KM RAG 系統設計說明

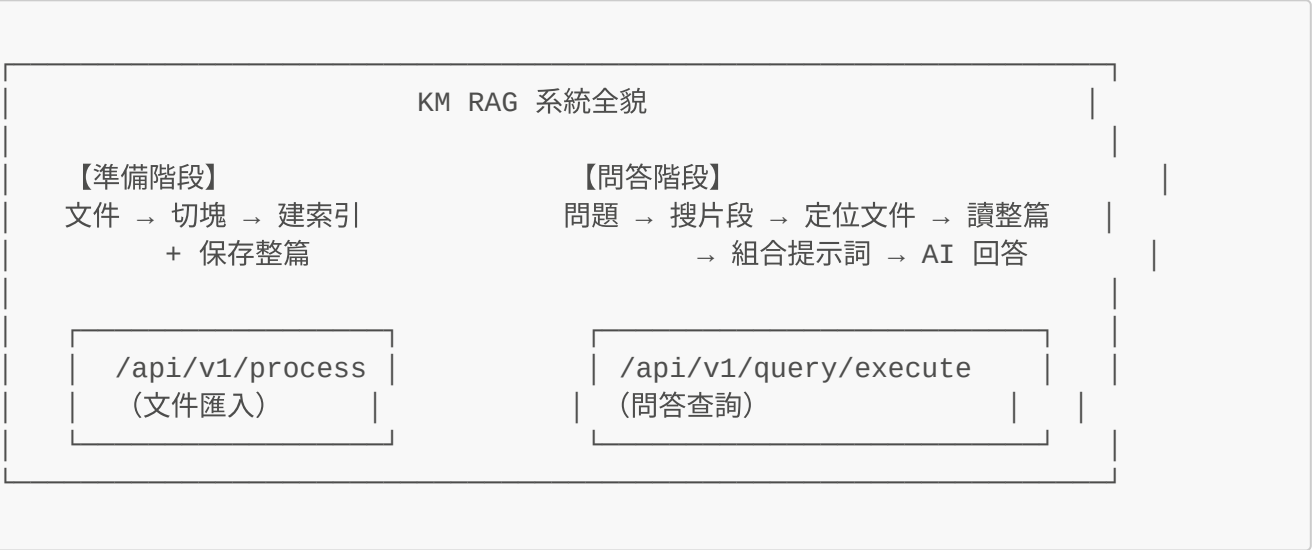
> 本文件以概念角度，圖解說明 `/api/v1/query/execute` 背後的 RAG（Retrieval-Augmented Generation，檢索增強生成）系統如何運作。

一、什麼是 RAG？一句話解釋

> **先從知識庫「找資料」，再把找到的資料「餵給 AI」，讓 AI 根據這些資料回答問題。**

傳統的 AI 只能靠自己訓練時學到的知識回答，容易「幻覺」（瞎編）。RAG 的核心思路是：**回答之前先查書**。

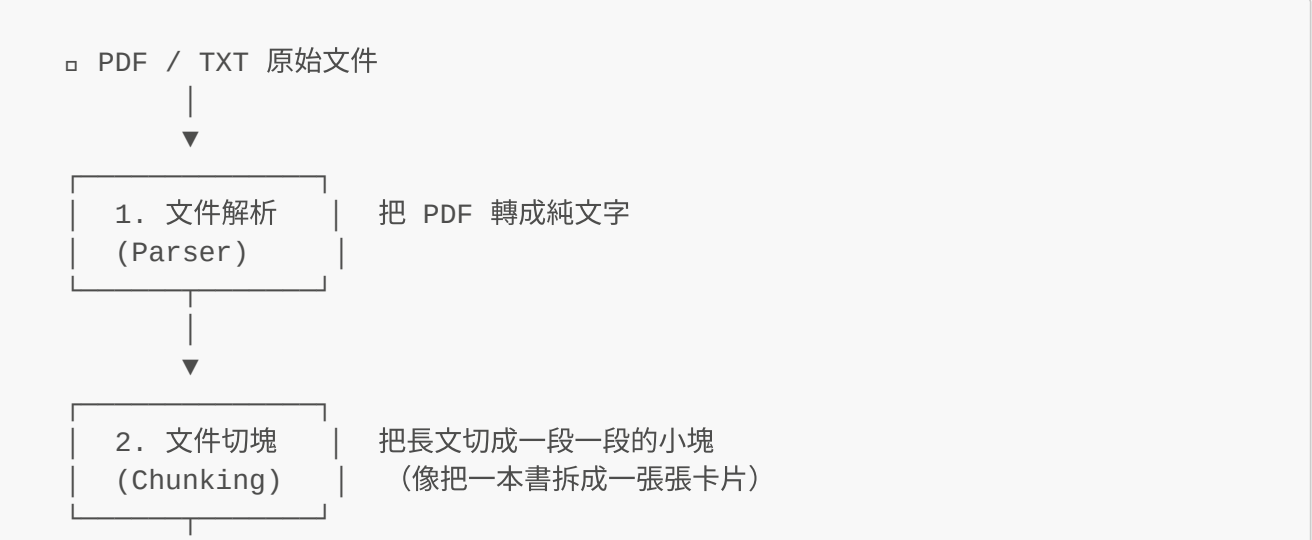
二、全局流程鳥瞰

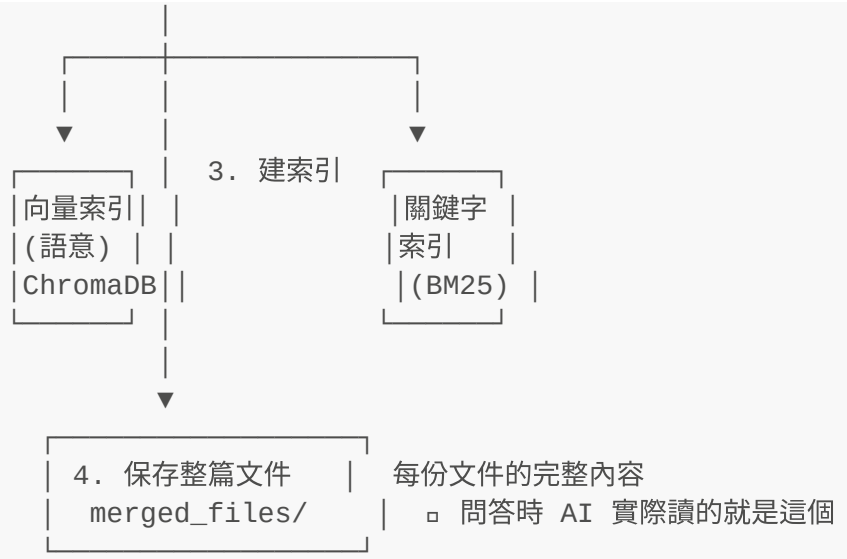


系統分為兩大階段：**準備階段**（把文件變成可搜尋的知識庫）和**問答階段**（用知識庫輔助 AI 回答）。

三、準備階段：把文件變成「AI 能搜尋的知識庫」

概念流程圖





每個步驟在做什麼？

步驟	做什麼	生活比喻
文件解析	把各種格式（PDF、Word 等）轉成純文字	把手寫筆記打成電子檔
文件切塊	把整份文件切成適當大小的段落	把一本書的內容做成一張張索引卡
向量索引	用 AI 把每段文字轉成「數字向量」，理解語意	幫每張卡片標記「這張在講什麼概念」
關鍵字索引	建立傳統的關鍵字搜尋索引（BM25）	幫每張卡片標記「這張出現了哪些詞」
保存整篇文件	每份文件的完整內容存入 `merged_files/`	把**原本的書也放在書架上**，方便之後整本拿出來給 AI 讀

對應的 API

```
POST /api/v1/process
├── 輸入：collection 名稱 + 檔案清單
├── 回傳：task_id（非同步處理）
│
└── GET /api/v1/status/{task_id}    ← 查詢處理進度
    └── 回傳：pending → processing → completed
```

四、問答階段：`/api/v1/query/execute` 的完整流程

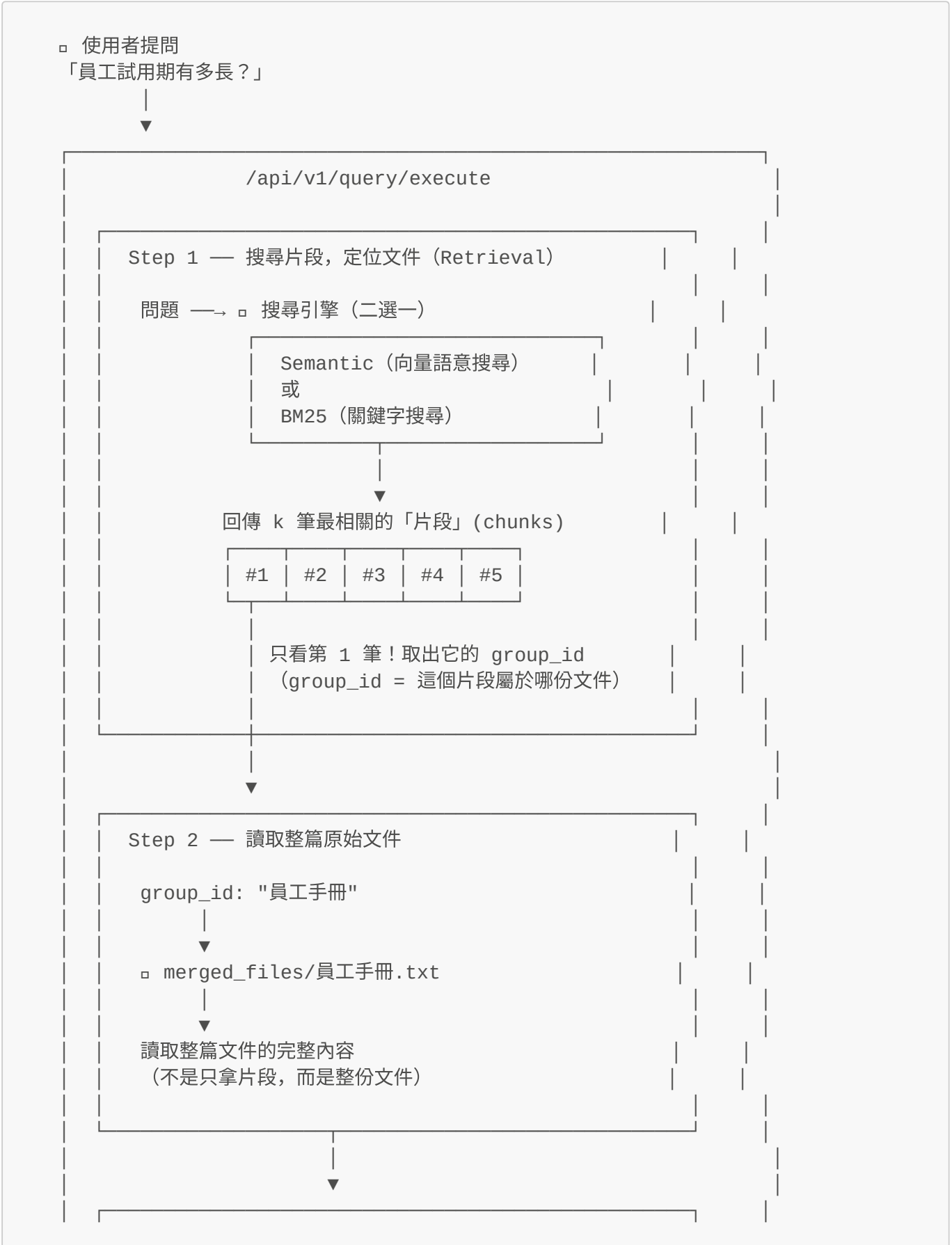
這是測試檔案重點測試的 API，也是整個 RAG 系統的核心。

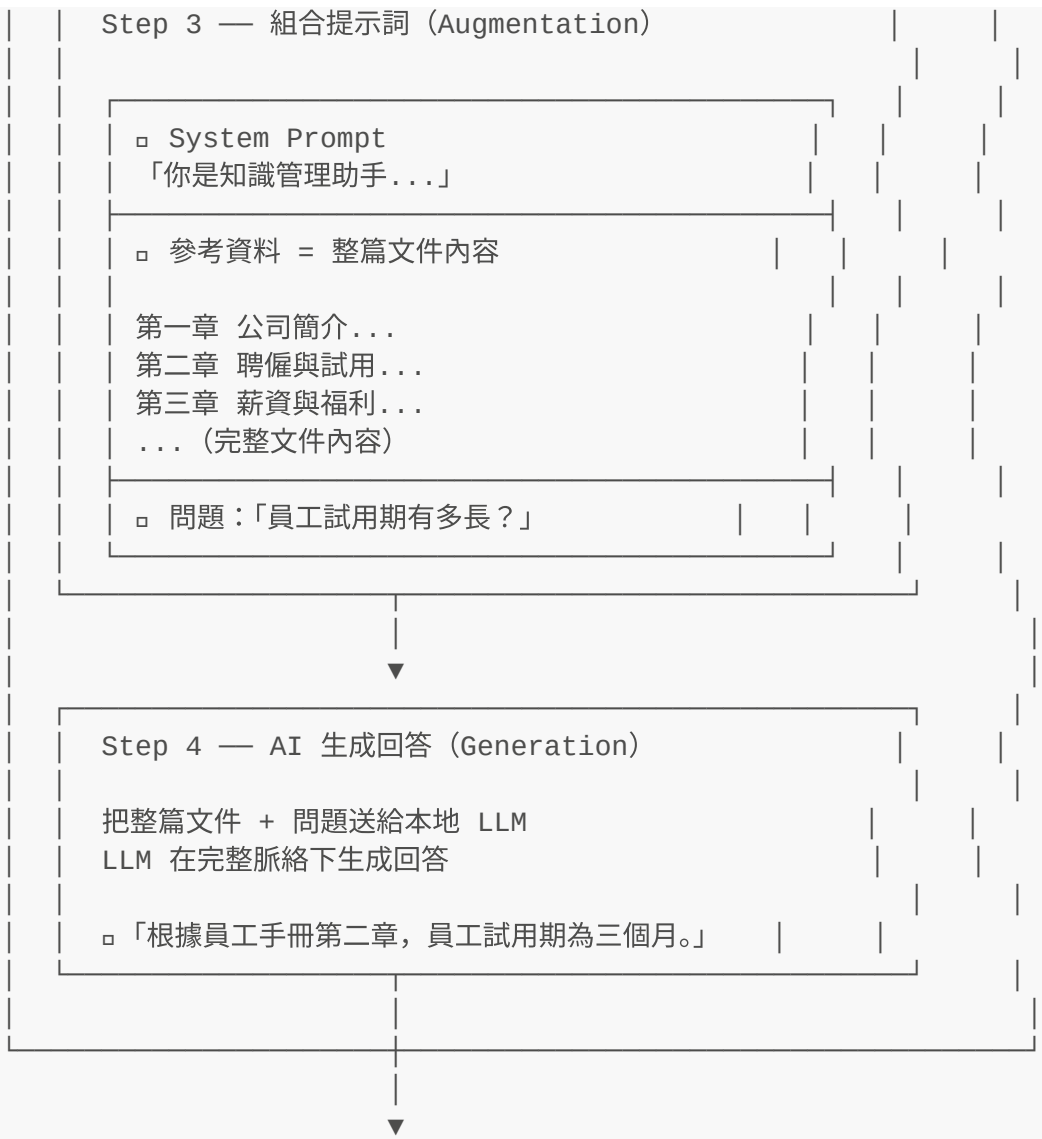
核心設計理念：「用片段找文件，用整篇餵 AI」

> 一般的 RAG 系統會把搜到的「片段」直接拼給 AI。但這個系統不一樣——**搜尋片段只是為了「定位是哪份文件最相關」**，定位到之後，會去讀取那份文件的**完整內容**，整篇餵給 AI。

為什麼這樣設計？因為片段可能斷章取義，把整篇文件給 AI，AI 才能理解完整脈絡。

概念流程圖





四個步驟的生活比喻

想像你是一位**圖書館員**，有人來問你問題：

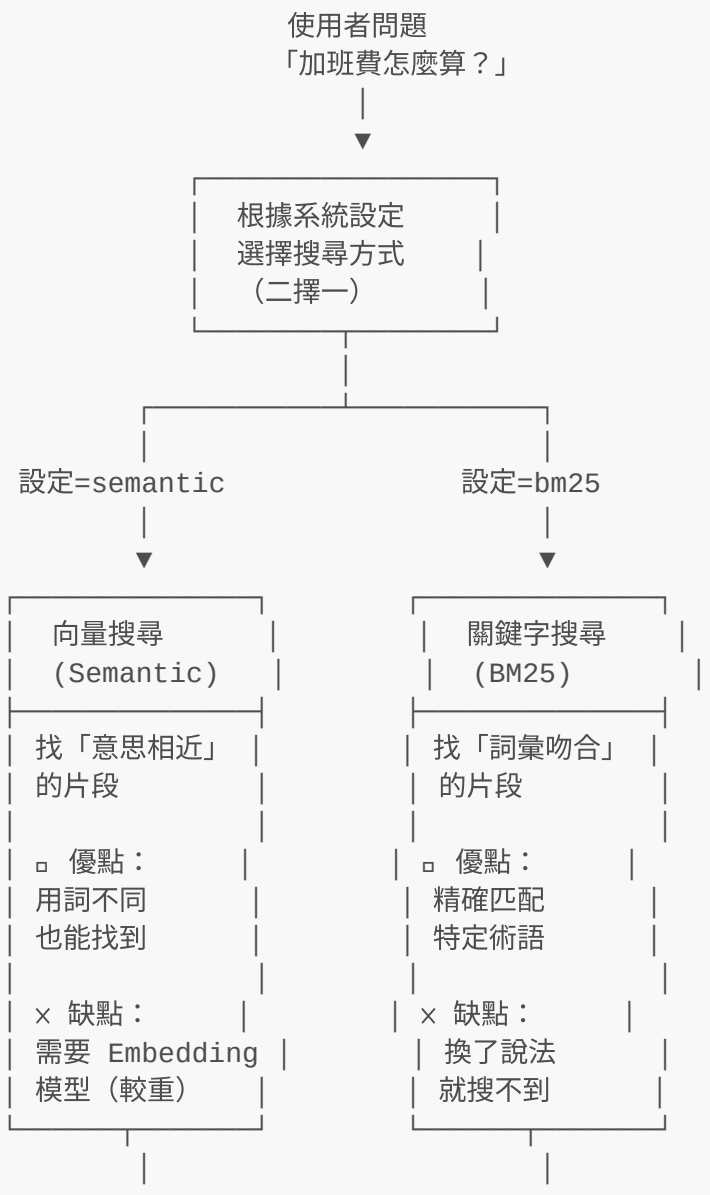
步驟	圖書館員做的事
Step 1 搜尋片段	用索引卡片搜尋，找到「員工手冊」這本書最可能有答案
Step 2 讀取整篇	不只看索引卡片，而是**把整本書從書架上拿下來**
Step 3 組合提示	把整本書攤開在桌上，旁邊放上問題
Step 4 AI 回答	讓 AI 讀完整本書後，回答問題

為什麼不只給片段，而要給整篇？

```
┌──────────────────────────────────────────────────────────────┐
│                                                              │
│   一般 RAG（給片段）              本系統 KM RAG（給整篇）      │
│                                                              │
│   「...試用期為三個月...」         ▫ 完整員工手冊             │
│   「...試用期滿需評核...」          （第一章到最後一章）       │
│   「...未通過可延長...」                                      │
│                                                              │
│   × 片段之間可能斷章取義           ▫ AI 能看到完整脈絡        │
│   × 遺漏未被搜到的相關資訊         ▫ 不會漏掉同份文件的資訊   │
│   ▫ 省 token、速度快               × 消耗較多 token           │
│                                                              │
└──────────────────────────────────────────────────────────────┘
```

五、搜尋引擎：二擇一機制（Semantic 或 BM25）

系統提供兩種搜尋演算法，但每次查詢**只會使用其中一種**（由設定檔決定），不是兩者合併：



```
  └────────────────────────┘
              │
              │
              ▼
  回傳 k 筆結果 (預設 k=5)
              │
              ▼
  只取第 1 筆的 group_id
  → 定位到哪份文件最相關
```

注意：搜尋回傳 k 筆片段，但系統**只關心排名第 1 的那筆**屬於哪份文件。搜尋的目的不是收集片段，而是**找出最相關的那份文件**。

六、測試怎麼驗證這個系統？

測試腳本 `test_rag.py` 用「黃金問答」（標準答案）來驗證整個流程：

```
┌──────────────────────────────────────────────────────────────┐
│                          端對端測試流程                          │
│                                                              │
│  Step 0    檢查服務是否啟動                                    │
│    │       GET /health                                       │
│    ▼                                                         │
│  Step 1    準備測試資料                                        │
│    │       PDF → TXT (用 pdfplumber 轉檔)                     │
│    ▼                                                         │
│  Step 2    載入標準答案                                        │
│    │       解析「黃金問答」檔案                                  │
│    │       ┌──────────────────────────────────┐              │
│    │       │ Q: 員工試用期有多長？               │              │
│    │       │ A: 三個月                          │              │
│    │       │ 來源: 員工手冊.pdf                  │              │
│    │       └──────────────────────────────────┘              │
│    ▼                                                         │
│  Step 3    匯入文件建立知識庫                                   │
│    │       POST /api/v1/process                              │
│    │       → 等待處理完成                                      │
│    ▼                                                         │
│  Step 4    逐題測試問答                                        │
│    │       ┌──────────────────────────────────────────────┐  │
│    │       │ 對每一題：                                      │  │
│    │       │                                              │  │
│    │       │  POST /api/v1/query/execute  ←── 核心測試      │  │
│    │       │  {                                           │  │
│    │       │    collection_name: "test_chinese",          │  │
│    │       │    query: "員工試用期有多長？",                  │  │
│    │       │    k: 5,                                     │  │
│    │       │    language: "zh-TW"                         │  │
│    │       │  }                                           │  │
│    │       │            │                                 │  │
│    │       │            ▼                                 │  │
```

```
│    │       │  取得 model_response                           │     │
│    │       │            │                                   │     │
│    │       │            ▼                                   │     │
│    │       │  模糊比對：AI 回答 vs 標準答案                    │     │
│    │       │  （bigram 重疊率 >= 40% → PASS）                  │     │
│    │       └──────────────────────────────────────────────┘     │
│    │                                                            │
│    ▼                                                            │
│  Step 5   輸出報告                                               │
│           通過: 18/20 (90.0%)                                    │
└─────────────────────────────────────────────────────────────────┘
```

七、答案驗證機制：模糊比對

測試不是要求 AI 回答**一模一樣**，而是用「模糊比對」來判斷是否正確：

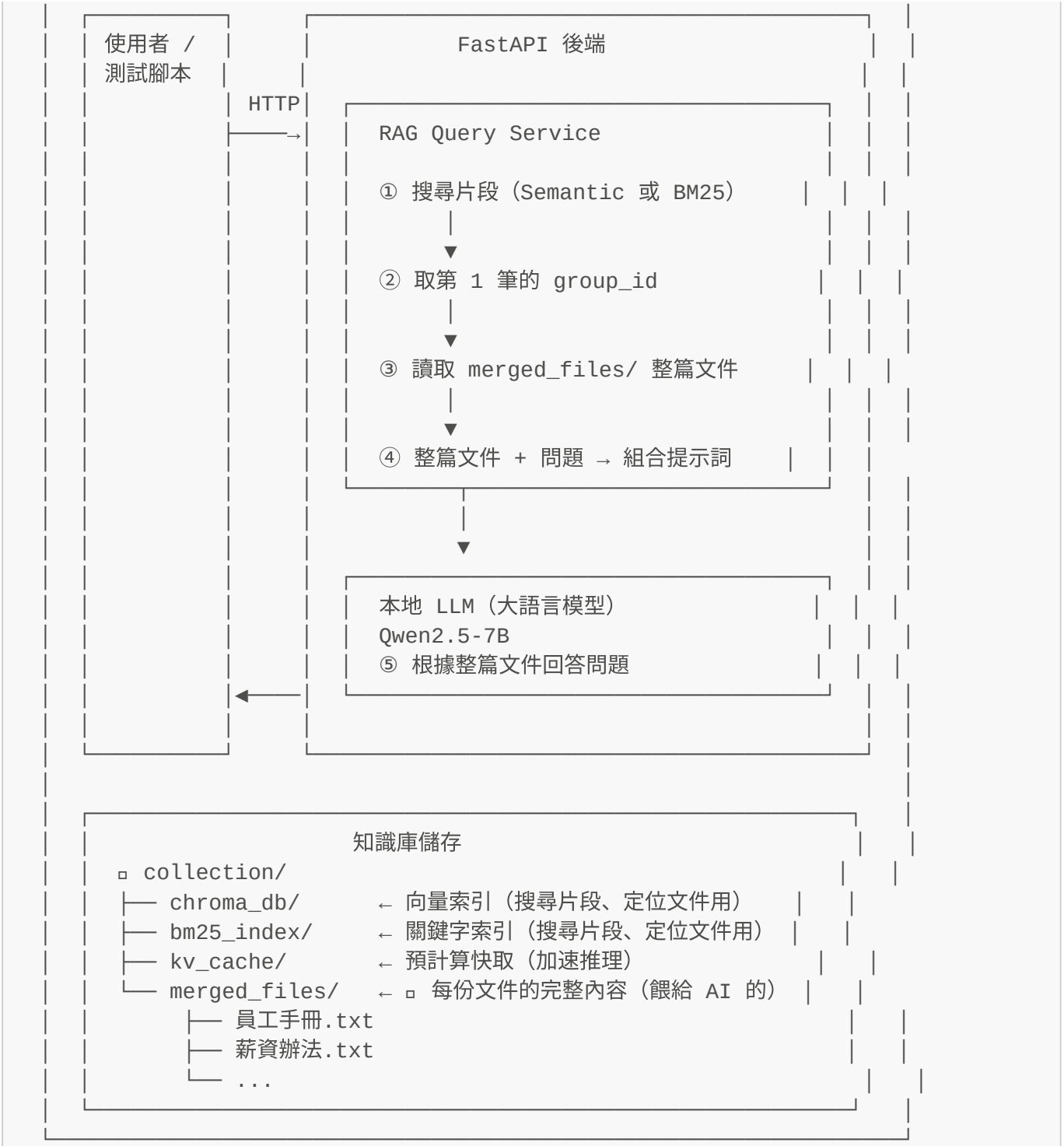
```
標準答案：「試用期為三個月」
AI 回答：   「根據公司員工手冊，新進員工的試用期為三個月，期滿後...」

                        比對過程
                        ──────
    ┌──────────────────────────────────────────┐
    │  1. 正規化                                │
    │     去掉標點、空白                         │
    │     簡體 → 繁體                           │
    │     「試用期為三個月」→「試用期為三個月」     │
    │                                          │
    │  2. 拆成 bigram（兩兩一組）                │
    │     「試用」「用期」「期為」「為三」          │
    │     「三個」「個月」                        │
    │     共 6 組                               │
    │                                          │
    │  3. 看 AI 回答中包含幾組                    │
    │     「試用」▫「用期」▫「期為」▫              │
    │     「為三」▫「三個」▫「個月」▫              │
    │     6/6 = 100% ≥ 40%                     │
    │                                          │
    │  ▫ PASS                                  │
    └──────────────────────────────────────────┘
```

為什麼用模糊比對？ 因為 AI 的回答不會跟標準答案一字不差，只要「關鍵資訊有被提到」就算通過。40% 的門檻代表：即使 AI 用了不同的措辭，只要核心內容有覆蓋到就行。

八、系統架構總覽

```
┌─────────────────────────────────────────────────────────────────┐
│                           KM 系統架構                            │
│                                                                 │
```



九、API 互動一覽

API 端點	用途	階段
`GET /health`	確認服務是否存活	前置檢查
`GET /api/v1/collections`	列出已建立的知識庫	前置檢查
`POST /api/v1/process`	匯入文件、建立知識庫	準備階段
`GET /api/v1/status/{task_id}`	查詢文件處理進度	準備階段

API 端點	用途	階段
`POST /api/v1/query/execute`	**RAG 問答（核心）**	**問答階段**

十、一句話總結

> `/api/v1/query/execute` 的 RAG 設計就是：**收到問題 → 用搜尋引擎（Semantic 或 BM25 擇一）找出最相關的片段 → 取排名第 1 的片段定位到它所屬的原始文件 → 讀取該文件的完整內容 → 把整篇文件 + 問題組合成提示詞 → 送給本地 AI 模型生成回答**。測試則透過「黃金問答 + 模糊比對」來驗證整條流水線是否正確運作。