自然語言處理

第 11 章 檢索增強式文本生成

Retrieval Augmented Generation (RAG)

講師：紀俊男

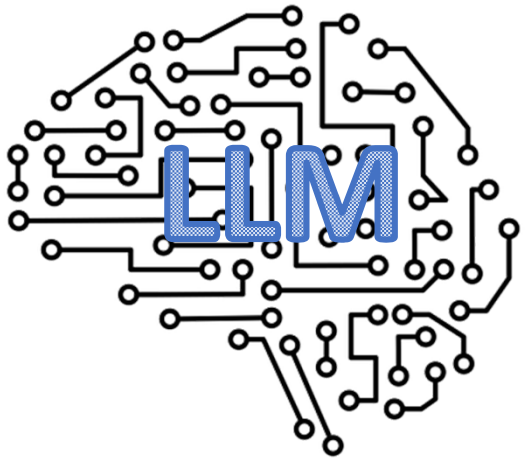
本章大綱

- RAG 簡介
- RAG 實例講解
- 本章總結

RAG 簡介

大語言模型的困境

2023 文本
(Corpus)



太新的問題

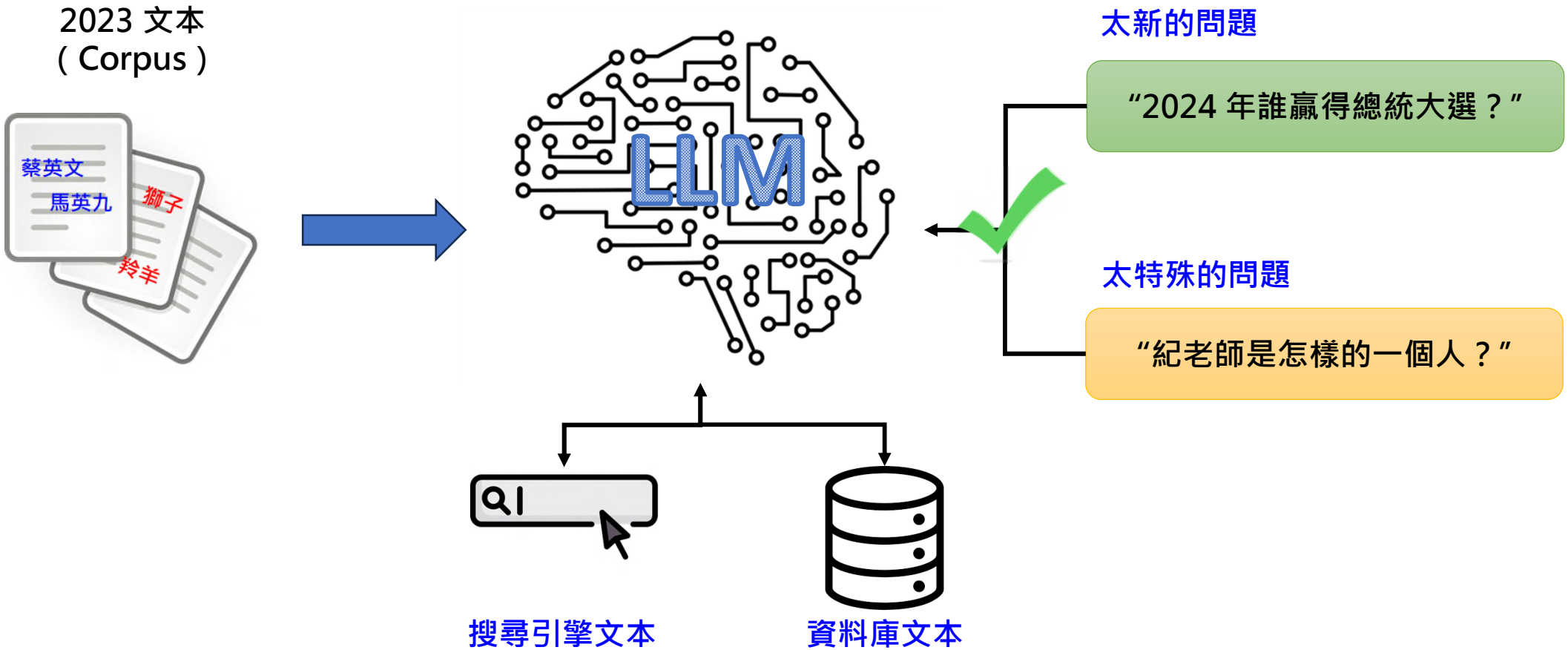
“2024 年誰贏得總統大選？”

太特殊的問題

“紀老師是怎樣的一個人？”

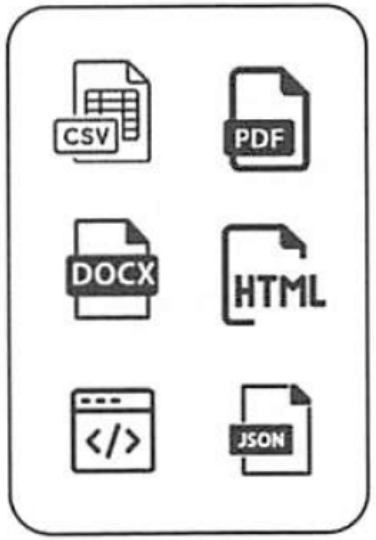
解決方法

- RAG = Retrieval Augmented Generation（檢索增強式文本生成）



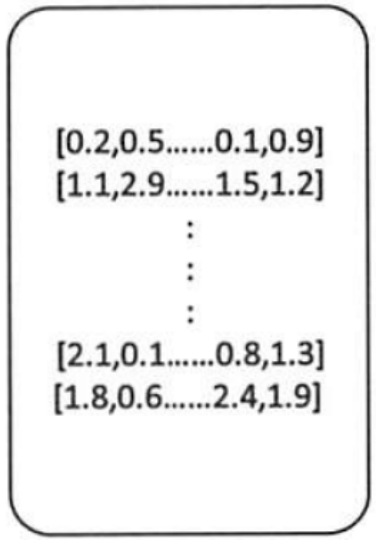
RAG 資料來源之生成步驟

資料載入
Data Loading

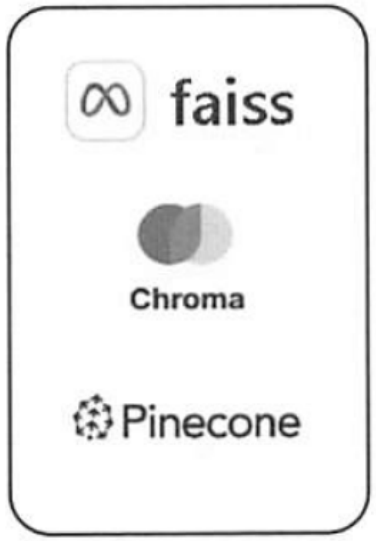


資料分割
Data Chunking

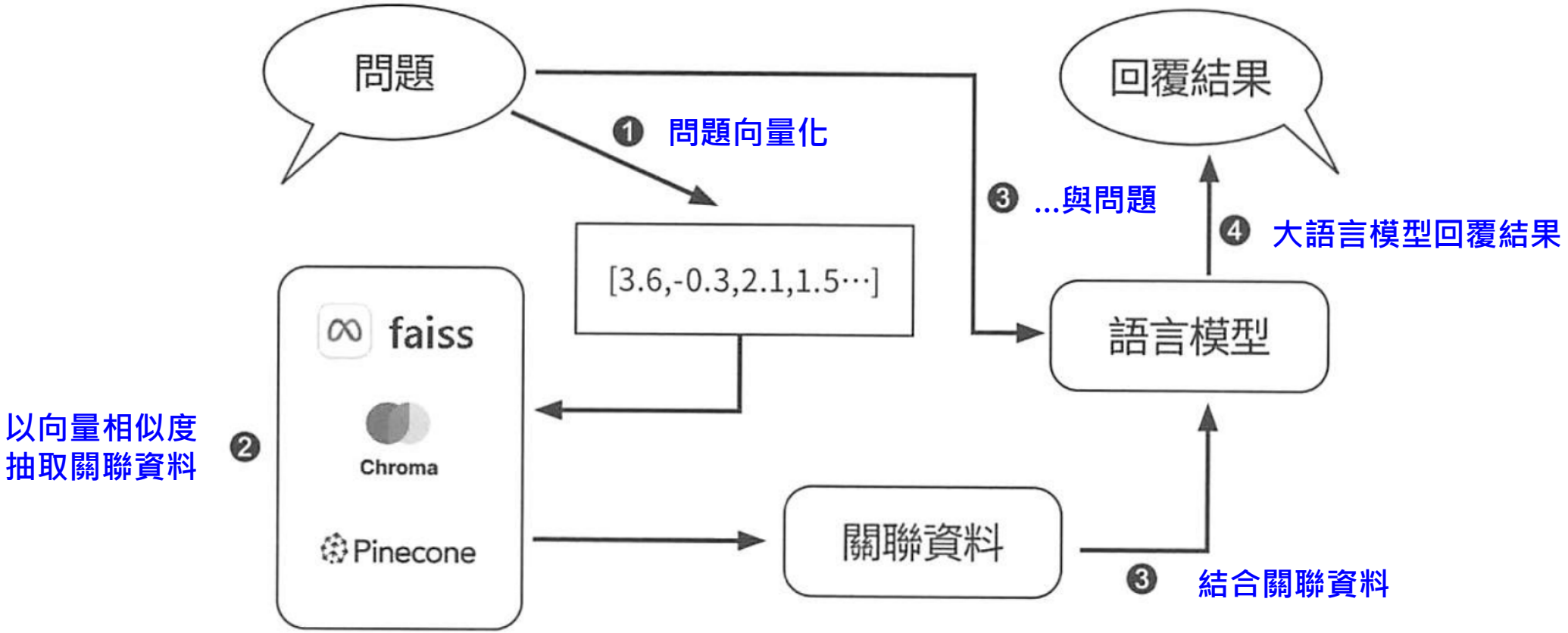
資料嵌入化
Data Embedding



向量資料庫
Vector Databases



RAG 運作原理

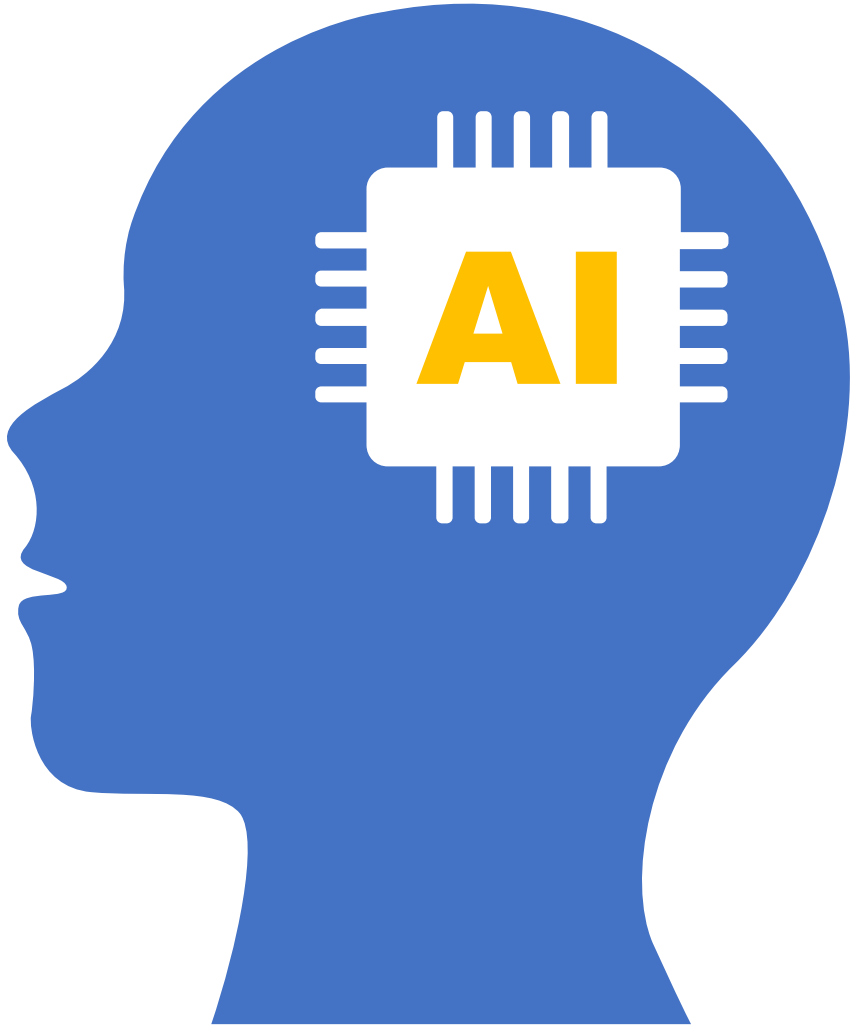

問題
回覆結果
❶ 問題向量化
❸ ...與問題
❹ 大語言模型回覆結果
[3.6,-0.3,2.1,1.5⋯]
語言模型
以向量相似度
抽取關聯資料
❷
faiss
Chroma
Pinecone
關聯資料
❸ 結合關聯資料

AI RAG 實例講解

範例完整原始碼：
https://lurl.cc/2w9nLm

RAG 實例講解

- **環境安裝**
- 直接連上大語言模型（不使用 RAG）
- 整合大語言模型與 RAG
- 以圖形介面互動



安裝 GPU 驅動相關套件

```
# 更新 Linux 內的套件清單至最新版
!apt-get update

# 安裝 PCI 匯流排工具（PCI Utility）與 lshw (LiSt HardWare)，以便能偵測到 GPU
# -y：遇到詢問是否安裝，一律自動回答 yes
!apt-get install -y pciutils lshw

# 用 nVidia 的 System Management Interface (SMI) 確認 GPU 的確抓得到
!nvidia-smi
```

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 12.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |                    0 |
| N/A   65C    P8              11W /  70W |      0MiB / 15360MiB |      0%      Default |
|                                         |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

安裝 Ollama

```
# 至 https://ollama.com/download/linux
# 直接將安裝 Ollama 於 Linux 的指令貼上
!curl -fsSL https://ollama.com/install.sh | sh

# 啟動 Ollama，讓它執行於背景中
# ollama serve: 用 Server 模式、而非互動模式執行 Ollama
# > server.log：將本應顯示於螢幕的訊息，轉向輸出至 server.log 這個檔備查
# 2>&1：2 為 stderr。將所有錯誤訊息，轉向 &1 (stdout，螢幕) 輸出。
# &：將程式啟動之後，馬上返回，不要等該程式執行完成
!ollama serve > server.log 2>&1 &

# 將 Llama-3 模型下載，並做為此次的大語言模型
# ollama run <模型名稱>：下載並執行特定 LLM
# > model.log：將本應顯示於螢幕的訊息，轉向輸出至 model.log 這個檔備查
!ollama run gemma:7b > model.log 2>&1 &

# 將 nomic-embed-text 模型，拉下來備用
# nomic-embed-text 是一款能將文字向量化的嵌入層模型
!ollama pull nomic-embed-text > embedding.log 2>&1 &

# 注意：上述指令皆以「&」後綴，告知 Colab「不用等 Linux 執行完」。
# 但事實上，不論啟動為 Server，或下載 LLM Models，皆須 1~5 分鐘不等的時間。
# 可以查看 *.log 檔案內容，得知當前執行狀況。
```

← 這次多裝這個 Ollama 模型

安裝其它相關套件

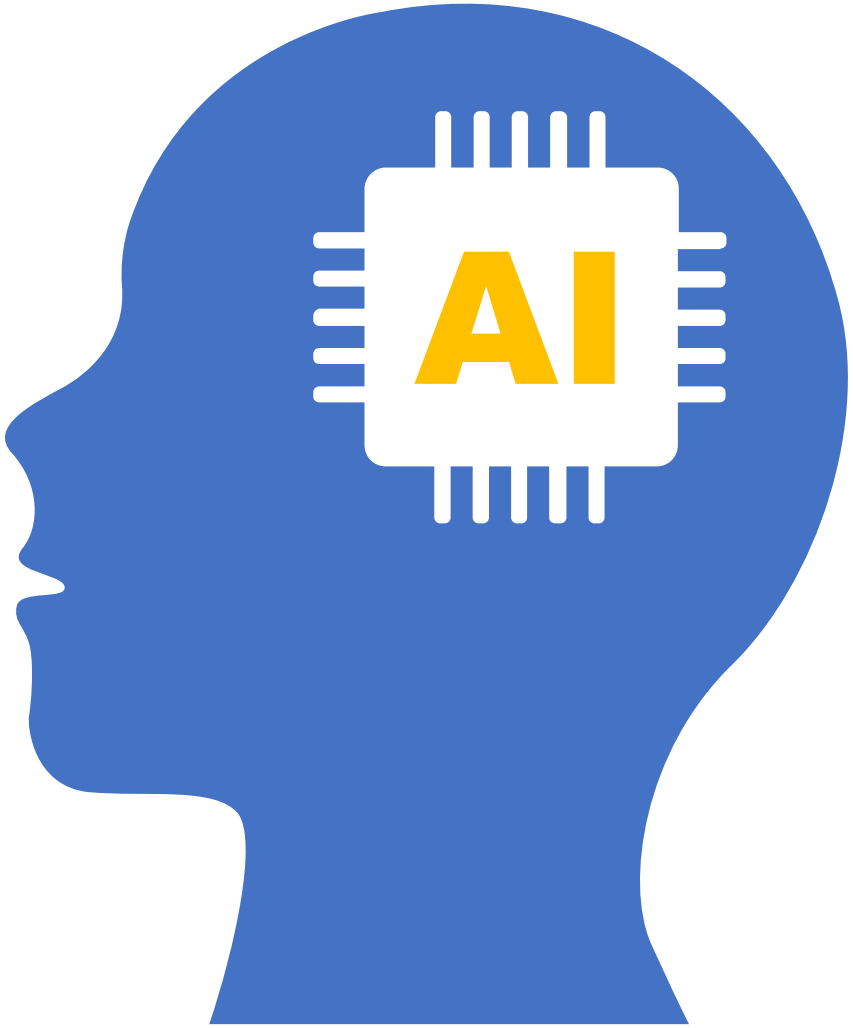
```
1  # 下載 LangChain, 一套專門連上各種大語言模型的 Python 套件
2  !pip install langchain  # LangChain 核心元件
3  !pip install langchain-community  # 各種開源大語言模型連接函數套件
4
5  # ChromaDB 是一款蠻受歡迎的開源式向量資料庫
6  !pip install chromadb
7
8  # 將做為範例的 PDF 檔下載下來
9  import os
10 Dataset_File = "GoingLearn_BookReviews.pdf"
11
12 if not os.path.isfile(Dataset_File):
13   os.system("wget https://raw.githubusercontent.com/cnchi/datasets/master/" + Dataset_File)
14
15 # 安裝 PDF 檔內容解析套件
16 !pip install pypdf
17
18 # 由 Hugging Face 開發, 開源式網頁圖形介面套件
19 # 詳見官方首頁 https://www.gradio.app/
20 !pip install gradio
```

隨堂練習：設定相關環境

- **請依照前述投影片內容，分別撰寫並執行下列程式碼：**
  - 安裝 GPU 驅動相關套件
  - 安裝 Ollama（Ollama, gemma:7b, nomic-embed-text）
  - 安裝大語言模型相關套件（LangChain）
  - 安裝向量資料庫相關套件（ChromaDB）
  - 安裝外部文件檔與套件（PyPDF）
  - 安裝圖形式介面相關套件（Gradio）

RAG 實例講解

- 環境安裝
- **直接連上大語言模型（不使用 RAG）**
- 整合大語言模型與 RAG
- 以圖形介面互動



程式碼與執行結果

```python
# 載入 Ollama 以便連上後端 LLM
from langchain_community.llms import Ollama

# LLM 名稱需與 ollama run 後方名稱相同
llm = Ollama(model="gemma:7b")

# 對 LLM 送出提詞，並且印出回應
msg = llm.invoke("紀俊男老師是怎麼樣的一個人？請用繁體中文回答我。")
print(msg)
```

紀俊男老師以其熱情、有情和有包容性而著稱。他是一位充滿熱情和有禮貌的教師，並努力建立一個包容和支持性的班級環境。以下是一些關於紀俊男老師的人格特徵：

* 熱情滿滿：紀俊男老師對教學的熱情和動力是令人驚嘆的。他積極參與課程的準備和實踐，並努力引導學生思考和成長。
* 有情有為：紀俊男老師會主動關心學生的生活狀況，並提供他們有需要的支持。他也積極參與校園活動，並為學生爭取更多權益。
* 有包容性：紀俊男老師會接受並尊重學生的不同觀點和生活方式。他也努力建立一個包容和支持性的班級環境，讓所有學生都能安心學習。
* 有禮貌有成：紀俊男老師有良好的禮貌和溝通技巧。他也擁有充足的教導經驗和專業素養。

總之，紀俊男老師是一位熱情、有情和有包容性的教師，值得學生和家長信賴。

一本正經
胡說八道

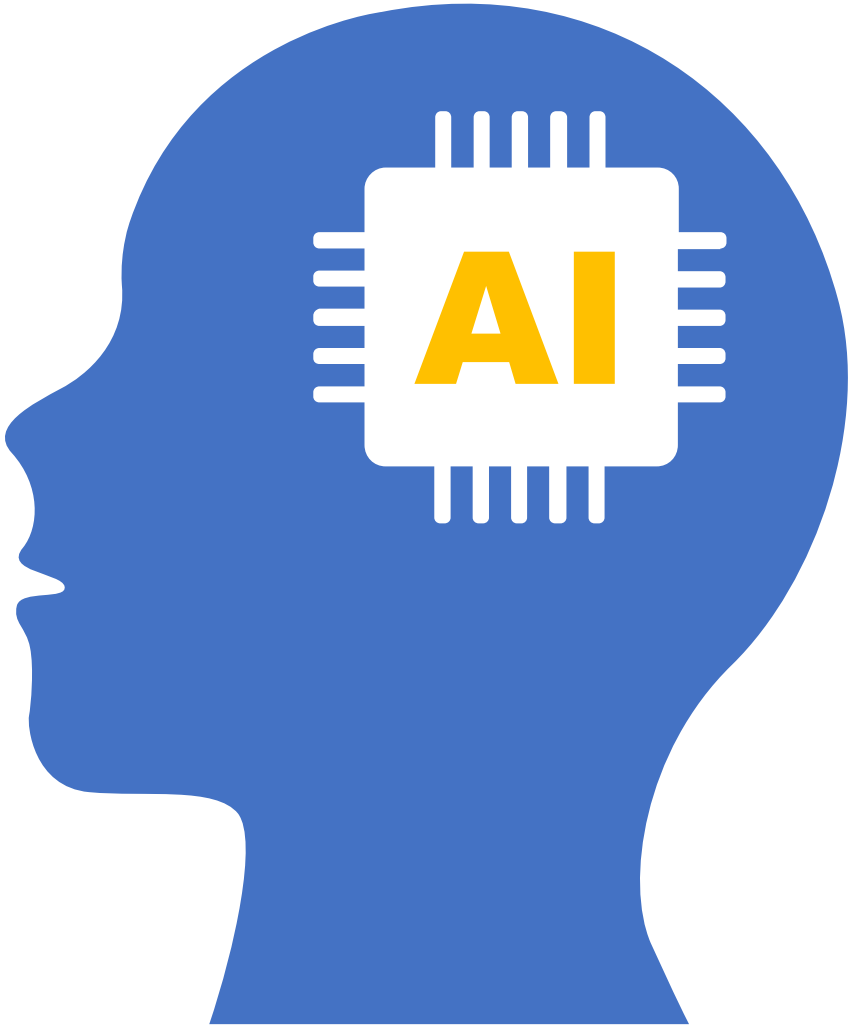
隨堂練習：不使用 RAG

- 請依照前述投影片內容，撰寫並執行下列程式碼：

```python
# 載入 Ollama 以便連上後端 LLM
from langchain_community.llms import Ollama

# LLM 名稱需與 ollama run 後方名稱相同
llm = Ollama(model="gemma:7b")

# 對 LLM 送出提詞，並且印出回應
msg = llm.invoke("紀俊男老師是怎麼樣的一個人？請用繁體中文回答我。")
print(msg)
```

RAG 實例講解

- 環境安裝
- 直接連上大語言模型（不使用 RAG）
- **整合大語言模型與 RAG**
- 以圖形介面互動



載入外部資料

- (1) 載入網頁資料

引入必要套件

```
1  # 載入網頁資料
2  from langchain_community.document_loaders import WebBaseLoader
3
```

抓取特定 URL 內容

```
4  url = "https://goinglearn.com.tw/about-this-site"
5  loader = WebBaseLoader(url)
6  web_docs = loader.load()
7
8  print(web_docs)
```

```
[Document(page_content='\n關於本站 | Going Learn：紀老師程式教學網\n\n跳至主要內容
\n\n\n\n \n\n\n\n\nGoing Learn：紀老師程式教學網\n程式初心者最佳學習網站\n\n\n\n\n\n\n\n\n\n
選單\t\n關於本站\n初學者\n程式語言 \n\nPython\n\n\n網頁設計\n手機程式\n嵌入式\n軟體工程\n
電腦書評\nRSS 訂閱\n\n\n\n\n\n\n\n\n\n\n\n\n\n關於本站 \n\n關於站長\n嗨！我是 Going Learn 的
站長紀俊男。由於長年在電腦補習班任教，也常被叫「紀老師」，或直接以英文名字「Robert」、
「Robert 老師」稱呼。我從 1982 年看到隔壁鄰居的「小教授二號」開始，就對電腦產生無比的好奇。
...
```

載入外部資料

- (2) 載入 PDF 資料

我是如何選書、切書、看書、評書的？

常有同學問我：「老師，我想學 XXX，可以幫我推薦幾本書嗎？」因為擔任補習班老師的關係，會有蠻多機會推薦書籍給同學。一般而言，我就把口袋名單提供給同學就完事了。但有些補習班的「常客」，跟我跟久了，知道我是「重度書蟲」。不僅擁有兩萬多本電腦書，出門還一定得帶個一兩本在包包裡才安心的那種人。所以有些人就好奇：

「老師，你是怎麼選書、看書的？」
「你怎麼有辦法兩個禮拜就看完一本電腦書？」
「我看你都用 iPad 在看書，那些書是怎麼電子化的？」
「你是怎麼把好幾本書的內容，那麼快就融會貫通的？」

```
1  # 載入 PDF 資料
2  from langchain_community.document_loaders import PyPDFLoader
3
4  loader = PyPDFLoader(Dataset_File)
5  pdf_docs = loader.load()
6
7  print(pdf_docs)
```

[Document(page_content='我是如何選書、切書、看書、評書的？\n常有同學問我：「老師，我想學XXX，可以幫我推薦幾本書嗎？」因為擔任補\n習班老師的關係，會有蠻多機會推薦書籍給同學。一般而言，我就把口袋名單\n提供給同學就完事了。但有些補習班的「常客」，跟我跟久了，知道我是「重\n度書蟲」。不僅擁有兩萬多本電腦書，出門還一定得帶個一兩本在包包裡才安\n心的那種人。所以有些人就好奇：\n「老師，你是怎麼選書、看書的？」\n「你怎麼有辦法兩個禮拜就看完一本電腦書？」\n....

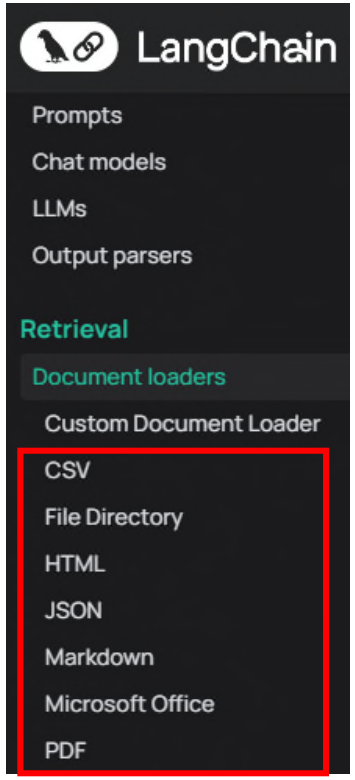
載入外部資料

- (3) 將所有的資料接合在一起

```
1  # 將所有資料接合在一起
2  all_docs = web_docs + pdf_docs
3
4  print(all_docs)
```

到底可以載入多少種外部資料？

- LangChain Documents
  - https://python.langchain.com/v0.1/docs/modules/data_connection/document_loaders/



可以載入這麼多種外部資料

隨堂練習：載入外部資料

- 請依照前述投影片內容，撰寫並執行下列程式碼：

```python
# 載入網頁資料
from langchain_community.document_loaders import WebBaseLoader

url = "https://goinglearn.com.tw/about-this-site"
loader = WebBaseLoader(url)
web_docs = loader.load()

print(web_docs)

# 載入 PDF 資料
from langchain_community.document_loaders import PyPDFLoader

loader = PyPDFLoader(Dataset_File)
pdf_docs = loader.load()

print(pdf_docs)

# 將所有資料接合在一起
all_docs = web_docs + pdf_docs

print(all_docs)
```

資料分割

```python
# 將載入的文件以每段 1000 Tokens、重複 200 Tokens 的方式切分
# 可保證原文不會因切分而失去上下文
# 也可以讓 LLM 透過重複部分，得知誰是上文、誰是下文
from langchain.text_splitter import RecursiveCharacterTextSplitter

text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
splits = text_splitter.split_documents(all_docs)

for split in splits:
    print(split)
```

```
page_content='關於本站 | Going Learn：紀老師程式教學網\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n
page_content='關於站長\n嗨！我是 Going Learn 的站長紀俊男。由於長年在電腦補習班任教，也常被叫「紀老師」，或直接以英文名字「Robert」
page_content='最後，如果您覺得這個網站不錯，還請多多推薦給您的朋友！您可以常常把本站文章連結分享至 Facebook、Google+、Twitter、或 L
page_content='Copyright (c) 2008-2021 和群資訊有限公司\n台北市中正區 10041 忠孝西路一段 33 號 7 樓\n(02) 23920369\n\n \n\n\n\n\t\t本
page_content='我是如何選書、切書、看書、評書的？\n常有同學問我：「老師，我想學XXX，可以幫我推薦幾本書嗎？」因為擔任補\n習班老師的關
page_content='接著，我會翻看「目錄」，憑藉直覺判斷，這樣的章節安排，是否足夠涵蓋作\n者想傳達的「東西」？各章節的安排順序，是否符合邏
page_content='1.攤開書籍，用刀片將整本書切開成四～六等分。此時一等分大約是\n70~100 頁左右。\n2.用裁紙器，將每一份的書背切斷。此時，
```

資料向量化 & 存入向量資料庫

```python
# 利用 nomic-embed-text 模型，執行文本向量化
from langchain_community.embeddings import OllamaEmbeddings
from langchain_community.vectorstores import Chroma

embeddings = OllamaEmbeddings(model="nomic-embed-text")
vector_db = Chroma.from_documents(documents=splits, embedding=embeddings)
```

隨堂練習：資料分割與向量化

- 請依照前述投影片內容，撰寫並執行下列程式碼：

```python
# 將載入的文件以每段 1000 Tokens、重複 200 Tokens 的方式切分
# 可保證原文不會因切分而失去上下文
# 也可以讓 LLM 透過重複部分，得知誰是上文、誰是下文
from langchain.text_splitter import RecursiveCharacterTextSplitter

text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
splits = text_splitter.split_documents(all_docs)

for split in splits:
    print(split)

# 利用 nomic-embed-text 模型，執行文本向量化
from langchain_community.embeddings import OllamaEmbeddings
from langchain_community.vectorstores import Chroma

embeddings = OllamaEmbeddings(model="nomic-embed-text")
vector_db = Chroma.from_documents(documents=splits, embedding=embeddings)
```

定義 RAG 與 LLM 相關函數

- (1) 定義 LLM 詢問函數

```python
# 定義一個能丟入「使用者問題」與「RAG 抽取之相關文本」，用來詢問 LLM 的函數
def llm_ask(question, rag_contents):
    # 製作提問用的提詞（Prompt）
    formatted_prompt = f"Question: {question}\n\nContext: {rag_contents}"
    # 丟入 LLM，取得答案
    msg = llm.invoke(formatted_prompt)
    return msg
```

定義 RAG 與 LLM 相關函數

- (2) 定義 RAG 相關資料抽取函數

```python
# 定義向量資料庫抽取物件
retriever = vector_db.as_retriever()

# 根據與使用者問題的相似度，抽取相關資料
def get_rag_docs(question):
    # 依照與問題的相似度，抽取相關文本
    retrieved_docs = retriever.invoke(question)
    # 將所有相關文本，以空行 \n\n 黏成一個大字串
    relavent_contents = "\n\n".join(doc.page_content for doc in retrieved_docs)

    # 將問題 + RAG 相關文本，一口氣丟入 LLM，取得答案
    return llm_ask(question, relavent_contents)
```

1
2
3
4
5

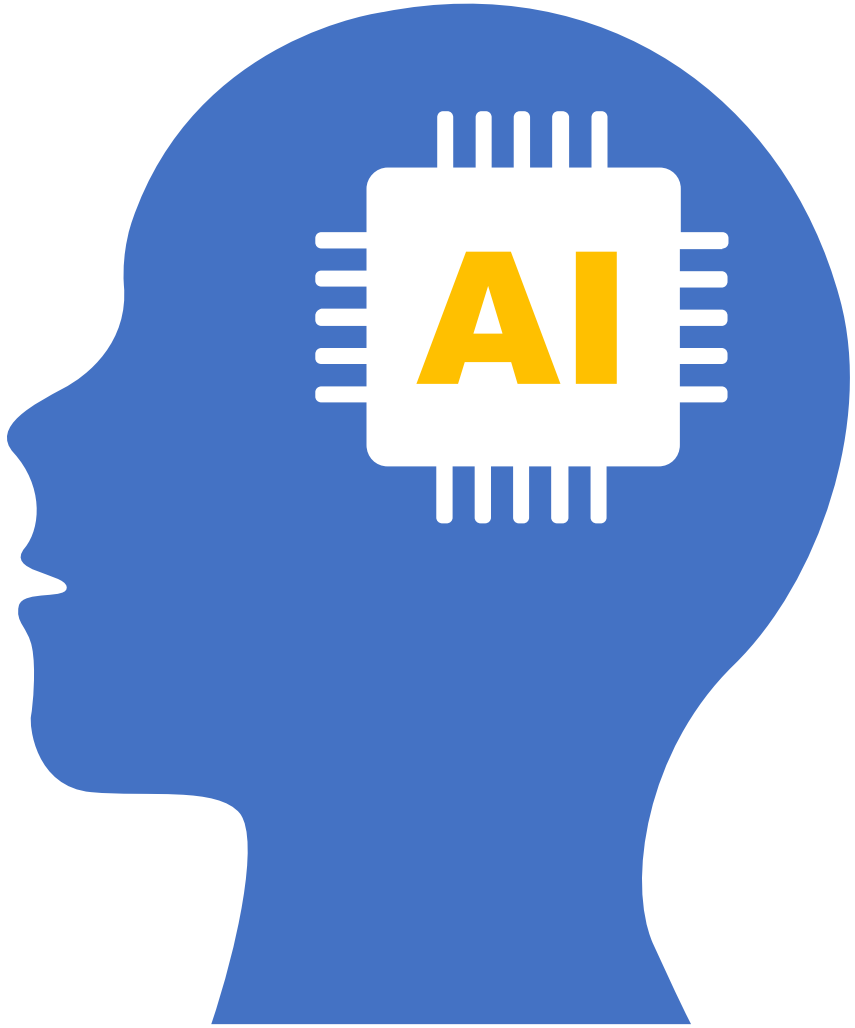
隨堂練習：LLM 與 RAG 函數定義

- 請依照前述投影片內容，撰寫並執行下列程式碼：

```python
# 定義一個能丟入「使用者問題」與「RAG 抽取之相關文本」，用來詢問 LLM 的函數
def llm_ask(question, rag_contents):
    # 製作提問用的提詞（Prompt）
    formatted_prompt = f"Question: {question}\n\nContext: {rag_contents}"
    # 丟入 LLM，取得答案
    msg = llm.invoke(formatted_prompt)
    return msg

# 定義向量資料庫抽取物件
retriever = vector_db.as_retriever()

# 根據與使用者問題的相似度，抽取相關資料
def get_rag_docs(question):
    # 依照與問題的相似度，抽取相關文本
    retrieved_docs = retriever.invoke(question)
    # 將所有相關文本，以空行 \n\n 黏成一個大字串
    relavent_contents = "\n\n".join(doc.page_content for doc in retrieved_docs)

    # 將問題 + RAG 相關文本，一口氣丟入 LLM，取得答案
    return llm_ask(question, relavent_contents)
```

RAG 實例講解

- 環境安裝
- 直接連上大語言模型（不使用 RAG）
- 整合大語言模型與 RAG
- **以圖形介面互動**



定義圖形介面

```python
# 用 Gradio 建立圖形介面，並用 get_rag_docs 做為處理函數
import gradio as gr

gui = gr.Interface(
    fn=get_rag_docs,
    inputs=gr.Textbox(lines=2, placeholder="在此輸入您的問題..."),
    outputs="text",
    title="紀老師答客問",
    description="請在這邊輸入您想提問的問題",
)
```

← 會收取 inputs= 傳回來的數值

啟動圖形介面

```
12  # 啟動圖形介面
13  gui.launch()
```

隨堂練習：以圖形介面互動

- 請依照前述投影片內容，撰寫並執行下列程式碼：

```python
# 用 Gradio 建立圖形介面, 並用 get_rag_docs 做為處理函數
import gradio as gr

gui = gr.Interface(
    fn=get_rag_docs,
    inputs=gr.Textbox(lines=2, placeholder="在此輸入您的問題..."),
    outputs="text",
    title="紀老師答客問",
    description="請在這邊輸入您想提問的問題",
)

# 啟動圖形介面
gui.launch()
```

本章總結

- **大語言模型的困境**
  - 無法回答太新的資料。
  - 無法回答太專業、特定領域的資料。

- **檢索增強式文本生成（RAG）**
  - 利用網路搜索、資料庫文本，補強訓練時語料庫太舊、或太廣泛的問題。

- **RAG 資料來源的生成步驟**
  - 資料載入（Data Loading）
  - 資料分割（Data Chunking）
  - 資料嵌入化（Data Embedding）
  - 儲存於向量資料庫（Vector Databases）