冠東 AI 客服小助手 — 設計與實作教學手冊（新手版）

> 這份手冊帶你從零理解「一個能回答公司網站問題的 AI 客服」是怎麼設計與實作的。不需要 AI 背景，但看完你會懂 RAG、向量檢索、LLM、部署成本，以及——最重要的——**我們在真實使用中踩到哪些坑、為什麼當初沒想到、後來怎麼解決**。

目錄

1. 這個系統在做什麼
2. 核心概念：RAG 是什麼
3. 系統架構總覽
4. 資料怎麼進來（Ingest）
5. 問題怎麼被回答（Retrieve + Generate）
6. 部署與成本
7. 後台管理
8. ★ 實戰踩坑與解法（本手冊重點）
9. 給新手的學習路徑
10. 名詞速查表

---

1. 這個系統在做什麼

把冠東官網（118 個中 / 英 / 日 HTML 頁）變成一個**會回答問題的小幫手**：

> 使用者問「ER-100LED-7K 的規格？」→ 系統從官網內容找出相關段落 → 交給 AI 生成一段**有根據**的回答，並附上來源頁與產品圖。

四大賣點：**檢索精準、三語支援、可溯源、不亂編**。

---

2. 核心概念：RAG 是什麼

RAG = Retrieval-Augmented Generation（檢索增強生成）。

用一個比喻：把 AI 想成一個很會講話、但**沒讀過冠東網站**的客服。如果直接問它冠東的產品，它會亂編（幻覺）。RAG 的做法是：

> 先幫它「翻到相關那幾頁」（**檢索**），再叫它「照著這幾頁回答」（**生成**）。

所以 RAG 是**「開書考試」**——AI 不用背，只要會找、會讀、會整理。這也是它**不亂編、可標來源**的原因。

一個 RAG 系統拆成兩大塊：

區塊	白話	吃什麼資源
檢索（Retrieval）	找出相關內容	CPU（很輕）
生成（Generation）	用 AI 講成人話	GPU 或雲端 API（成本重點）

---

3. 系統架構總覽



- **檢索**在服務裡跑，靠 CPU；
- **生成**呼叫外部 LLM，透過 **LiteLLM 閘道**（統一金鑰、用量統計、可換供應商不動程式）；
- 兩者**解耦**：LLM 掛了，系統仍能退化成「只列出找到的片段」。

技術棧：Python + FastAPI、sentence-transformers（embedding）、rank-bm25（關鍵字檢索）、 numpy（向量計算，**不需向量資料庫**，因為資料量小）、Docker、LiteLLM。

---

4. 資料怎麼進來（Ingest）

「建索引」就是把網站變成 AI 查得到的知識庫，分四步（對應 `src/ingest.py`、`src/extract.py`）：

步驟 1：抽取（Extract）

把 HTML 的雜訊（選單、頁尾、版權、裝飾）拿掉，只留正文。

> 難點：舊網站每頁都有重複選單/頁尾 → 用「跨頁高頻的行＝樣板」自動偵測移除。

步驟 2：切塊（Chunking）

把長文切成約 500 字的小段（chunk），相鄰段落重疊一點，避免句子被切斷。

> 為什麼要切？因為檢索是「找最相關的**一段**」，段落太長會混入無關內容、也超過模型輸入。

步驟 3：向量化（Embedding）

把每個 chunk 用多語模型（`multilingual-e5-small`）轉成一串數字（向量）。

> **向量**可以想成「這段文字的語意座標」。意思相近的文字，座標也相近。

步驟 4：存檔

把向量（`embeddings.npy`）和文字（`chunks.jsonl`）存起來。之後問問題就跟這些向量比對。

> 結果：118 頁 → 107 頁有內容 → 約 150 個乾淨片段。索引只有幾 MB，用 numpy 內積就能算， **不需要 Milvus/FAISS 之類的向量資料庫**。

---

5. 問題怎麼被回答（Retrieve + Generate）

檢索（`src/retrieve.py`）— RAG 的核心

用兩種方式找，再融合：

方法	擅長	白話
向量檢索（dense）	語意相似	「問燈，找到照明」這種同義
關鍵字檢索（BM25）	精確詞、型號料號	「ER-100LED-7K」逐字命中
RRF 融合	兩者取長補短	把兩份排名合併成一份

> 為什麼要 hybrid？因為**向量對型號很弱**（它擅長語意，不擅長精確字串），而**BM25 對型號很強**。 兩個一起用，語意題和料號題都能顧到。

生成（`src/rag.py` + `config.py`）

把「找到的片段 + 使用者問題 + 系統提示詞」組起來丟給 LLM。系統提示詞（SYSTEM_PROMPT）是關鍵， 它規定 AI：**只根據提供內容作答、找不到就說查無、用同一種語言、不要亂輸出。**

6. 部署與成本

系統設計成「一份程式、環境變數切換」，兩種跑法：

- **開發（Mac / CPU）**：Docker 起服務，LLM 可接本機或雲端。
- **上線（一台便宜 VPS）**：官網 + RAG + LiteLLM 一份 `docker compose` 跑起來，LLM 走**雲端 API**（免 GPU）。低流量月費約 **NT$300~700**。

> 關鍵成本觀念：**檢索很便宜（CPU），貴的是生成（LLM）**。流量不大時，用「雲端 API 按量付費」最划算，不要為了低流量去租 24 小時 GPU。詳見 `deploy/README.md`、`docs/cost_for_client.html`。

7. 後台管理

`/admin` 是一個管理後台（薄代理到 LiteLLM），可以： - 看有哪些模型、建立/撤銷 Virtual Key、看用量花費； - **切換小幫手使用的 Key 與模型**（免改設定檔、免重啟，即時生效）。

> 這個「模型可切換」的設計，後來成了救命功能——見下一節的踩坑 3。

8. ★ 實戰踩坑與解法

> 這節是本手冊的精華。每個坑都照「**現象 → 為什麼當初沒想到 → 根本原因 → 解法 → 學到的原則**」講。 這些不是 bug 而已，而是**做 AI 產品最真實的一課：資料的形狀，決定系統的行為**。

坑 1：問產品的「圖面」，小幫手看不到任何圖

- **現象**：問「ER-100LED-7K 的圖面」，回「查無」，或答非所問。
- **為什麼當初沒想到**：初版把「回答問題」等同於「處理文字」，很自然地只抽取 HTML 的**文字**。
- **根本原因**：抽取用 `get_text()`，`<img>` 圖片**整個被丟掉**——連檔名、alt 說明都沒進索引。 對向量庫來說，那些圖**根本不存在**。
- **解法**：ingest 時把每頁的圖片（`src` + `alt`）存進 chunk 的 metadata；回答時把對應圖片 URL 一起回傳，前端顯示縮圖。（`extract.py` / `rag.py` / `web/index.html`）
- **學到**：**RAG 只知道你「餵給它」的東西**。圖片、表格、PDF 都要「特別處理」才進得了知識庫。

坑 2：圖片顯示錯的、還混進 logo

- **現象**：問「金屬鹵素燈箱」卻跳出隔壁產品的圖、甚至虛擬主機 logo。
- **為什麼當初沒想到**：初版「把檢索到的每一頁的圖都附上」，想說多給總比少給好。
- **根本原因**：（a）一次檢索會撈**多頁**，把每頁圖全倒出來就混入無關圖；（b）裝飾圖（logo、橫幅）沒被過濾；（c）用「排名第一的頁」配圖，但排名不穩，可能配錯頁。
- **解法**：

  1. **只顯示「確定對得上」那一頁**的圖——用**型號**或**產品名稱（比對頁面標題）**鎖定，區分「箱 vs 泡」；
  2. **跨頁高頻的圖＝裝飾**，自動過濾（logo/橫幅出現在很多頁）；
  3. 檔名含 `logo` / `banner` 或 alt 是泛稱 `product` 的一律排除。

- **學到**：**顯示錯的比不顯示更傷信任**。寧可保守，也要讓內容和佐證對得上。

坑 3：AI 明明有資料卻說「查無」，還簡繁夾雜

- **現象**：檢索明明把含規格的頁放第一，AI 卻回「查無相關資訊」，或中文夾雜簡體、亂抄格式。
- **為什麼當初沒想到**：初版用了一個**小型開源模型（7B）**，以為「有給內容就會照著答」。
- **根本原因**：**模型能力不足**。7B 模型指令跟隨弱，會過度保守（亂說查無）、語言不穩、甚至把系統標記抄進答案。這**不是檢索或提示詞的錯**，是模型本身的天花板。
- **解法**：

  1. 提示詞收緊（擇一回答、強制繁體、禁止輸出圖片/網址/來源標記）——能改善但治標；
  2. **根本解：換更強的模型**。因為系統早就設計成「模型可在後台切換」，換成 `gemini-2.5-flash` / `claude-sonnet` 後，答案品質立刻穩定。

- **學到**：**RAG 的品質 = 檢索品質 × 模型品質**。檢索對了，模型不夠力一樣白搭；而「讓模型可插拔」是很值得的架構投資。

坑 4：問英文型號（AAP 4 Chip、ER-100LED）查不到——最嚴重的一個

- **現象**：官網明明有「AAP 4 Chip」，小幫手卻查不到。
- **為什麼當初沒想到**：初版加了「**同語言過濾**」——中文問句只回中文頁，避免三語內容互相干擾，立意良善，但沒料到「語言中性的型號」會出事。
- **根本原因**：`AAP 4 Chip` 全是英文字母 → 語言偵測判成「英文」 → 系統**只搜英文頁** → 但這型號**只出現在中文頁** → 被自己的語言過濾擋掉。（檢索其實找得到，是被過濾規則排除的）
- **解法**：把「同語言過濾」從**硬性排除**改成**軟性優先**—— 同語言優先，但**只要跨語言有明顯更相關的結果就改用它**。這樣語言中性的型號代碼不會被擋。

- **學到**：**任何「過濾規則」都可能誤殺**。過濾要保留逃生門（fallback），尤其是型號、代碼、專有名詞這種「跨語言」的東西。

坑 5：問「聯絡信箱」查不到

- **現象**：電話、地址查得到，唯獨 **Email** 查不到。
- **為什麼當初沒想到**：初版為了乾淨，把每頁的**頁尾（footer）整段移除**——而 Email 剛好只放在頁尾 / `mailto:` 連結裡。
- **根本原因**：清理雜訊時「一刀切」把有用的聯絡資訊也切掉了。
- **解法**：在**移除頁尾之前**，先從 `mailto:` 連結把信箱撈出來，再補回聯絡 / 關於頁。
- **學到**：**清理雜訊時，先問「這裡面有沒有寶」**。footer、mailto、alt、圖片裡常藏著關鍵資訊。

這五個坑的共同啟示

> **AI 系統的行為，取決於資料的形狀。** 舊網站把資訊藏在圖片、頁尾、mailto、單一語言頁裡；初版設計為了「處理常見情況」做了合理簡化（只抽文字、同語言過濾、砍頁尾），但真實資料的邊緣情況，只有**在真實使用中才會浮現**。這正是為什麼 AI 產品要**快速上線**、**用真實問題迭代**，而不是一開始就想把所有情況設計完美。

---

9. 給新手的學習路徑

1. **先懂概念**：RAG、embedding、向量相似度、chunking、BM25、LLM、幻覺。（本手冊第 2、4、5 節）
2. **動手跑一次**：用 Docker 起服務、建索引、問幾題、看 `/health`。
3. **讀三個檔**：`config.py`（所有旋鈕）、`src/ingest.py`（資料進來）、`src/rag.py`（問題出去）。
4. **做一次診斷**：用 `make diagnose Q="..."` 看某問題撈到哪些片段、分數多少——這是理解檢索最快的方法。
5. **改一個小東西**：例如把 `TOP_K` 從 5 改成 8，重問，觀察差異。
6. **重現一個坑**：試試問英文型號、問圖片、問聯絡方式，體會「資料形狀」如何影響結果。

---

10. 名詞速查表

名詞	白話解釋
RAG	先檢索、再生成的「開書考試」式 AI 問答
Embedding / 向量	把文字變成一串代表語意的數字座標
Chunk / 切塊	把長文切成小段，方便檢索
BM25	傳統關鍵字檢索，擅長型號料號等精確詞
RRF	把多份排名清單融合成一份的方法
Reranker	用更精細的模型對候選重新排序（選用，較慢較準）
LLM	大型語言模型，負責把片段講成人話
幻覺 (Hallucination)	AI 一本正經地亂編。RAG 就是為了壓低它
LiteLLM	LLM 閘道：統一金鑰、記用量、可換供應商不動程式
Virtual Key	LiteLLM 發的推論金鑰，可限定模型、記錄用量
同語言過濾	讓中文問句優先回中文頁的規則（要留 fallback！）

本手冊對應專案：eastring-rag-agent。相關文件： `README.md` 、 `docs/HARDWARE.md` 、 `deploy/README.md` 、 `docs/cost_for_client.html` 。