【Day 3】 LangChain (1) 使用 API 驅動 LLM

1. 串接 API: `langchain_google_genai`

今天是挑戰的第三天，讓我們先來看看如何用 API 來使用 LLM。這個系列使用的模型是 gemini-2.5 系列，主要原因是因為它有免費的 api key。

因為後續 RAG 系統搭建，我們會使用 langchain 跟 google-genai 的整合套件：`langchain_google_genai`。實例化模型之後，可以使用以下方法來呼叫它：

- `.invoke()` ：處理單一輸入，返回一個輸出。
- `.batch()` ：批次處理，返回一個列表。
- `.stream()`：串流，返回一個生成器（generator）。

在 LangChain 中，這些操作都基於其核心概念 `runnable`。每個 `runnable` 都實作了這些呼叫方法。當我們將整個系統串接起來後，也能像呼叫單一模型一樣，使用 `.invoke()` 來驅動整個流程。

```
from langchain_google_genai import GoogleGenerativeAI

llm = GoogleGenerativeAI(model="gemini-2.5-flash-lite")

llm.invoke("你喜歡 spaghetti 還是 penne?，直接輸出答案，不要解釋。") # 'penne'
```

以下介紹常用的參數：

參數名稱	類型	描述
`model`	**str**	gemini-2.5-{pro, flash, flash-lite}
`google_api_key`	**str**	如果沒有設定會自動讀取環境變量 GOOGLE_API_KEY
`max_output_tokens`	**int** > 0; default=64	最多輸出多少 token
`thinking_budget`	**int**	思考的 token 數量

include_thoughts	**bool**	輸出思考過程

此外也有 temperature、top_p、top_k 等參數，會影響模型輸出的**隨機性與創意程度**。

temperature：控制輸出的平滑度。

- **低溫**（例如 0.3）：輸出更具確定性與可重現性，適合用於**推理、代碼生成與摘要**等任務。
- **高溫**（例如 0.7 以上）：輸出更具變化與創意，適合**撰寫文案或腦力激盪**。

top_p, top_k 分別將 sample 範圍控制在機率分布累積機率 p 以內的 token 或機率前 k 高的 token。

- 更多 GoogleGenerativeAI 模型參見[文檔](https://ai.google.dev/gemini-api/docs/models?hl=zh-tw)。
- 關於 temperature, top_k, top_p 的細節，網路上已有許多介紹，例如[這篇](https://medium.com/seaniap/ai輸出的調節術-了解openai的temperature與top-p參數-d849e29dc505)。

從上面的範例與參數，能看出用 API 跟用網頁來使用 LLM 很大的不同，我們需要自己設定參數，呼叫的方法，除此之外，為了讓我們的 LLM-RAG 系統有實用性，還需要實作 **System Prompt（系統提示詞）**和 **Chat History （對話紀錄）**（明天做）。

2. 提示詞工程：賦予模型角色

直接在網頁上使用 LLM 不同，透過 API 我們需要更精準地控制模型行為。這正是 **Prompt**（提示詞）與 **System Prompt**（系統提示詞）的用武之地。

2.1 提示詞模板（Prompt Template）

在開發 AI 應用時，我們經常需要使用**模板化**的提示詞。LangChain 提供了 PromptTemplate，讓你能輕鬆地定義變數，並重複使用提示詞結構。

```
from langchain_core.prompts import PromptTemplate

prompt_template = PromptTemplate.from_template(
    "介紹 {type}。使用以下格式：\n\n**起源**\n**推薦搭配的醬料**"
)
```

```
chain = prompt_template | llm
chain.invoke({"type": "lasagna"})
```

> 好的，這是一份關於 Lasagna 的介紹：
>
> **起源**
>
> Lasagna，或稱千層麵，是一道源自義大利的經典麵食料理，其歷史可以追溯到古羅馬時期。最早的 Lasagna 形式與現代的截然不同，當時的羅馬人將一種稱為「laganum」的扁平麵皮，層層疊疊地與肉類和蔬菜一起烘烤。……

2.2 System Prompt

模型在經過**指令微調**（Instruction Fine-Tuning）與 **RLHF**（RL with Human Feedback）後，已經學會如何依照指示回應。而 **System Prompt（系統提示詞）**則能更進一步地定義模型的角色、回覆風格與行為規範。 使用網頁介面時，模型通常已有預設的 System Prompt。而透過 API，我們則可以**自訂**這些設定。

```
from langchain_core.prompts import ChatPromptTemplate
chat_template = ChatPromptTemplate.from_messages(
    [
        ("system", "你是一個義大利麵 Youtuber。\n你會活潑、幽默地介紹義大利麵。"),
        ("human", "介紹 {type}。使用以下格式：\n\n**起源**\n**推薦搭配的醬料**"),
    ]
)

chat_chain = chat_template | llm
chat_chain.invoke({"type": "lasagna"})
```

> 嘿！各位
> 麵粉們！我是你們最愛的義大利麵 Youtuber！今天我們要來
> 聊聊一位重量級的麵食巨星——**千層麵 (Lasagna)**！
>
> 準備好你的胃了嗎？因為我們要開始一場美味的旅程！……