【Day 2】RAG 簡介

1. 簡介

在 2025 年，**RAG**（Retrieval-Augmented Generation, 檢索增強生成） 是常見的 LLM 應用架構之一。它的核心概念是：**透過檢索外部知識，來彌補 LLM 預訓練的不足，並提升生成的準確度與可靠性**。

一個典型的 RAG 系統通常包含以下幾個模組：

- **原始資料庫（Raw Knowledge Base）**
  - 以 *chunk* 為單位存放知識。
  - 一個 chunk 可能是固定長度（例如 k 個 tokens／字符），或依據語意段落來切割。
- **編碼器／Embedding Function**
  - 將 query 與 chunks 轉換成向量表示。
  - 可以使用 OpenAI、Google 提供的 Embeddings。也有開源的 Embeddings 例如 bge-m3。
- **向量資料庫（Vector Database）**
  - 儲存並管理 embeddings，支援高效檢索。
  - 常見工具有 FAISS、Milvus、Pinecone、Weaviate。
- **相似度函數（Similarity Function）**
  - 判斷 query 與 chunks 的相似度。
  - 常見度量方式：內積（dot product）、cosine similarity、歐式距離（Euclidean distance）。
- **LLM（Large Language Model）**
  - 負責最終的生成，將檢索內容與 query 結合。

Retrieve 階段

1. 使用者輸入 query。
2. query 經過 embedding function 轉換為向量表示。
3. 在向量資料庫中檢索，取出與 query 最相關的 k 個 chunks。

Generate 階段

1. 將 query 與檢索結果一起丟進 LLM。
2. LLM 生成最終回答。

2. 優化方向

理解 RAG 的基本結構後，我們可以思考如何優化，來提升答案的**準確度**、**完整性**與**穩健性**。這也是後續論文討論時的主要方向。常見的優化包括：

- **檢索策略**
  - hybrid search：結合語意 embedding 與關鍵字檢索（例如 BM25）。
  - 多階段檢索：例如先檢索 100 個目標，再使用計算成本更高的 **“reranker”** 來進行排序找到最接近 query 的 20 個 chunks。
  - 使用對比學習（contrastive learning）來訓練 retriever，讓模型可以泛化在不同應用場景。
- **Chunking 策略**
  - 決定如何切分文本：固定長度 vs 語義切分。
  - 引入 sliding window 來保持上下文。
- **從檢索到生成的策略（Fusion Strategies）**
  - Concatenation：直接拼接 chunks 與 query。
  - Fusion-in-Decoder（FiD）：LLM 在生成時動態選擇相關 chunks。
- **多模態資料（Multimodal RAG）**
  - 不僅檢索文本，也能檢索圖片、表格、程式碼或聲音等。