前言

我不知道被問過多少次：「LLM / AI 工程師與 LLMOps 工程師有什麼不同？」無論是在會議上、研討會中，或只是與業界人士一起喝咖啡，這都是一個被不斷問到的問題。

一開始，我經常從技術層面解釋這些角色的差異，但隨著時間過去，我意識到真正的問題在於：大家其實不太瞭解，要讓大型語言模型（LLM）在生產環境中長期穩定運作需要做哪些事情。

截至我在 2025 年初撰寫本書時，最先進的模型、技術與最佳做法幾乎每隔幾天就會更新。因此，真正理解 LLM 複雜性的人很少。多數人仍將**運作**（*operationalizing*，Ops）視為部署的同義詞，但在 LLM 的脈絡中，Ops 其實是協調人員、程序，與技術，讓模型在生產環境中保持安全、穩定、可靠。

各企業和他們的人力資源部門，正努力釐清這一切對於團隊與專案有什麼意義，我將在本書盡力回答這個問題。本書不是一本教導如何定義角色，或如何建構與部署 LLM 的書籍，雖然我們會涵蓋這些主題，但光是探討它們已不足以應對現實挑戰。一旦 LLM 應用程式進入生產階段，就必須有人持續優化它們，否則它們可能會變成一個過度設計（overengineered）的簡單問題解決方案，甚至淪為在高負載或提示注入攻擊之下崩潰的脆弱系統。

在傳統軟體開發（Software 2.0）中，你不會要求首席開發者既負責建構產品，又負責維護整個產品，你會讓軟體開發工程師負責建構，可靠性工程師負責維護。建構與維護 LLM 同樣需要這種職責分工。在 Software 3.0 時代，LLM/AI 工程師負責建構，而 LLMOps 工程師負責維護。

雖然機器學習運作（MLOps）是 LLMOps 的基礎，但工程師從結構化資料與判別式模型（discriminative models）中學到的 MLOps 技能，無法全部套用到生成式模型上。

簡言之，我寫這本書，是為了幫助你瞭解整個 LLM 應用程式生命週期的獨特之處，包括資料工程、模型部署、API 設計，再到監視、安全機制，與資源優化。我希望讓你具備堅實的基礎，在建構、維護，與優化 LLM 的資料、模型與應用程式時，能做出明智決策。

本書編排慣例

本書所使用的格式慣例如下：

斜體字（*Italic*）

表示新術語、URL、電子郵件地址、檔案名稱，與副檔名。中文以楷體表示。

等寬字體（`Constant width`）

表示程式碼，或在文字段落中的程式元素，例如變數、函式名稱、資料庫、資料型態、環境變數、陳述式，與關鍵字。

致謝

我要感謝 Lucas Meyer 的大力支持，他的想法促成本書的多個章節。感謝編輯 Nicole 與 Sarah，在緊迫的期限內協助我完成這本書；以及技術審閱 Lalit Chourey、Ammar Mohanna、Nirmal Budhathoki，感謝他們提供的寶貴回饋。

最重要的是，我要感謝家人——謝謝你們包容我對本書的執著，並為我準備源源不絕的茶水。最後，致讀者——無論你是已經在這個領域深耕多年，還是才剛踏入，希望這本書能加快你的成長速度。

第四章

LLM 領域的資料工程

在這一章，我們將學習資料工程、資料管理方法，以及資料庫工具與系統。本章的對象是想要成為 LLMOps 工程師，或希望領導資料工程任務的資料、DevOps 與 MLOps 工程師。看完本章後，你將瞭解資料工程的基本知識，以及 LLM 的最佳做法。

資料工程與 LLM 的興起

Edgar F. Codd 是英國電腦科學家，他在 1960 年代末，完成關於自我複製電腦的博士研究之後，進入 IBM 工作。Codd 對資料結構理論有濃厚興趣，並在 1970 年於 IBM 發表一篇名為「A Relational Model of Data for Large Shared Data Banks」（*https://oreil.ly/JG1bn*）的內部論文，該論文提出今日眾所周知的**關聯式資料庫**。關聯式資料庫將資料分別儲存在多個相關的表格中，例如一個顧客表、一個產品表與一個銷售表，而不是將全部的產品與顧客資訊都儲存在銷售表內的每一筆紀錄中。在關聯式資料庫出現之前，像「變更顧客地址」這種簡單的動作都必須修改該名顧客的所有銷售紀錄，對主機來說，這是昂貴的操作。但是在關聯式資料庫中，你只要更新顧客紀錄，所有相關紀錄便會自動更新。

雖然這篇論文在 IBM 內部並未立即引起關注，卻吸引其他電腦科學家與愛好者的注意，包括 Oracle 創辦人 Larry Ellison，他開發並銷售了第一個與 IBM 主機相容的關聯式資料庫。IBM 也開發了一種用來查詢資料庫的語言，最初稱為 SEQUEL，後來更名為 Structured Query Language（SQL），並成為業界標準。Codd 在 1981 年憑藉關聯式資料庫的研究獲得圖靈獎（電腦科學界的最高榮譽）。隨著關聯式資料庫的普及，以及管理系統需求的增加，IBM 在 1983 年推出自家的資料庫管理系統 DB2。關聯式資料庫逐漸成為產業標準，被廣泛用於索引、編目…等用途。IBM 與 Oracle 將負責維護企業資料庫系統的人稱為**資料庫管理員**（*database administrators*），簡稱 DBA（**資料工程師**這個頭銜是在 2010 年代雲端運算興起時才流行起來的）。

後來，Codd 與別人一起撰寫了另一篇論文「Providing OLAP to User-Analysts: An IT Mandate」（*https://oreil.ly/gUwKl*），並首次提出 *online analytical processing*（OLAP）這個術語，它是指一種能夠快速處理與查詢多維度資料的系統。OLAP 是今日多數資料處理系統的基礎。

Tim Berners-Lee 在 1990 年創造全球資訊網（World Wide Web），導致大家產生與記錄的資料量呈指數級成長。在這些資料中，有相當一部分是結構化的，也就是具備固定的長度與型態，例如郵遞區號，但也有大量非結構化資料，它們長度與型態不一，例如音樂、文章，與影片。關聯式資料庫會將資訊組織成表格，那些表格具有預先定義的欄位與嚴格的資料型態。由於表格內的每一列資料都必須使用相同的結構，關聯式資料庫非常擅長處理高度結構化的資料，並支援複雜的、基於 SQL 的查詢，能將多個表格連接起來，同時維持一致的 ACID（原子性、一致性、隔離性、持久性）保證。因此，關聯式資料庫是交易系統（例如銀行、庫存，與傳統商業應用）的首選，特別是在需要資料完整性與跨表關聯的情況下。但是關聯式資料庫不適合網際網路上常見的非結構化資料。

非關聯式（NoSQL）資料庫的目的，是為了處理關聯式系統無法有效率地處理的工作，例如龐大、快速變動或結構鬆散的資料集。鍵值儲存體（key-value store）藉著將不重複的鍵與資料配成一對，來提升查詢速度。文件型資料庫（document database）是鍵值儲存體的一種特化形式，它將每筆紀錄存為獨立的 JSON 文件，允許每個文件擁有不同的結構。這種彈性非常適合內容管理系統、產品目錄，以及不同的紀錄之間，有極為不同的欄位的其他領域，這種資料在網際網路中相當常見。鍵值資料庫也能以二進位大型物件（binary large objects，簡稱 *blob*）的形式儲存影片與圖像等二進位檔案，使其成為新資料環境的理想選擇。

此外，我們還有向量資料庫與圖資料庫。**圖資料庫**（*graph database*）專門儲存互相關聯的實體，這種資料庫不是使用表格或文件來儲存資料，而是使用節點與邊，從而支援毫秒級的路徑搜尋查詢，例如社交網路中的「朋友的朋友」搜尋、供應鏈影響分析，或探索文件之間的關係。

向量資料庫的用途是儲存與檢索高維度的 embedding。embedding 是能夠捕捉文字、圖像、音訊，或其他內容之語義意涵的密集數值向量。在檢索它們時，資料庫並非尋找完全相同的紀錄，而是利用**近似最近鄰**（*approximate nearest neighbor*，ANN）演算法，在多維空間中，找出與查詢向量最接近的項目。所以它們是驅動語義搜尋、推薦系統、圖像及音訊相似度比對的引擎，也是能在幾毫秒內為 LLM 提供相關脈絡的**檢索增強生成**（*retrieval-augmented generation*，*RAG*）作業線的核心機制。

力，並將他們整合進資料團隊；或是聘請資料工程師，將他們整合進 LLM 開發團隊，並培訓他們成為 LLMOps 工程師。

無論如何，整個領域從「專為特定任務訓練 ML 模型」轉向「訓練不限於特定任務的 ML 模型」這個趨勢都會持續發展下去。龐大的資料市場會持續成長，與 LLM 有關的公司需要聘請專業人才，以管理他們的資料工程系統。

DataOps 工程師的角色

DataOps 工程師通常具備資料工程師或資料科學家背景，並具備處理「領域構成（domain composition）」、「資料規模」與「大規模資料品質」…等複雜問題的專業知識。他們熟悉進階技術，例如「全域重複資料刪除」與「動態資料選取」，可用來持續微調模型時。

LLM 領域的資料工程涉及設計、開發、管理資料作業線和基礎設施，以支援模型的訓練、評估，與部署。DataOps 工程師負責實作並優化擴展法則（scaling laws），平衡「資料品質」與「數量」兩者，並管理多樣且大規模的資料集。然而，他們的職責除了管理資料作業線之外，還要協調整個 LLM 的資料生命週期（從取得資料開始一直到部署），在高度複雜且不斷演變的環境中，持續提升模型效能。

這種專業化象徵著資料工程與管理實務的重大演進，資料工程師必須在日常工作中採取更精密、更具針對性的做法。在 LLM 時代之前，資料工程的重心是透過作業線，將定義明確、結構化的資料從作業系統移至資料倉儲（data warehouses）或資料湖（lakes），以便在報表中或分析時使用。當時的重點是將 ETL/ELT 工作批次化、維度建模（dimensional modeling）、緩慢地改變維度，以及一些治理工作，例如根據資料是否符合結構定義，來決定資料的品質、參照完整性（referential integrity），以及基本的重複資料移除。非結構化文字資料可能被封存於資料湖中，但很少被視為一等公民；搜尋與分析工作仍以列、欄，及聚合 SQL（aggregate SQL）為核心運作。

以 LLM 為核心的作業徹底改變了一切。如今的原始資料包含異質的文字、程式碼、圖像、音訊，與聊天記錄，其價值取決於**語義豐富度**（也就是內容的資訊價值）而非嚴格的結構。資料作業線必須針對這些內容執行斷詞、分段、embedding 與版本控制，將它們儲存在向量索引中，以支援相似度搜尋；並濾除個資、有害內容，及根據授權來篩選。團隊不再執行傳統的 ETL 工作，而是持續運行資料提取與重新 embedding 循環，以保持 RAG 系統的即時性，並記錄每一次的提示詞與回應，以利後續評估輸入與輸出，進而改進系統效能。在這種情境下，資料品質的評估標準包括真實性、事實準確度，與偏差

指標，這些屬性需要透過自動化紅隊測試與人機互動（human-in-the-loop，HITL）來處理，而不是像過去那樣，只需要檢查資料結構是否違反規定。

因此，現代資料工程技術結合了傳統的資料倉儲、物件儲存、向量資料庫，與特徵儲存庫。像 Airflow 與 Dagster 這類的流程協調框架與 LLMOps 工具互相依存，而資料治理的範疇也往外拓展，涵蓋模型卡、資料集營養標籤（dataset nutrition labels），以及追蹤每個詞元的來源譜系，直到找到其合法的原始來源。因為必須支援 LLM 的工作，資料工程已經從「處理列與欄的底層工作」轉變為「語言與知識的擁有者與守護者」。

資料工程直接影響機器學習應用程式與 LLM 的效能。訓練模型時使用的資料的品質、類型與數量，決定了模型的成敗。在 LLM 領域有兩個額外的挑戰。第一，LLM 使用的資料幾乎都是非結構化資料，第二，資料量多非常多。這兩項差異，使許多任務更加困難。舉例來說，在傳統機器學習中，你可以檢查輸入資料中的異常值，例如移除年齡為負數或超過 130 歲的紀錄。但在非結構化資料中，這類檢查難以實施，導致 LLM 的資料管理任務比非生成式 ML 模型的資料工程更加複雜。

資料管理

資料管理著重管理組織的資料資產，而**資料工程**則關注設計與建立資料儲存、處理與分析的基礎設施。有效率的 LLM 資料工程團隊必須同時擁有 DataOps 工程師（負責管理資料）與資料工程師（負責設計與管理資料作業線）（見圖 4-1）。這兩個角色必須一起整合多樣化的資料來源，協助 LLM 更有效地學習，並減少幻覺與偏差等問題的發生。

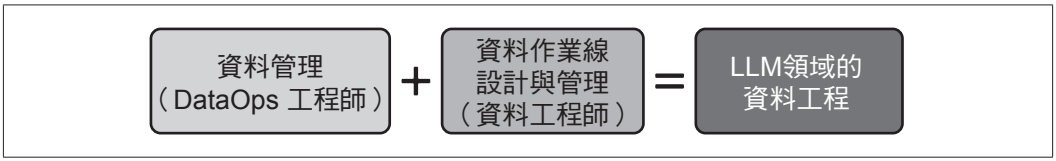

圖 4-1　LLM 資料工程的組成要素

在 LLM 中管理資料的基本方法有兩種：靜態與動態。**靜態資料管理**是在整個訓練過程中保持資料集不變，這可能導致重複的資料，且無法隨著模型需求的變化而調整。**動態資料管理**則是在訓練過程中，持續更新與調整資料，這種方法比較靈活，但比較有挑戰性，因為它需要持續監視資料的品質與相關性。

有些方法會在訓練過程中動態調整資料集。例如，**動態資料修剪**（*dynamic data pruning*）會隨著訓練的進展，而移除比較無用的範例；而**二元分類器**（*binary classifiers*）可用來判

斷模型依照指令運作的程度，以決定是否提前停止訓練。其他技術還有選擇資訊量最高的任務，以及藉由迭代的過程精煉任務內容。

合成資料

在筆者撰稿時，較新的模型（如 Microsoft Phi-4 與 DeepSeek-R1）已證明了使用**合成資料**可提升效能。所謂的合成資料就是用現成的資料來自動生成有相同統計特性的資料。例如，假設一個資料集有 100 位真實籃球員的身高、位置與得分紀錄，我們可以利用統計方法來生成許多類似但現實不存在的虛構球員資料，以擴增資料集。為了合成訓練 LLM 所需的長篇文字，DataOps 工程師經常使用上一代的文字生成模型來產生這類資料。

如前所述，在組合任務（task composition）時，關鍵在於平衡資料的數量與品質。較大的資料集通常意味著更多樣且品質更好的資料，通常能提升模型效能，但也需要高效率的資料處理作業線。為了建構強大的模型，DataOps 工程師必須精通調配（orchestration），也就是能在適當的時機，自動套用這些技術。

LLM 的作業線

那麼，傳統機器學習與 LLM 有哪些差異？為什麼要使用不同的作業線？

如前所述，在傳統機器學習中，你要處理的，主要是**結構化資料**，它們是整齊排列在表格或試算表內的數值，通常來自資料庫、感測器或 API，乾淨、可管理且清晰。傳統機器學習高度依賴**特徵工程**，資料工程師會將原始資料轉換為有用的形式，提取數值特徵以供模型預測，這是一項需要真人參與的過程，人的經驗與判斷在過程中至關重要。

在多數情況下，你處理的資料集規模較小，不需要龐大的資料量，並可使用傳統 CPU 或 GPU 來處理，這種做法高效、可控，特別適合任務明確的情境。當問題明確、資料是結構化的，而且界限清楚時，傳統機器學習仍是預測與分類任務的首選。

然而，在 LLM 的情境下，我們處理的全是**非結構化資料**，它們是雜亂、龐大、未整理的文本，例如文章、程式碼、社群媒體貼文。資料來源包括網路、文件庫，與文字 API。它們是資訊洪流，比傳統 ML 的乾淨試算表混亂得多。現實世界的資料本質上是非結構化的。舉例來說，若要使用新聞網站的資料來訓練模型（注意，這個用途必須取得授權），打開幾個新聞網站時，你會發現除了文章本身之外，裡面還夾雜著廣告、圖像、補充說明框、相關新聞區塊、編輯推薦清單…等多種元素，而且每一個部分的格式都不同。此外，新聞網站本身是龐大的資料集，裡面有大量文字，必須使用強大的圖形處理單元（GPU）甚至是專用的張量處理單元（TPU）才能處理。這是大規模的資料，你的運算能力必須跟得上它。

最後一項差異在於，傳統機器學習模型有明確的效能指標，例如 precision 與 recall，但 LLM 必須生成擬人內容，要判斷輸出是否具備人類特徵，通常必須由真人來審查。相較於使用結構化的資料，在處理這種任務時，你要做更多實驗。例如，在使用結構化資料時，你可以移除離群值或錯誤樣本，來迅速提升模型效能，這意味著有一些類型的資料明顯比其他資料更有價值。根據目前的研究狀況，我們仍然無法確定哪些資料類型最有價值，適合用來訓練 LLM。例如，如果描述錯誤資訊的句子有很好的語句結構，它也可能有幫助。LLMOps 工程師除了必須處理非結構化資料，以及滿足更高的運算需求外，還要做各種實驗，以改進輸入資料，並評估期望的輸出結果。

訓練 LLM

訓練 LLM 的過程大致可分為兩個階段：預訓與指令微調（見圖 4-2）。**預訓**階段的目標是讓模型學習語言的一般規則與知識，包括文法、句法、風格與領域知識。這個階段發生在你要求模型遵循指令，或執行特定任務之前。預訓通常使用遮蔽法：取一段資料、隱藏其中一個詞，然後訓練 ML 模型預測那一個詞。這個階段的目標是將模型的猜詞錯誤率最小化。**微調**階段則是提供一組複雜的指令與期望的答案，訓練模型將答案的誤差最小化。與傳統機器學習一樣的是，資料越好，結果越佳，因此本章剩餘的內容將專門討論維護資料工程品質的策略。

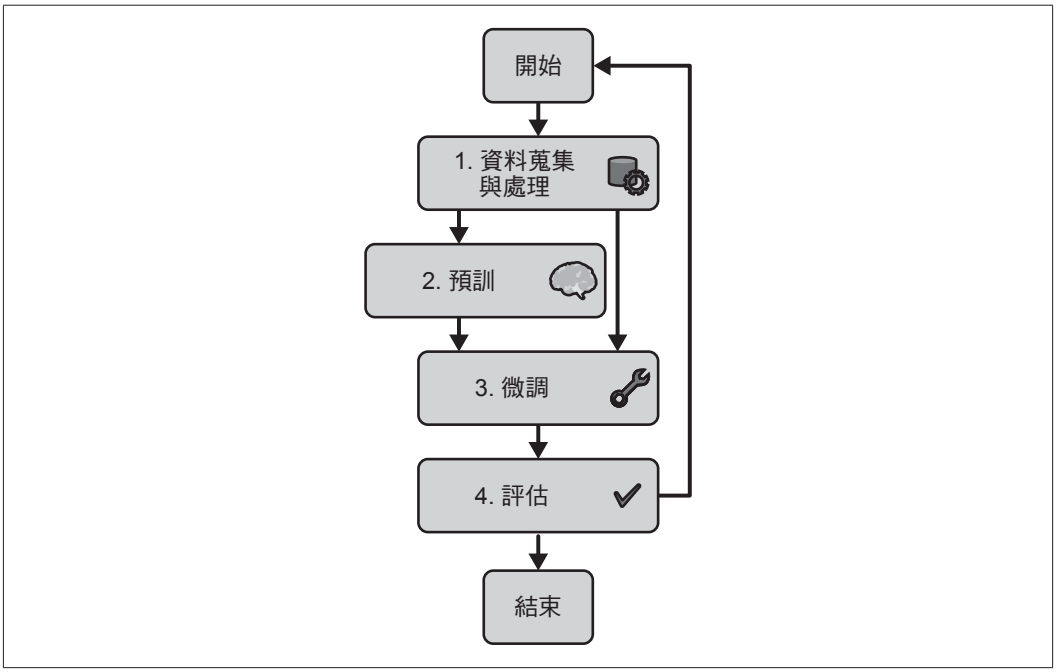

圖 4-2　LLM 訓練作業線範例

原始的資料工程生命週期

圖 4-3 是傳統 ML 團隊的資料工程生命週期（*Data Engineering Lifecycle*，DELC）。DELC 包含五個階段，它會將原始資料轉化為可供分析師、資料科學家、ML 工程師…等使用的成果。因此，在 LLM 出現之前，資料工程師的職責主要是開發與維護資料作業線，並確保資料品質。

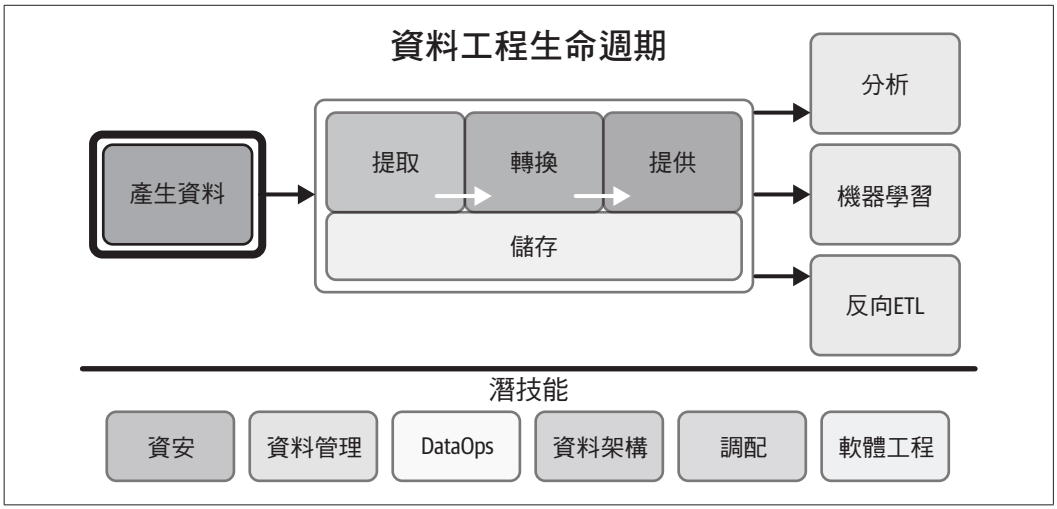

圖 4-3　資料工程生命週期（來源：Fundamentals of Data Engineering）

當時，DELC 的五個主要成分如下：

產生資料

在這個階段中，你要與產生資料的團隊及流程合作。例如，若資料來自 API 或問卷，工程師要和負責開發 API 或問卷的團隊合作，以確保資料品質良好。若有需要，這個階段也包括製作合成資料。

資料提取

此階段包含蒐集資料，並將它們傳送到適當的資料儲存系統中。

資料儲存

工程師將資料整合至資料湖，並儲存在資料庫中。

資料轉換

這包括資料清理，也就是處理異常值、遺漏的資料，與重複的資料。

提供資料

讓最終使用者或資料科學團隊可以使用轉換過的資料。

在資料工程領域浮現的新問題

為 LLMOps 建構 DELC（見圖 4-4）需要回答許多新問題，但其中的許多問題在資料工程的文獻與實務中仍未探討。

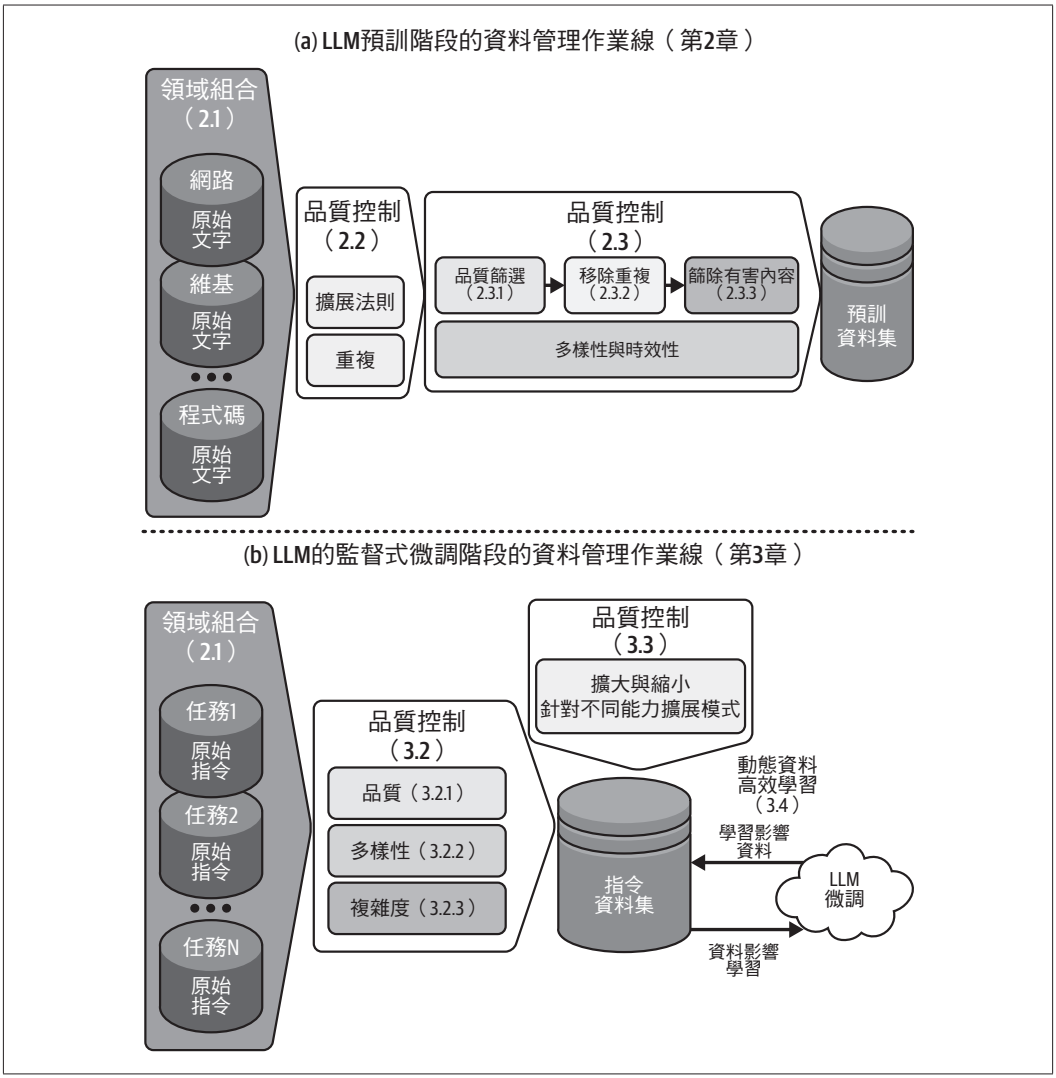

圖 4-4　LLM 的資料管理作業線（改編自 Wang 等人於 2023 年製作的圖表（*https://oreil.ly/o9BYJ*））

本章將藉著探討以下議題來逐步建立一個新的 DELC：

- 對你的 LLM 應用程式而言，理想的資料組合是什麼？
- 哪種擴展法則（scaling law）適合你的情境？
- 你可以接受多少重複資料？
- 你要採用哪些技術來篩選資料品質？
- 應該使用哪些模型來移除重複資料？
- 應該如何處理附帶有害內容或偏見的資料？
- 你需要多大的資料多樣性？
- 在建立合成資料時，該如何設計最佳提示詞？
- 如何監視資料老化？

如果你不熟悉其中的一些術語也不要緊，下一節會詳細解釋它們。事實上，在本章的後半部分，你將學習如何以方法論思考這些問題。不過，在此之前，我們來一一討論這些新興的議題。

資料組合

被公開的訓練資料集通常由多個領域的資料構成。混合多領域方法在 LLM 中相當普遍。早期的訓練語料庫來自高度多樣的資料來源，例如網頁與維基百科，這些資料因為廣泛的覆蓋範圍與多樣性而備受重視。然而，隨著資料品質越來越重要，對於更專業、更高品質資料的需求也日益增加，例如書籍與學術論文。之所以有這個轉變，是因為語言模型需要更強大的任務處理能力，與更高階的語言表現力。

較新的研究（*https://oreil.ly/nXFWr*）指出，同時使用程式碼與非結構化文字來訓練的 LLM，能夠更正確地處理非結構化任務。事實證明，LLM 從相對結構化的程式碼資料學到的能力，也能遷移至非結構化任務，與此同時，LLM 的程式碼生成能力也會明顯提升，如今軟體工程師與資料科學家已廣泛使用 GitHub Copilot 等工具。這些創新意味著，近年來，程式碼與數學文本等領域已開始佔據訓練資料更大的比重。

從這一個趨勢來看，隨著時間推進，訓練資料集之中的領域範圍會愈來愈廣，以賦予 LLM 更豐富且更強大的能力。有一項研究指出，若要訓練出類推能力強大的模型，採用適當混合許多領域的訓練資料集非常關鍵（*https://oreil.ly/5WJNK*）。

在筆者撰稿時，研究人員正致力於開發在預訓階段決定最佳領域混合比例的方法。早期的工作主要結合實驗與直覺推理進行，近期的發展則採用自動化的技術來分配領域權重，並建立合適的目標分布（*https://oreil.ly/13j5J*）。

擴展法則

早在 LLM 被廣泛採用之前，研究人員就已經非常關注訓練資料集的規模與 transformer 語言模型效能之間的關係了。訓練模型的焦點是將損失（loss）最小化，損失就是模型出錯的比率。例如，當模型預測「I was very thirsty so I drank⋯」這句話的下一個詞時，它預測「bookshelf」的損失，將遠大於預測「water」的。Kaplan 等人於 2020 年的研究（*https://oreil.ly/h2AGb*）指出，語言模型的損失遵循一種冪次定律關係（一個量會隨著另一個量的固定冪次成比例變化）。例如，球體的體積與半徑的立方成冪次關係。對語言模型而言，損失與訓練資料集大小或模型大小（即參數數量）呈冪次定律關係，前提是兩者皆非瓶頸，且訓練計算資源充足。

這種關係可以用數學來表示為：

$$\text{損失} \propto \left(\frac{1}{N}\right)^{\alpha} \quad \text{或} \quad \text{損失} \propto \left(\frac{1}{D}\right)^{\beta}$$

其中：

- N 是模型大小
- D 是訓練資料集大小
- α 與 β 是常數，依模型與資料集的特定條件而定

Kaplan 和共同作者的研究結論指出，只要同時擴增模型大小與訓練資料集規模，模型的損失就會以可預測的方式下降。此外，他們建議，若計算資源充足，為了維持最佳效能，模型規模與訓練資料集規模應大致以相同的速率擴張。

此外，他們固定計算預算 C，分析最佳資源分配策略，發現最適訓練資料集大小與最適模型大小之間，應該有以下的關係：

$$D_{\text{opt}} \propto C^{0.27} \quad \text{和} \quad N_{\text{opt}} \propto C^{0.73}$$

LLM 通用資料預處理作業線

以下介紹的作業線包含 10 個基本步驟。不過，在開始執行它們之前，你還要做「第 0 步」：定義如何衡量成功，也就是在完成所有步驟後，如何判斷流程是否有效，以及新版本是否勝過先前的版本？這些評估技術會在第 7 章詳細說明，但在此，我先介紹一個通用的基本衡量方法。

首先，建立一組可在每次作業線執行後計算的核心指標。最簡單的做法是準備一組你已經知道正確答案，而且可快速評估的提示詞，例如，從簡單問題（如「2 + 2 等於多少？」、「法國的首都是哪裡？」）到較複雜的問題（如「請以是或否回答：這是一張鳥圖片嗎？」或「請以是或否回答：Tom Cruise 是 Mary Lee Pfeiffer 的兒子嗎？」），這些可以迅速生成的答案，就像警報器一樣，可以在你的資料蒐集、重複資料移除、權重設定…等步驟出問題時，讓模型效能異常無所遁形。

此外，應定期評估 LLM，使用傳統的效能評測，如《Massive Multitask Language Understanding》（MMLU，第 7 章），以及你用來檢查安全性與偏差的提示詞（第 8 章）。雖然結果有一些波動是正常的，但在訓練過程中，你要確保整體效能呈現上升趨勢，而不是停滯在低點，甚至在修改某個資料預處理步驟後，出現明顯下滑。

知道衡量方法後，我們來逐步介紹資料預處理的 10 個步驟。

第 1 步：建立資料目錄

在開始之前，你要瞭解你實際需要哪種類型的資料。你的最終目標是什麼？你將如何應用模型？這些答案能夠引導你選擇合適的預訓練資料。及早定義資料型態、語言、領域、品質標準，能幫助你設定清晰的目標，而不是為了蒐集而蒐集。在資料庫內組織資料來源，以便日後為所蒐集的資料加上標籤。

第 2 步：檢查隱私與法規遵循

接著，確保你遵守資料隱私法與相關法律規範。這一步不僅是為了保護自己，更是為了尊重資料，以及那些資料所代表的人。務必取得資料的合法授權，再將它納入資料目錄，並在資料庫中記錄授權資訊，以便日後能用來標註所蒐集的資料。

第 3 步：過濾資料

並非所有資料都是等質的，資料來源的品質至關重要。雖然你可從多種來源蒐集資料，例如網站、書籍與學術論文，但務必確保它們符合你事先定義的標準。可靠且精確的資料是關鍵所在。例如，CulturaX 語料庫（*https://oreil.ly/ZJqyL*）使用黑名單機制來濾除有害內容，這是很好的風險防範案例。你也可以使用專門的過濾工具或雲端解決方案，在初期避免低品質來源。此時，你可以直接捨棄未使用的資料，或僅在資料庫中標註它已被濾除，第二種做法會佔用更多儲存空間，但方便你日後新增或移除過濾條件。

資料清理：幾項核心原則

在清理資料時，遵循以下原則可幫助你維持明確的方向，並確保高品質：

- 評估句子的完整性。如果句子不完整（包括缺少標點符號，或語意不通），都要濾除。不完整的句子沒有使用價值。
- 移除私人可識別資訊（PII）。你可以完全刪除，或是用佔位文字取代。保護隱私是首要原則。
- 刪除有害內容，例如暴力、色情，或其他不當資訊。這不僅是過濾問題，更是責任問題。
- 移除異常符號。不屬於內容或突兀的符號都是資料裡的雜訊。
- 移除技術性雜訊，如 HTML、CSS 或 JavaScript 標記。這些內容沒有實質用途，只會干擾主要文本。
- 刪除包含大括號的句子，它們通常是佔位符或垃圾資料。
- 移除太短的句子。雖然簡潔有時有益，但太短的句子往往缺乏前後脈絡或語意。沒有實質價值的內容都要排除。
- 移除多餘內容，如導覽列或「讚」按鈕。這些無關文字只會讓資料變得雜亂。
- 移除包含沒必要的特定詞彙的文本。總有一些詞彙或片語不該出現，請事先定義，並在資料中全面移除。

第 4 步：移除重複資料

資料蒐集是否成功取決於策略的明確性。你要投入多少時間？資料的範圍多大？多久蒐集資料一次？事先想好這些問題的答案，能在你維持應用程式即時特性的同時，幫助你蒐集多樣化的資料。此時雲端平台特別有用，因為它能提供可隨需擴展的資料管理能力。你同樣可以選擇直接捨棄重複資料，或標註它。

移除重複資料的方法

在移除重複資料時，有幾種主要的方法值得關注：

Term frequency–inverse document frequency (TF-IDF) 軟性去重複

比較詞語在文本中出現的頻率，以及它們在整體資料集中出現的頻率。如果兩段文字的 TF-IDF 分數高度相似，那就刪除其中一段。如果某些詞語在特定文件中頻繁出現、但在整體語料中不常見，它們將獲得較高權重，代表它們對該文件有關鍵意義。

MinHash

這一種演算法可估算兩組文字的相似程度。它用隨機雜湊化來產生一組最小雜湊值，並比較它們來估計相似度。這種方法的計算與儲存效率都很高，特別適合大型資料集。

SimHash

這種演算法將文字特徵向量轉換為固定長度的雜湊碼，並比較這些雜湊碼之間的距離來衡量相似度。雜湊碼越相近，代表文字越相似。這是一種可靠、輕量的文字相似度計算方法。

除了上述方法之外，還有許多其他去除重複的方法，其中一種簡單、有效的方法是刪除連續出現的重複句子，只保留第一個實例。你也可以移除資料集中具有相同 URL 的文件。另一個常見方法是使用 *n*-gram 與 MinHashLSH，當相似度超過一個門檻（通常是 0.8 左右）時，便將它標為重複內容。

這些方法各有優勢，但它們的最終目標一致：移除多餘、重複、無關的資料，讓資料保持精簡與聚焦。資料越乾淨，模型的表現就越好。

第 5 步：蒐集資料

從這一步開始就是實際的工作了。使用網頁爬蟲、API 或其他工具，從已確定的來源蒐集資料（務必檢查服務條款，並且避免侵犯版權）。無論是使用 HTML 剖析，還是 PDF 文字擷取，都要確保資料乾淨且結構化。有一種常見做法是使用整理過的資料集，如 Falcon（*https://oreil.ly/-5wkX*） 或 CommonCrawl（*https://oreil.ly/TaXZi*）。CommonCrawl 提 供 WARC（Web ARChive）格式的資料（*https://oreil.ly/-kH_a*），其中包含整個網頁的原始資料，以及 WET（WARC encapsulated text）格式，只保留網頁正文的純文字。即使你建立自己的資料集，遵循這些格式仍然有益，因為你可以利用 GitHub 上的各種工具來處理這類格式的檔案。

在蒐集資料的同時，將前幾個步驟建立的中繼資料一併加入。

第 6 步：偵測編碼

你一定要使用正確的編碼，錯誤的編碼會破壞資料，且編碼錯誤難以發現，它們看起來很正常，有時，它們甚至只會影響文字中的部分字元。你可以使用編碼偵測工具，例如開源工具 Chardet（*https://oreil.ly/V7Ub0*），來確保文字檔案以正確的編碼方式處理。如此可在模型輸出結果之前偵測並修正錯誤。請將編碼資訊加入第 5 步蒐集的資料的中繼資料中。

第 7 步：偵測語言

接下來，使用語言偵測工具（如 lingua-py，*https://oreil.ly/0jVgJ*）來辨識資料的語言，然後根據語言，將資料分成子集合。瞭解訓練資料的語言十分有用，你也可以檢查 LLM 是否具備足夠的語言代表性，以滿足應用需求。例如，在建立能夠使用葡萄牙語的 LLM 時，你必須在訓練時使用充足的葡萄牙語資料。將語言資訊加入中繼資料中。

第 8 步：資料分段

蒐集原始資料，並確保能夠用正確的編碼和語言來讀取它們之後，接下來要將資料拆為可用的部分。提取文字元素，並將它解析成可管理的資料段落。大多數模型能夠處理的文字長度都有上限，因此在這個步驟中，應將輸入文字拆為小於該上限的資料段落。

分段的方法有很多：

- 分成固定大小的段落很簡單，但可能會將同一個概念切成不同段落。
- 觀念清晰、內容分明的文件適用以句子為單位來分段。

- 根據段落來分段可以保留更完整的語意脈絡，但各段的大小通常大於句子分段。

此外還有更進階的文字分段技術。例如，你可以利用現有的 LLM 為每個分段添加中繼資料，讓模型評估該分段的情緒傾向，並判定主題，甚至可以將整份文件傳給現有 LLM，要求它回傳分段結果。代理式分段是更高階的技巧（*https://oreil.ly/ZETMR*），它是將文件上傳至既有的 LLM，建立一個代理來模擬一位詢問關於文件的問題的分析師，並記錄最常被引用的分段。

注意，使用 LLM 來分段可能會耗用大量的計算資源。無論採用哪種方法，在各段裡，都要一併儲存前幾步的所有中繼資料，例如資料來源、授權、編碼，與語言。

第 9 步：備份資料

這一步看起來很簡單，但定期備份是關鍵的安全防護動作。資料遺失可能造成嚴重損失，定期備份能確保在出問題時，仍有可恢復的資料版本。

第 10 步：執行維護與更新

最後，這不是一次性的流程。資料蒐集系統必須持續維護。定期更新資料來源與改善蒐集策略，可確保資料保持新鮮與相關。持續改進是關鍵所在。

這些步驟可用來產生預訓與微調階段所需的原始預處理分段。第 5 章將詳細說明 LLM 的訓練過程。為了讓你瞭解快速生成微調資料的方法，接下來要介紹向量化。

向量化

向量化就是將文字資料轉換為高維度數值表徵（即**向量**）的過程，用數值表徵來捕捉資料的關鍵特徵。動詞的 *embedding* 經常與 *vectorizing* 交替使用，而名詞的 *embedding* 則是指生成的向量本身。

生成向量後，你可以將它儲存在向量資料庫中。向量化早在 LLM 普及之前就已被廣泛應用了。例如，ElasticSearch 資料庫系統早在 2019 年就支援向量功能，讓使用者能以文字查詢相似內容。現今的向量資料庫則將向量儲存與搜尋能力擴展至龐大的資料集規模。

embedding 程序（亦稱 ***embedding 模型***）最理想的特性是能有效捕捉語意差異。優秀的 embedding 模型會在文字或片語語意相近時，生成彼此接近的向量，在語意不同時，生成彼此距離較遠的向量。

indexing。當使用者送出文字查詢時，查詢的內容也會被向量化，然後資料庫會用**最近鄰搜尋**（*https://oreil.ly/oq2Du*）來計算查詢句向量與所儲存的向量之間的距離，進而在向量空間中，找到與查詢句最相似的項目。

向量資料庫可以一起儲存中繼資料、你的資料，和它的 embedding 向量，讓你可以在執行向量相似度搜尋前，先縮小搜尋範圍，從而大幅提升查詢效率，特別是在使用大型資料集時。例如，倘若你已經知道查詢句的語言了，而且中繼資料有語言欄位，你可以利用這個資訊來提升搜尋效率。

在選擇向量資料庫時，除了成本外，你還要考慮隨需擴展性、容錯能力、索引技術的可用性…等因素，詳見表 4-1。

表 4-1　向量資料庫的選擇

特性	說明	常用指標	索引技術（影響）
隨需擴展性	處理不斷增長的資料量與查詢負載的能力	吞吐量（每秒查詢數） 延遲（查詢執行時間） 儲存容量	水平擴展能力（影響吞吐量） 分片（sharding）策略（影響查詢效率）
容錯能力	在發生故障時持續服務的能力	運作時間百分比 恢復時間（MTTR）	資料複寫（確保資料可用性） 高妥善率（HA）功能（將停機時間最小化）
indexing 技術	在高維向量空間中提升搜尋效率的方法	搜尋準確度（提取相關向量的能力） 搜尋速度	Metric trees（HNSW、VP-Tree）：適合中等規模的資料與維度的高效搜尋法 雜湊化（LSH）：加快近似最近鄰搜尋速度（犧牲準確度） Inverted file（IVF）：若能結合多個中繼資料篩選條件，可提升搜尋速度

維持資料的新鮮度

資料預處理作業線第 10 步的重點是讓資料維持最新狀態。讓索引與底層資料在即時改變時保持同步並不容易，具體做法從「在我查詢時，再告訴我有沒有變動」到「一旦有變動就立刻通知我」形成一個連續光譜。**輪詢**是最簡單的方法：定期查詢來源資料，並比對最新快照與資料庫內的現有內容，這種方法容易理解，適合資料量小，或即時需求不高的情況，但浪費資源，因為即使資料未改變，也會重複查詢。此外，它會引入與查詢頻率相等的延遲，例如，如果每年更新一次資料的話，至少要等一年才能偵測到變更。

比較精確的替代方案是**變更資料擷取**（*change data capture*，CDC），它會直接讀取資料來源的交易紀錄、提交日誌，或與資料新鮮度相關的中繼資料（例如文件的最後修改日期）。與其在整個文件集內搜尋差異，不如直接讀取有變動的文件清單，並且只更新這些文件。CDC 仍屬於資料擷取方法的一種，但能避免盲目比對，並減少因為無意義的檢查而浪費的頻寬。

如果資料生產端能主動推送資訊，我們就進入事件驅動或串流領域了。在**事件驅動**的更新模式中，資料擁有者會發送訊息，例如「產品描述已變更」或「文章已更新」。每一個事件都是自成一體的，可立即觸發資料庫更新。

LlamaIndex

LlamaIndex（*https://oreil.ly/owRAr*）（最初在 2022 年底以 GPT Index 名稱推出）是一個開源資料框架，可讓你將資料連接至任何 LLM 工作流程。它負責處理繁瑣的底層作業，包括：從多種格式載入資料、分段、向量化、建立索引，與提取向量 embedding。

你可以使用 LlamaIndex 來預先處理資料、生成向量表徵，使用 CDC 或事件驅動更新機制來建立一個獨立的作業線，用來擷取即時資料變更，讓作業線在有資料異動時，觸發向量再生成，並使用 API（若支援）來更新向量資料庫。

生成微調資料集

你只要執行資料預處理小節介紹的 10 個步驟就能獲得足夠的資料來預訓 LLM 了。然而，在幾乎所有的實際應用中，LLM 必須具備的功能不是只有「預測被遮蓋起來的字詞」而已。LLM 最常見的任務是回答問題，為此，你要提供一份包含問題與答案的清單，這類資料稱為**指令資料集**（instruction dataset）或**微調訓練資料集**（fine-tuning training dataset）。

建立微調訓練資料集的主要方法有四種：

人工製作

這是親力親為的方法。你和團隊親自策劃與設計資料集，根據需求仔細挑選並撰寫每個指令。雖然這種做法既耗時，又需要大量人力，但你有最大的控制權，特別適合需要高度專用的，或針對特定任務調整的資料集。

蒐集並改良現有的開源資料集

如果有現成的高品質資料的話，何必從零開始製作？你可以從開源資料集擷取內容，進一步精修與改良以符合需求。這是一條不犧牲品質的捷徑，結合策略性優化還會有更好的效果。微調現有資料能讓你善用社群的集體成果，加速開發進程。

使用 *LLM* 來自動生成

你可以使用 LLM 來生成指令微調資料集。為此，你要先將分段資料儲存在向量資料庫中。下一節會詳細說明這個過程。

混合方法

最後，不要忽略結合各種方法的優勢。你可以兼用人工、開源資料、模型，全面覆蓋各種需求。最簡單的做法是將資料集結合起來，做成綜合資料集。這種混合技巧非常靈活，能根據任務特性，善用各種策略的優點。

微調資料集可分為兩大類。**通用型指令微調資料集**（*general instruction fine-tuning dataset*）涵蓋多個領域與多樣任務，目標是提升模型遵循一般指令的能力，讓它更通用。資料集涵蓋的範圍越廣，模型理解與執行多樣指令時的表現就越好。另一類是**領域專用型指令微調資料集**（*domain-specific instruction fine-tuning dataset*），它針對特定領域設計。例如，醫學指令資料集可用來訓練模型執行診斷或醫療輔助任務。聚焦特定領域能有效提升模型在該領域中的專業表現。

自動生成指令微調資料集

第 *1* 步：預處理與向量化

我們建議使用 LlamaIndex（*https://oreil.ly/1uMXc*）來處理大型文字語料庫。LlamaIndex 能執行上述的一般作業線中的多個預處理步驟，例如分詞與清理，並可為語料中的每個分段生成高品質的向量表徵。這些文件向量可儲存在多種不同的資料庫中。

第 *2* 步：建立檢索機制

設計一個簡單的程式，讓它根據輸入的問題，從向量資料庫取出最相關的分段。你可以使用 LlamaIndex 內建的 `VectorIndexRetriever` 功能來做這件事。

第 *3* 步：生成問題

將文件（而非分段）傳給現有的 LLM，要求它生成可以用文件來回答的問題。你可能要將文件分成多個部分，但每個部分可以遠大於資料庫中的分段。例如，單一分段通

常約有 8,000 個字元（以便生成 embedding），而 GPT-4o 則能為包含 400,000 字元的文件生成問題清單。

第 4 步：讓既有的 LLM 判定最佳答案

將第 3 步生成的每一個問題傳給第 2 步的程式。程式會回傳與問題最有關的分段清單。接著，將問題與該清單一起傳給現有 LLM，要求它選出具備最佳答案的分段，並據此生成清楚且完整的回答。你可以要求 LLM 以 JSON 格式輸出結果，例如 `{"instruction": <question>, "input": "", "output": <answer>}`。重複這個循環，直到獲得所需數量的樣本。

在生成問題與答案後，你必須執行基本的品質檢查，你可以計算文字層面的餘弦相似度來去除幾乎相同的問答、濾除引用了取出來的段落未提供之資訊的答案、人工抽查部分隨機樣本，以確保自動過濾規則正常運作。如果你想讓資料集更豐富，可提示 LLM 為同一個段落提出後續問題，或要求它以指定格式（如有效的 JSON）回傳答案。這兩種策略，都可以訓練「已微調的模型」處理更複雜的指令。

如果你的資源有限，你可以簡化整個作業線。在處理小型語料庫時，你可以使用 BERT 這類的輕量程式庫來預先計算 embeddings，略過檢索步驟，讓 LLM 用單一段落來生成問題和答案，並使用餘弦相似度來驗證答案是否與原始文本維持接近。

如果你的資料不是靜態的呢？

雖然這個問題沒有一體適用的方法可以處理，但以下是處理動態資料的建議：

- 整合即時資料流。使用低延遲訊息傳遞協定（如 Kafka 或 Apache Pulsar）來提升資料傳輸效率。
- 根據資料更新頻率，將資料分成時間區間（脈絡窗口），並在中繼資料中加入時戳，讓 LLM 知道資料的時效。例如，你可以製作一個包含已存在一週的資料的資料集、一個已存在幾個月的，以及一個包含歷史資料的資料集。你可以將它們放在同一資料庫中，但使用不同的中繼資料標籤來區分。

為不同時戳的資料維護獨立的訓練作業線可以節省再訓練的成本。例如，你可以每天使用已存在一週的資料集來重新訓練 LLM，並且每週使用已存在幾個月的資料集來訓練一次。

管理資料的版本，以追蹤各個 LLM 資料迭代，並在需要時，輕鬆恢復為先前的版本。