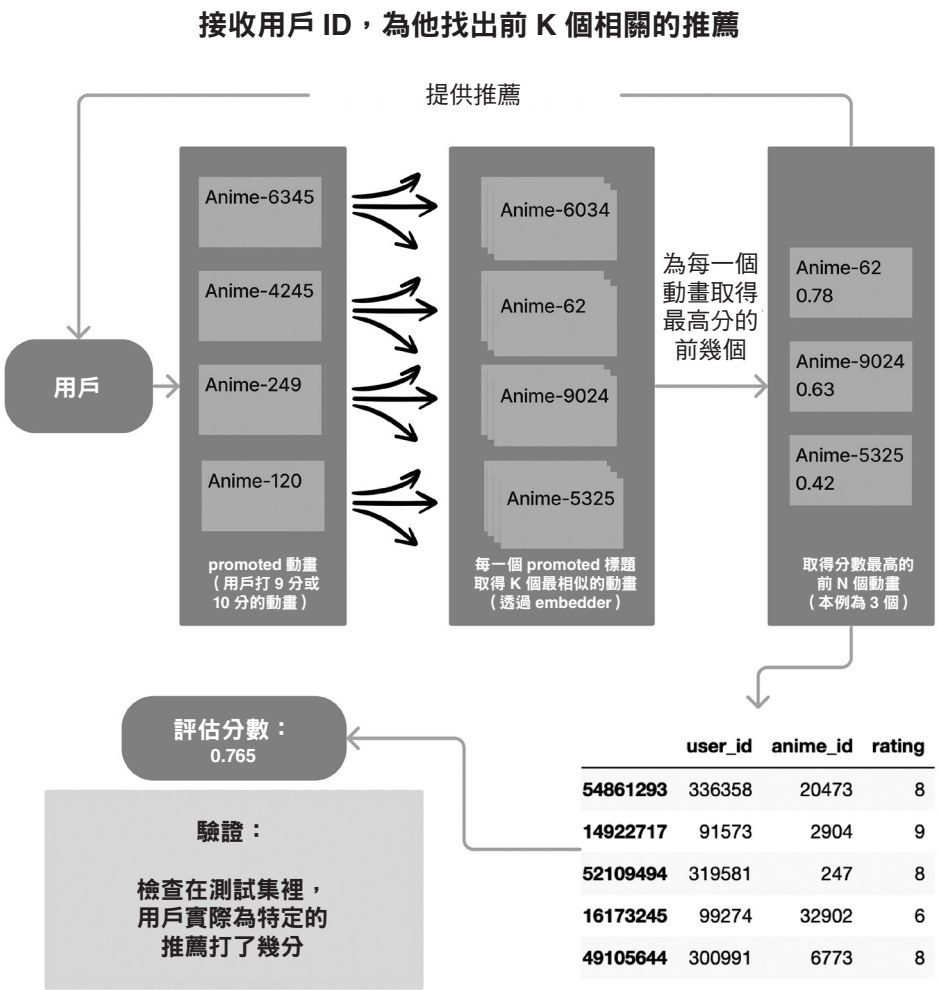

圖 7.4　整體推薦程序包括使用 embedder 及用戶 promoted 的標題來提取類似的動畫。然後，如果系統推薦的動畫可在評分測試集內找到，那就為推薦的動畫指定一個分數。