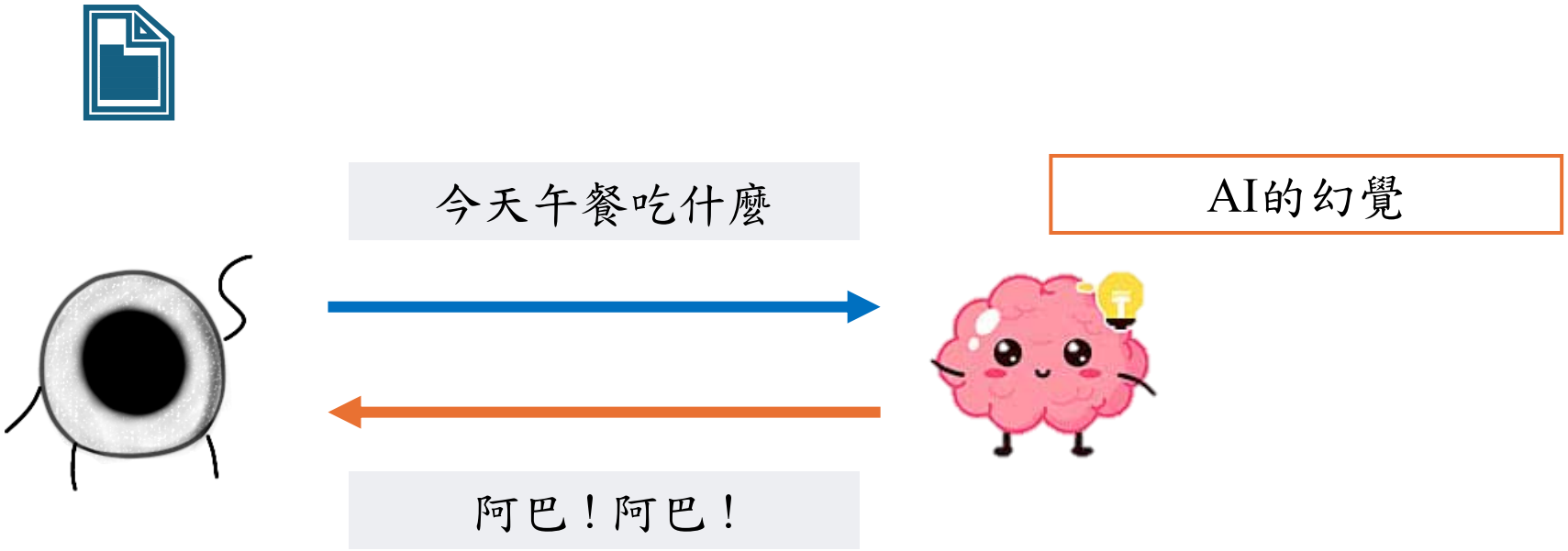
什麼是 RAG ??

主講：陳昱丞

國立聯合大學
NATIONAL UNITED UNIVERSITY

節能診斷服務中心
AI培訓課程

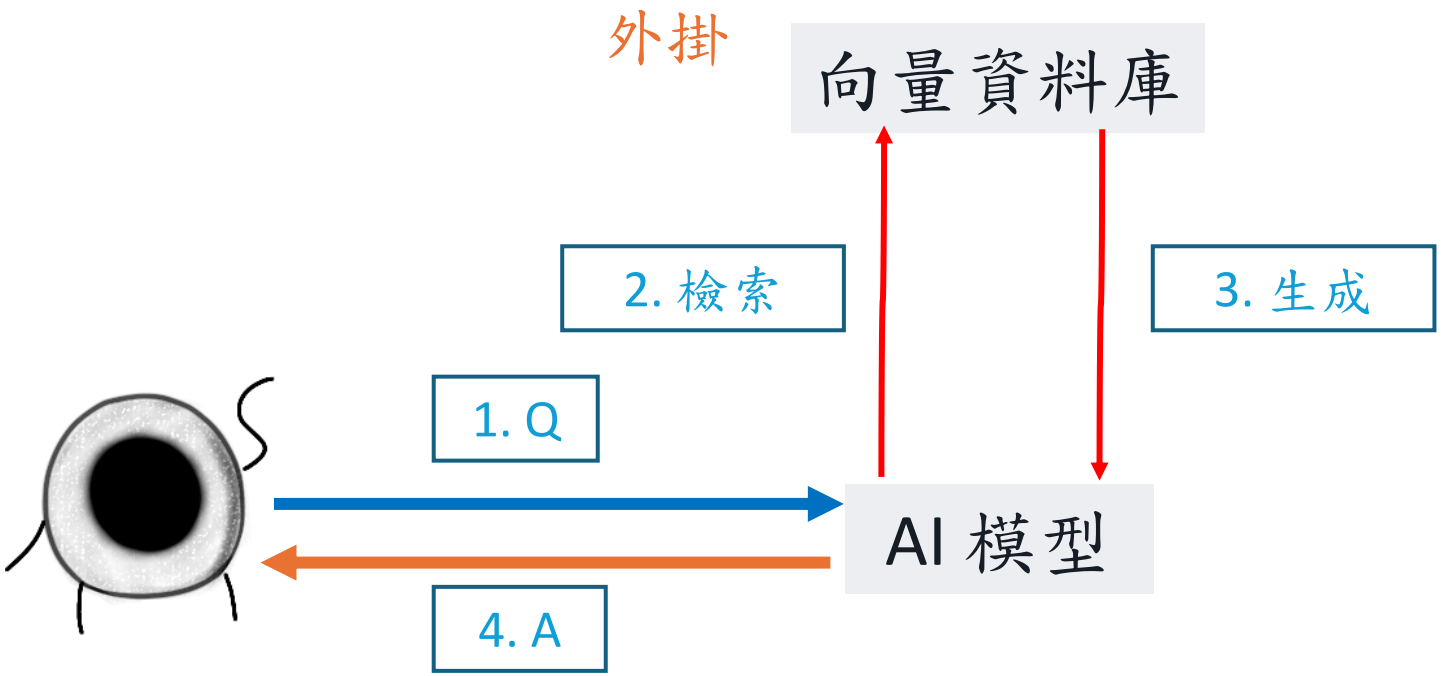
什麼是 RAG ？



什麼是 RAG ？

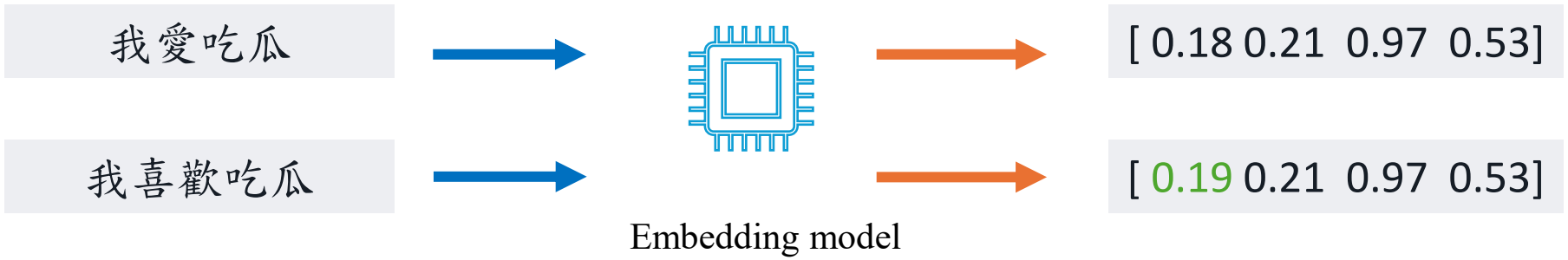
RAG (檢索增強生成)

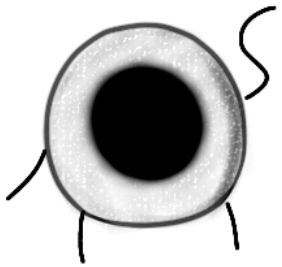
Retrieval Augmented Generation



什麼是 RAG ？

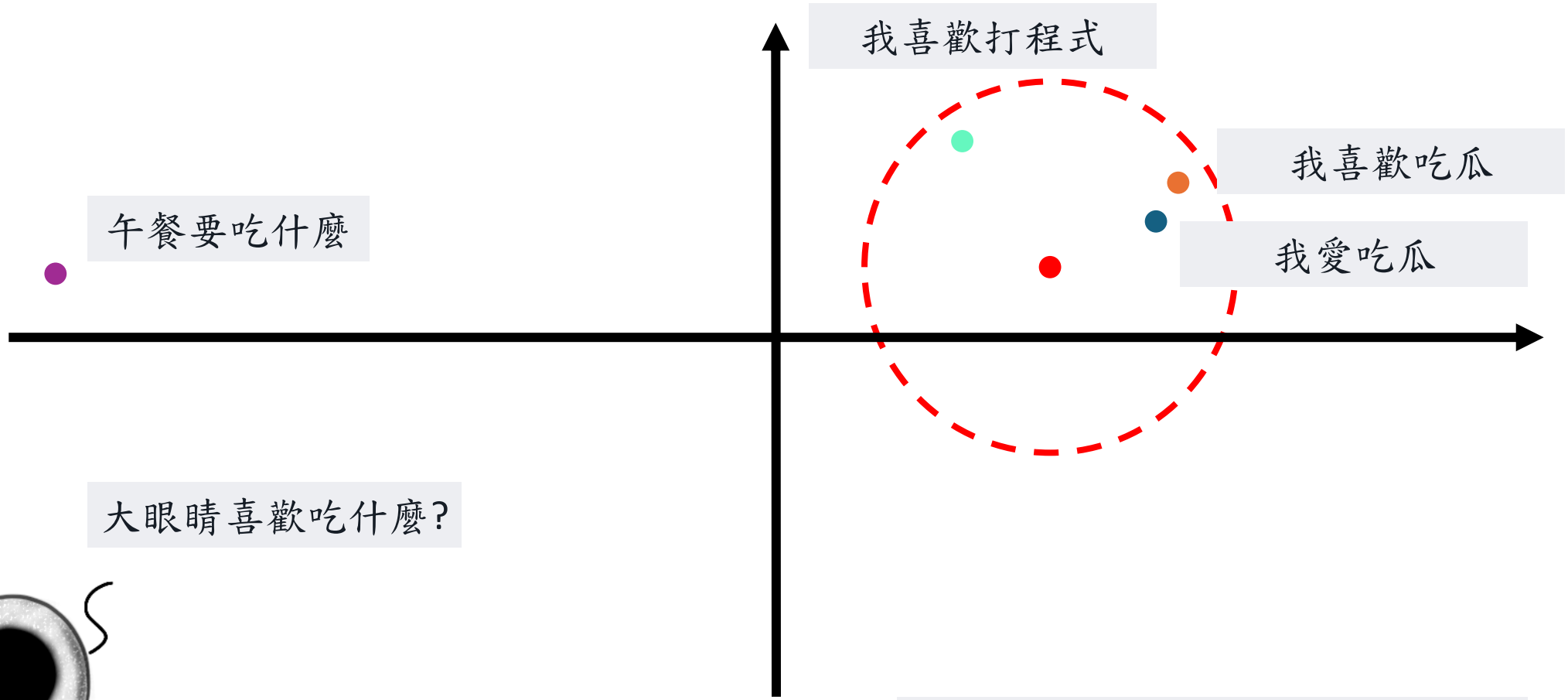
對原始文檔壓縮

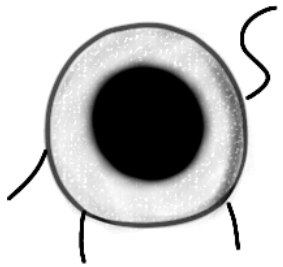




透過向量的距離判斷
文字是否相關

什麼是 RAG ？





真正的 Embedding model
是 1000維 甚至 3000多維

什麼是 RAG ？

我喜歡打程式

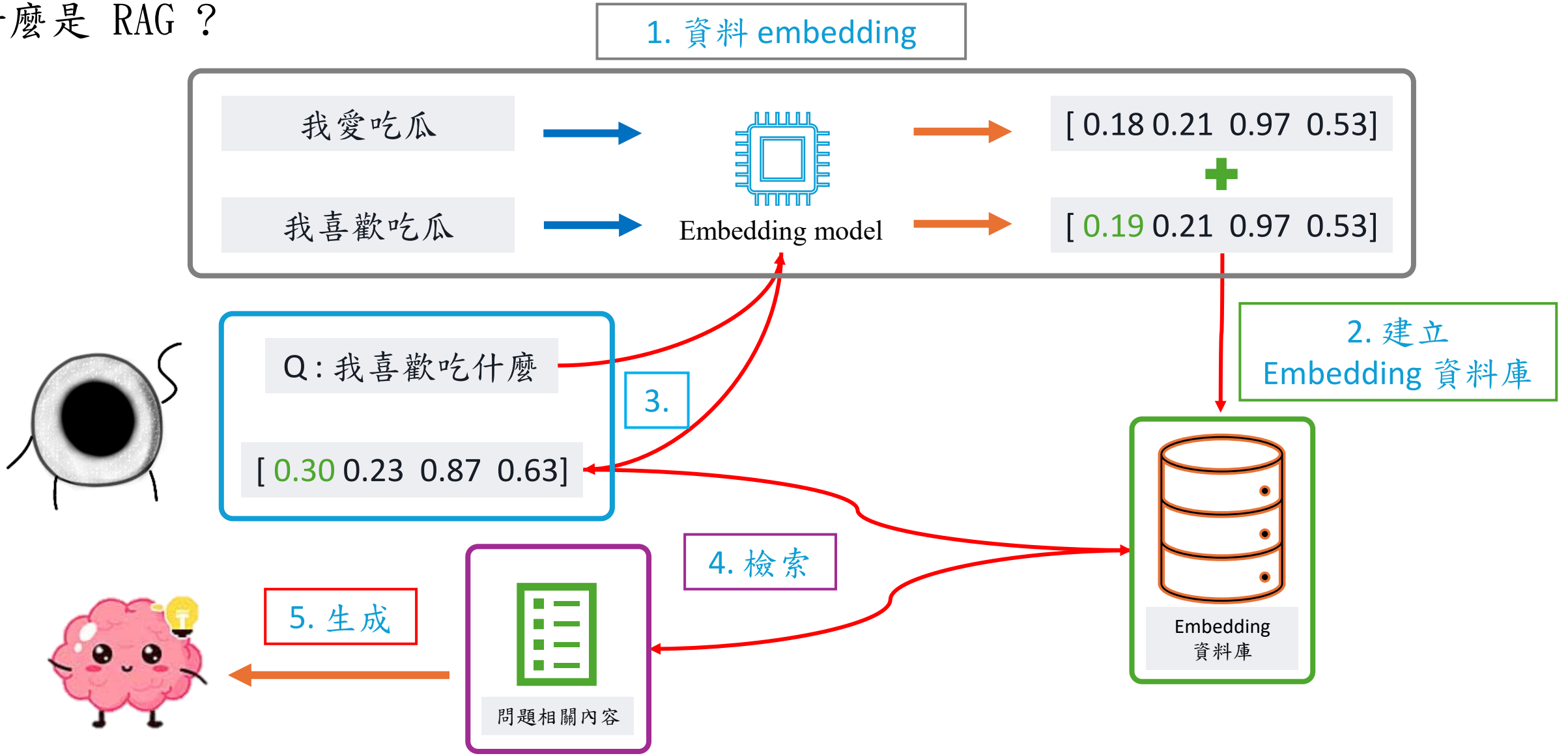
我喜歡吃瓜

我愛吃瓜

大眼睛喜歡吃什麼？

愛吃瓜

什麼是 RAG ？


1. 資料 embedding
我愛吃瓜
我喜歡吃瓜
Embedding model
[0.18 0.21 0.97 0.53]
[0.19 0.21 0.97 0.53]
2. 建立
Embedding 資料庫
Q：我喜歡吃什麼
[0.30 0.23 0.87 0.63]
3.
4. 檢索
5. 生成
問題相關內容
Embedding
資料庫

申請 Google Gemini API

主講：陳昱丞

國立聯合大學
NATIONAL UNITED UNIVERSITY

進入 Google Gemini API 網站

使用Chrome 或 其他瀏覽器

- 確認以登入自己的Google帳號

Google帳號

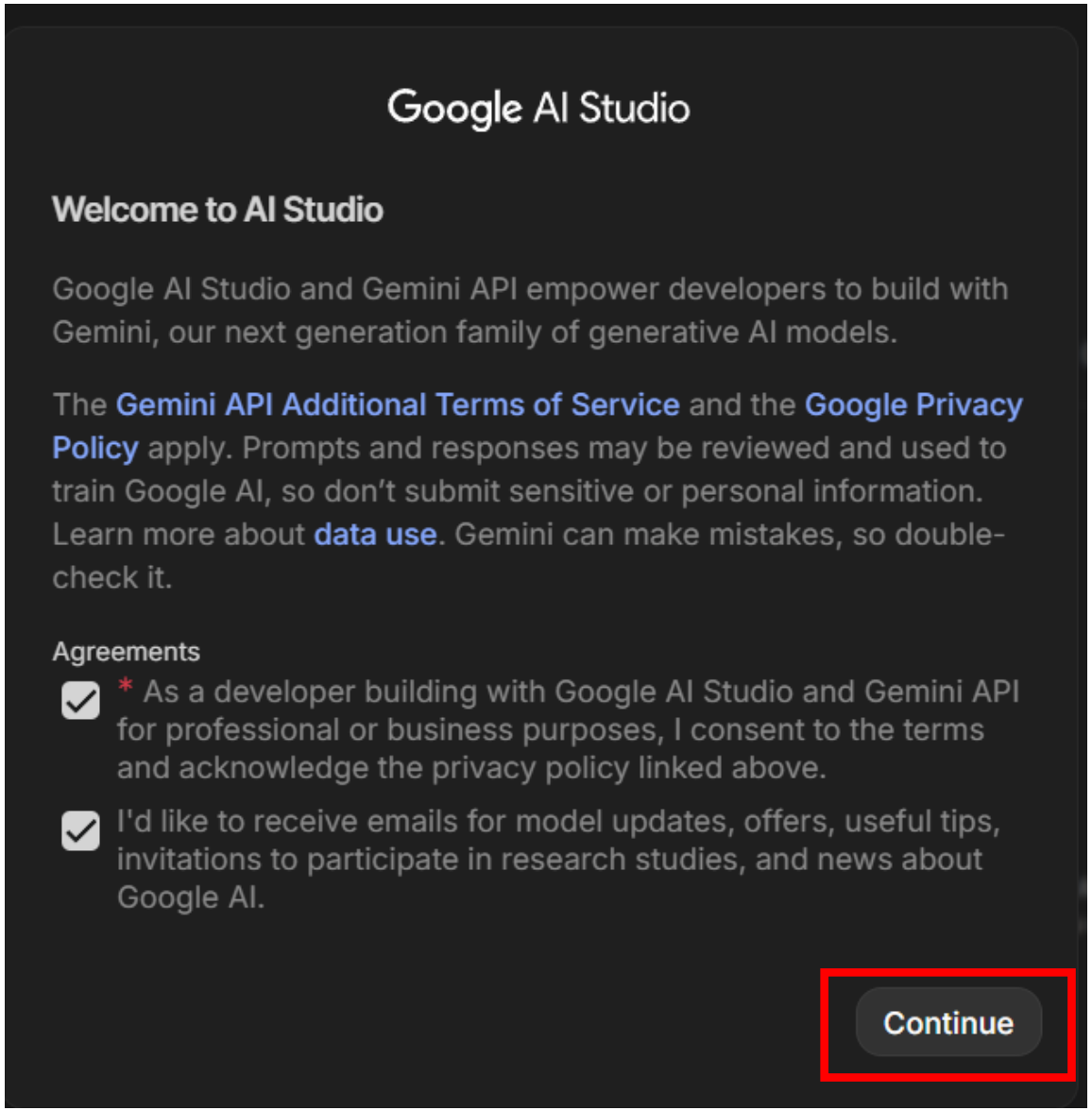
- 個人的gmail，非學校/組織帳號

https://ai.google.dev/gemini-api/docs/api-key?hl=zh-tw

取得Gemini API Key

開始使用 Gemini

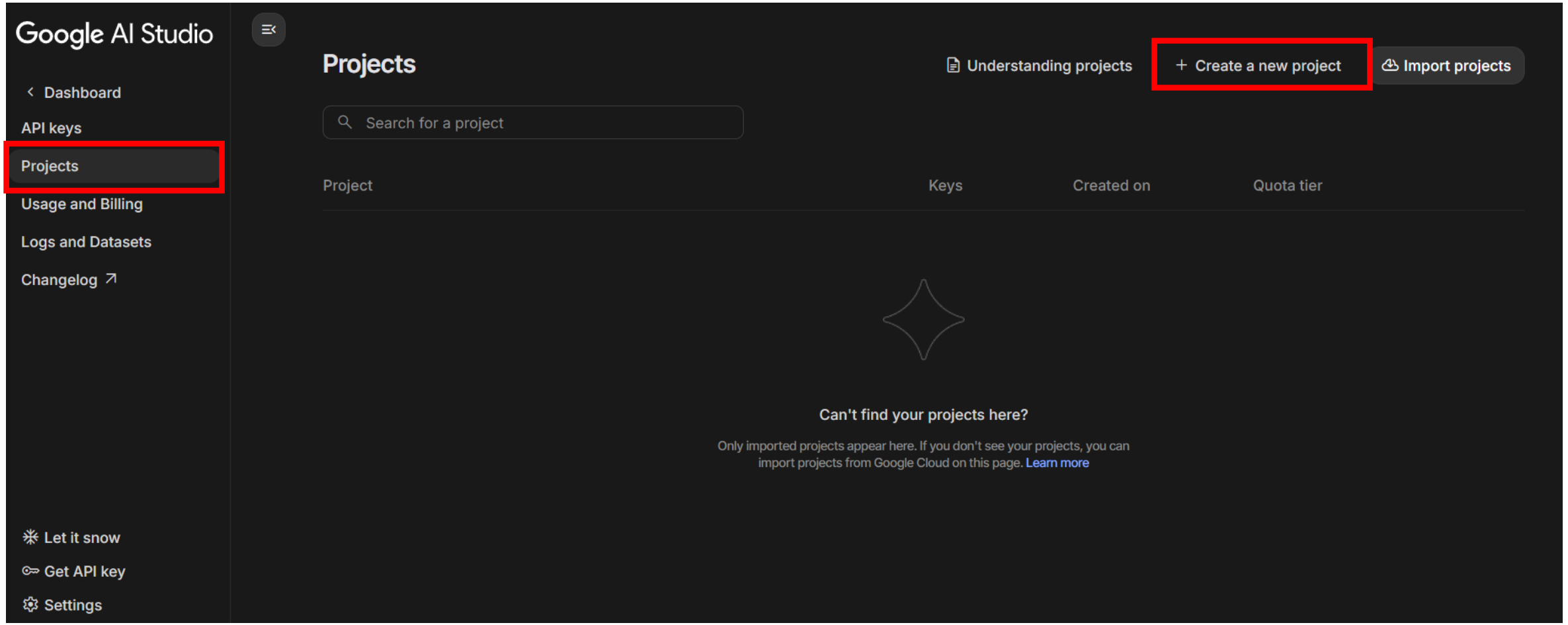
打勾後，按下 Continue



新增專案

1.進入 Projects 中

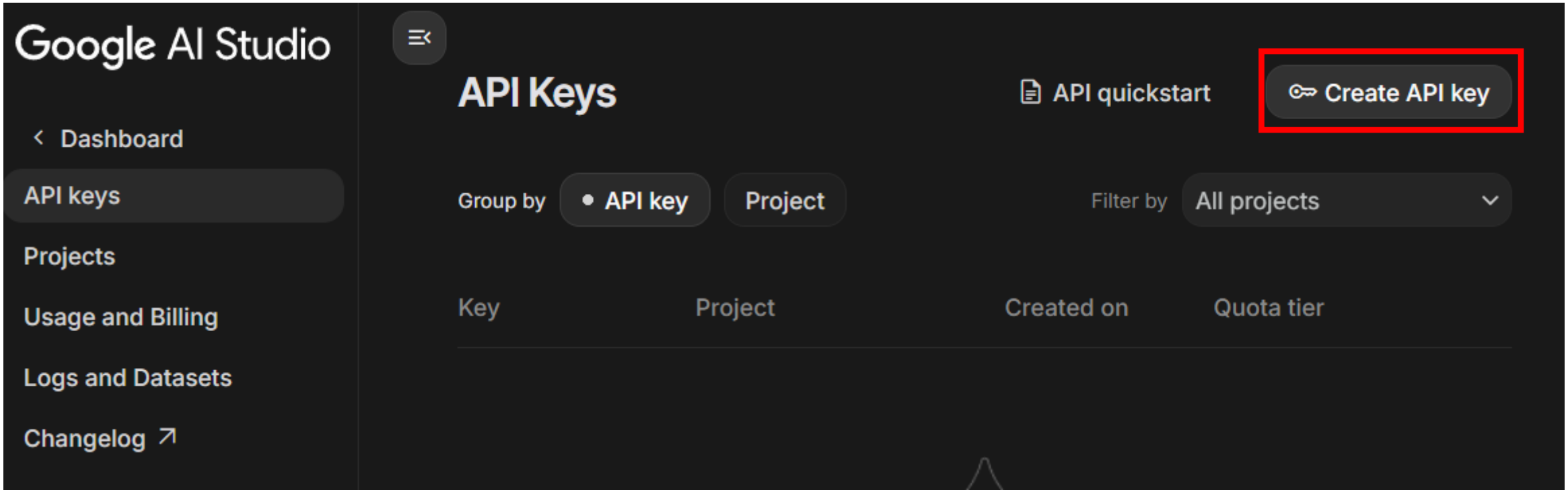
2.建立新的專案



建立API Keys

1.進入 API Keys 中

2. 按下「Create API key」

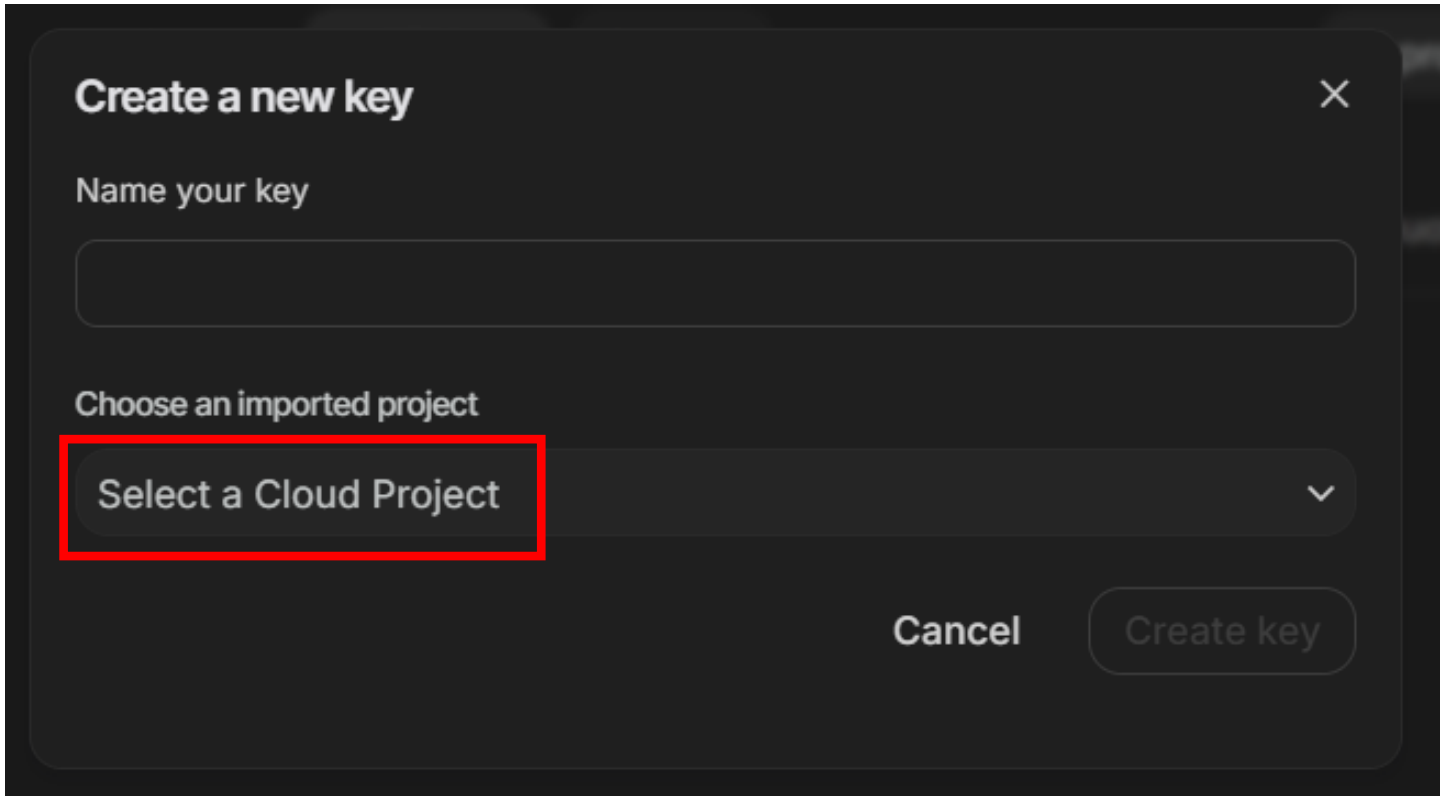


在什麼專案中使用

1.填寫Keys的名稱

2.選擇剛剛建立好的Projet

3.產生API key後，即可複製使用



注意收費規則

2024/5/2後

- 專案有開啟付費：會被收費
- 專案無開啟付費：免費使用

• 規則可能改變，以官方網站最新公告為主

Gemini 2.5 Pro

Gemini 2.5 Pro

`gemini-2.5-pro`

在 Google AI Studio 中試用

Google 最先進的多功能模型，擅長程式設計和複雜的推理工作。

標準　批次

	免費方案	付費層級，每 100 萬個權杖的費用 (美元)
輸入價格	無須支付費用	$1.25，提示詞 <= 20 萬個權杖 $2.50，提示詞 > 20 萬個權杖
輸出價格 (含思考權杖)	無須支付費用	$10.00 美元，提示 <= 20 萬個權杖 $15.00 美元，提示 > 20 萬個權杖
脈絡快取價格	無法使用	$0.125，提示 <= 20 萬個權杖 $0.25，提示 > 20 萬個權杖 每小時每 100 萬個權杖$4.50 美元 (儲存空間價格)
以 Google 搜尋建立基準	無法使用	1,500 個 RPD (免費)，之後每 1,000 個基礎提示 $35 美元
利用 Google 地圖建立基準	無法使用	10,000 個 RPD (免費)，之後每 1,000 個基礎提示 $25 美元
用於改善我們的產品	是	否

https://ai.google.dev/gemini-api/docs/pricing?hl=zh-tw

Anaconda 安裝

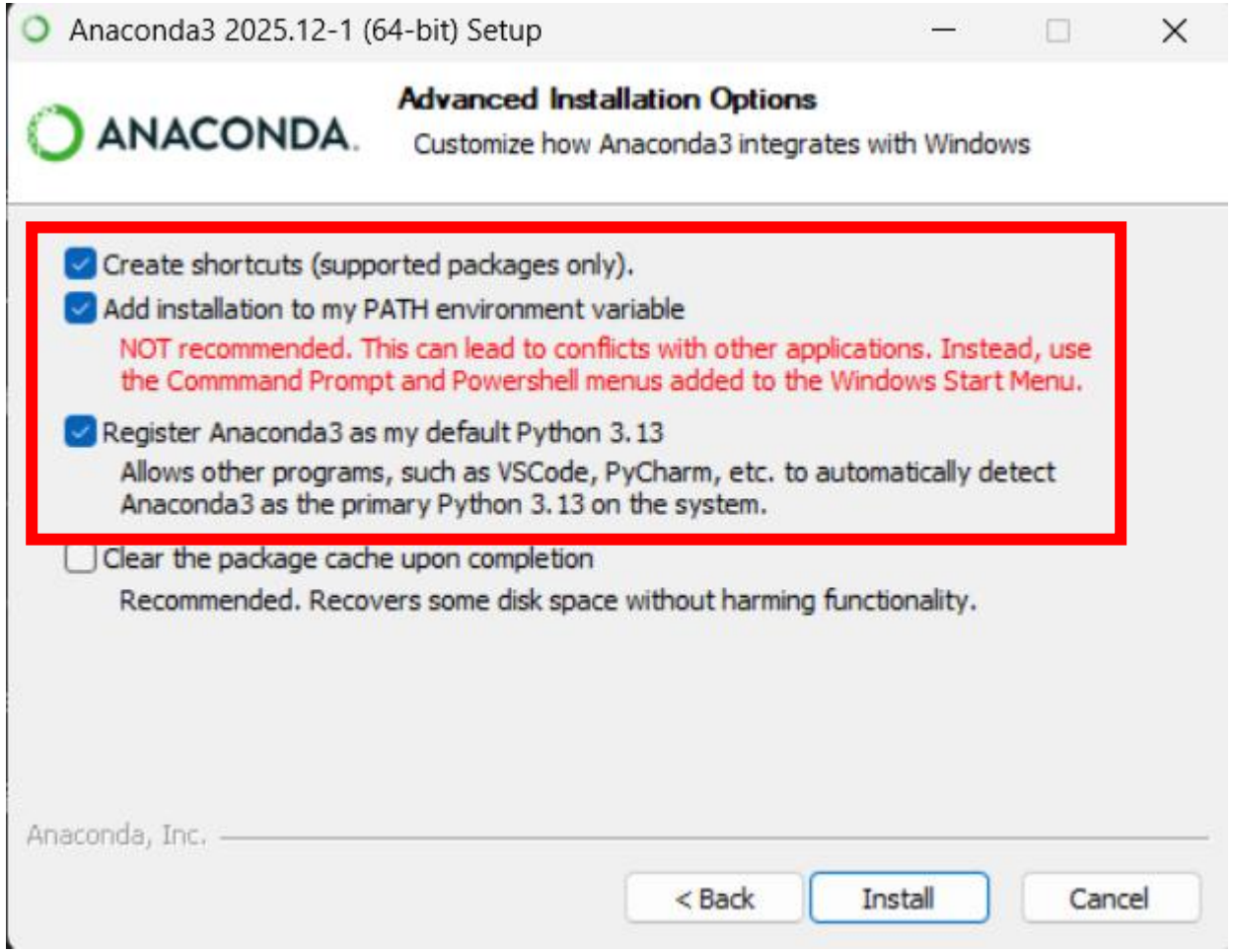
主講：陳昱丞

國立聯合大學
NATIONAL UNITED UNIVERSITY

Anaconda 安裝

安裝Anaconda

- 下載對應作業系統的版本
- 右方紅框一定要勾選



Vs code 安裝

主講：陳昱丞

國立聯合大學
NATIONAL UNITED UNIVERSITY

本機開發環境選擇

Microsoft Visual Studio Code

安裝擴充套件與設定

按下左邊Extensions圖示  或Ctrl + Shift + X

- Chinese (Traditional) Language Pack for Visual Studio Code
- Python
- Jupyter

設定 Ctrl+滑鼠滾軸 控制編輯器字體大小

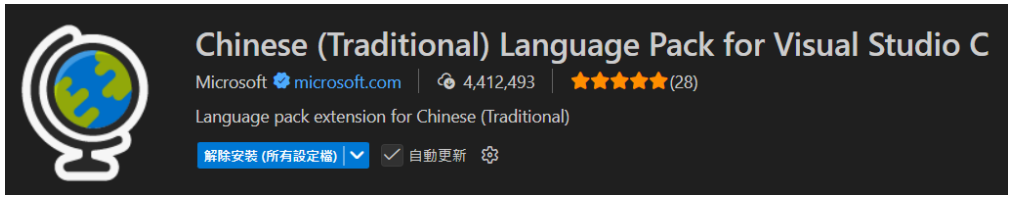
- 檔案–> 喜好設定–> 設定
- editor.mouseWheelZoom

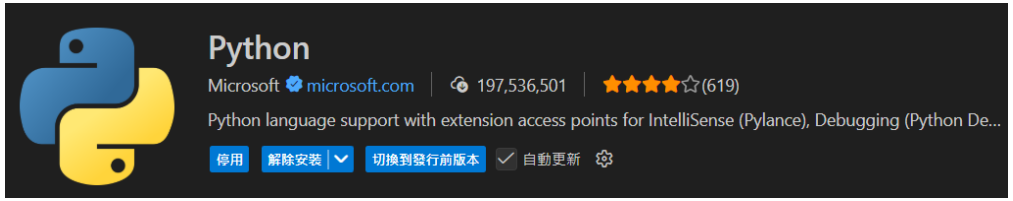
設定 Shift + Enter 執行輸出

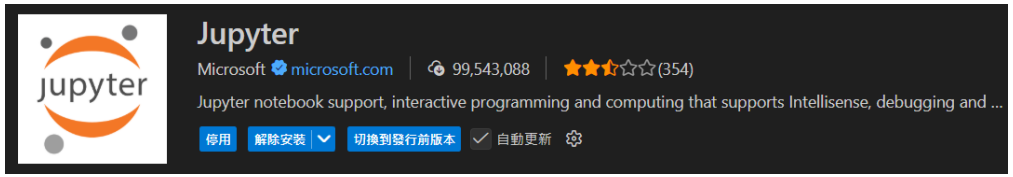
- Execute Selection

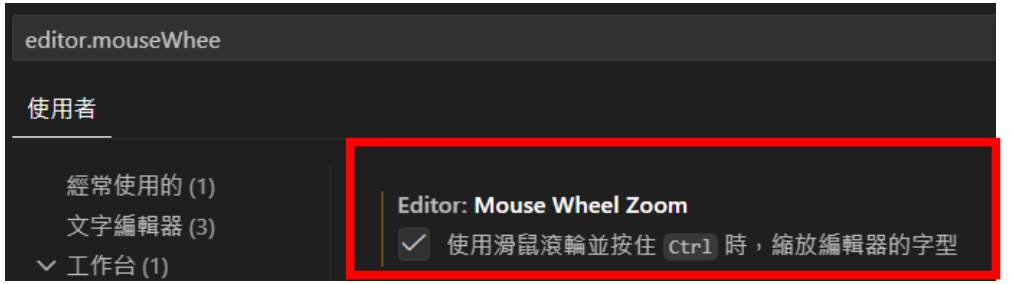
設定編輯時自動儲存

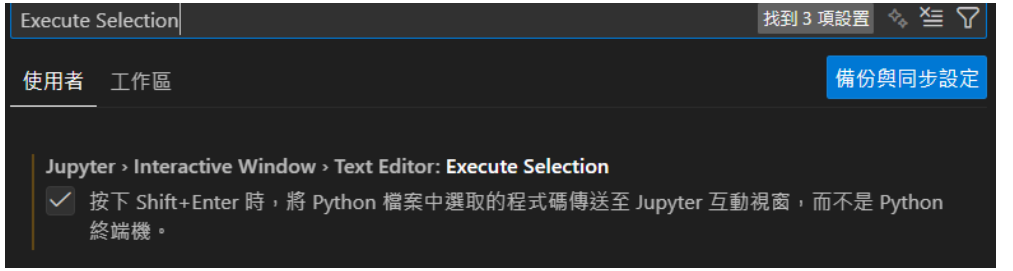
- 檔案–> 自動儲存







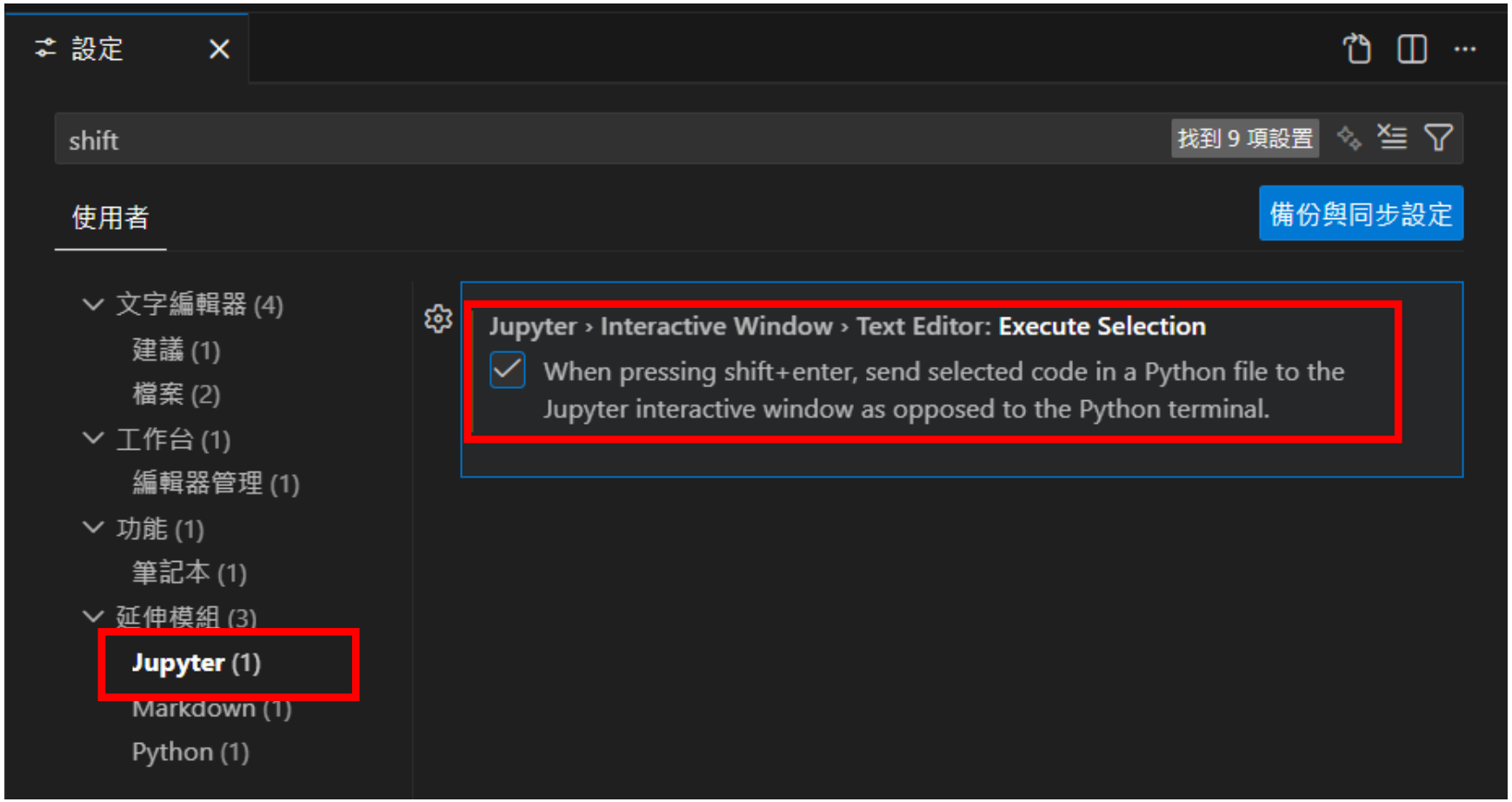




設定執行輸出(請先確認是否已安裝Jupyter延伸套件)

檔案-> 喜好設定-> 設定

搜尋shift

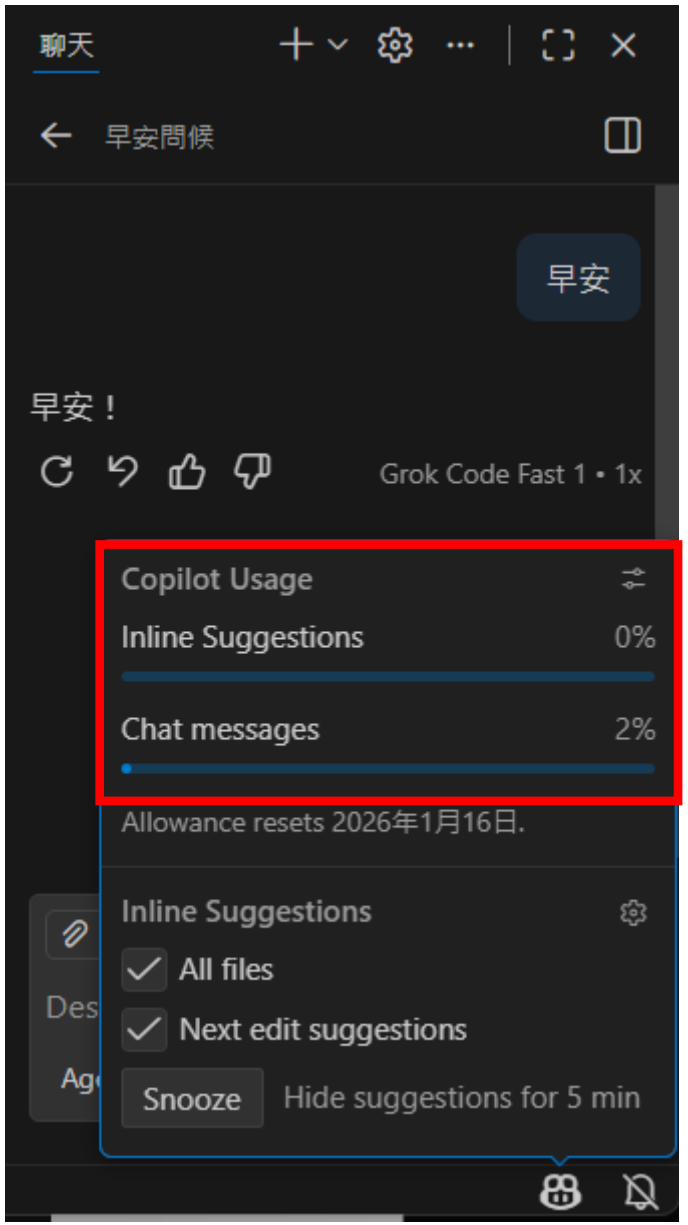


啟用GitHub Copilot

上方訊息圖示-> 開啟聊天

右邊聊天視窗傳送訊息測試(有次數的限制)

第一次需要登入GitHub帳號並授權



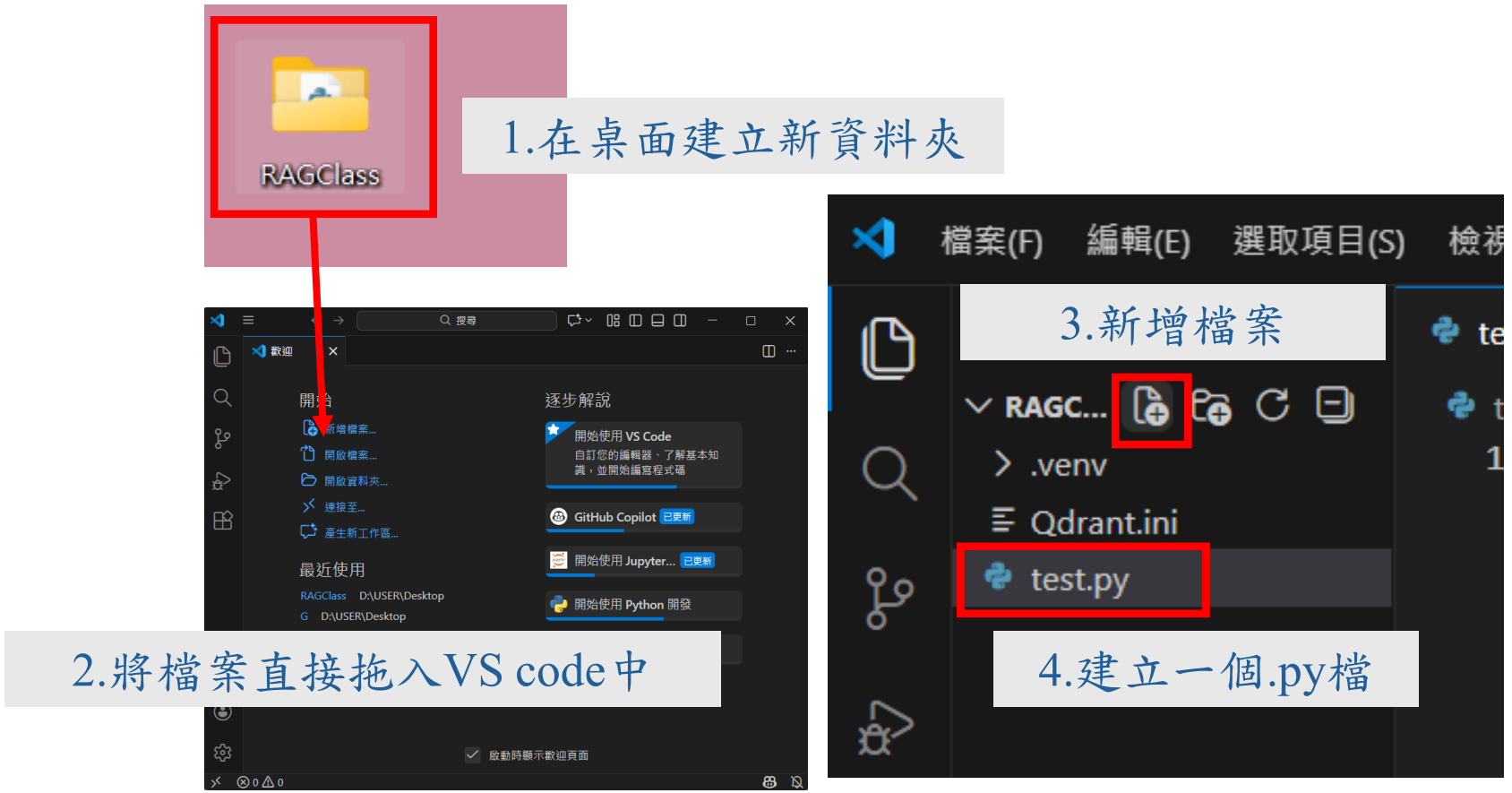
新建專案

新增資料夾

- RAGClass

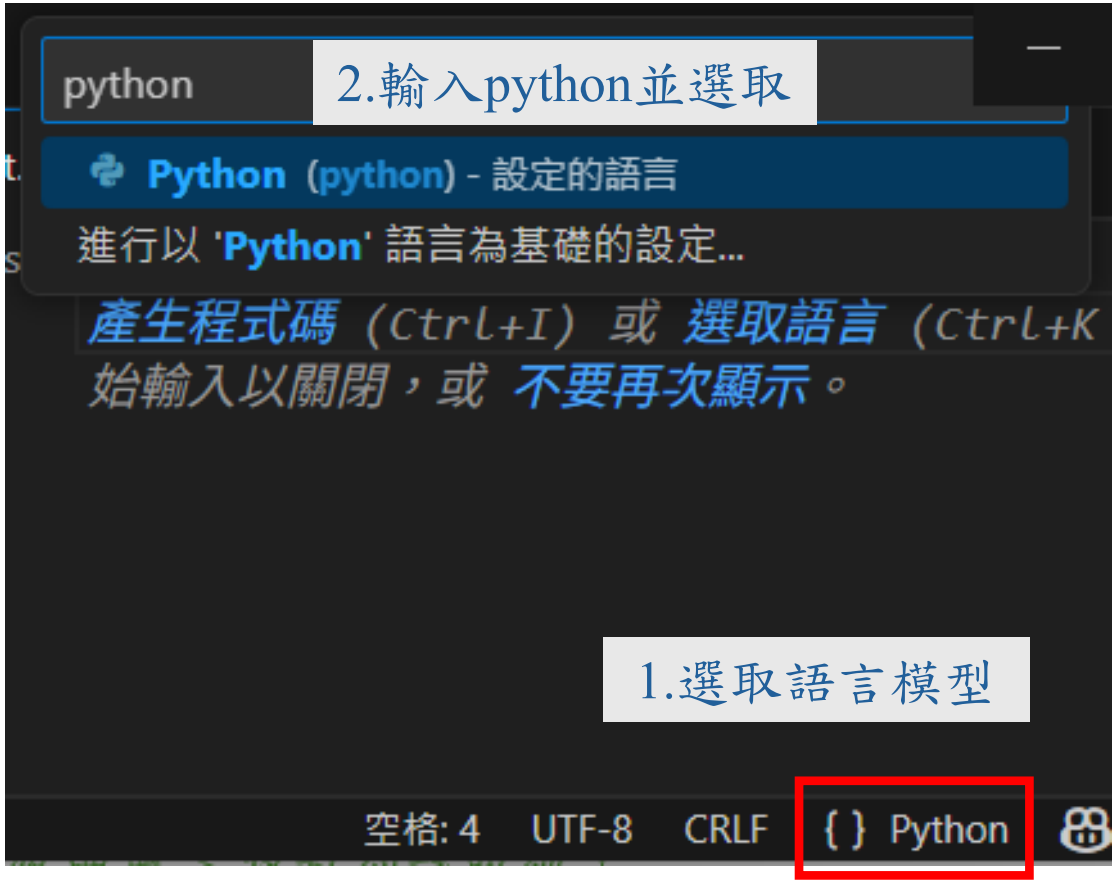
新增檔案

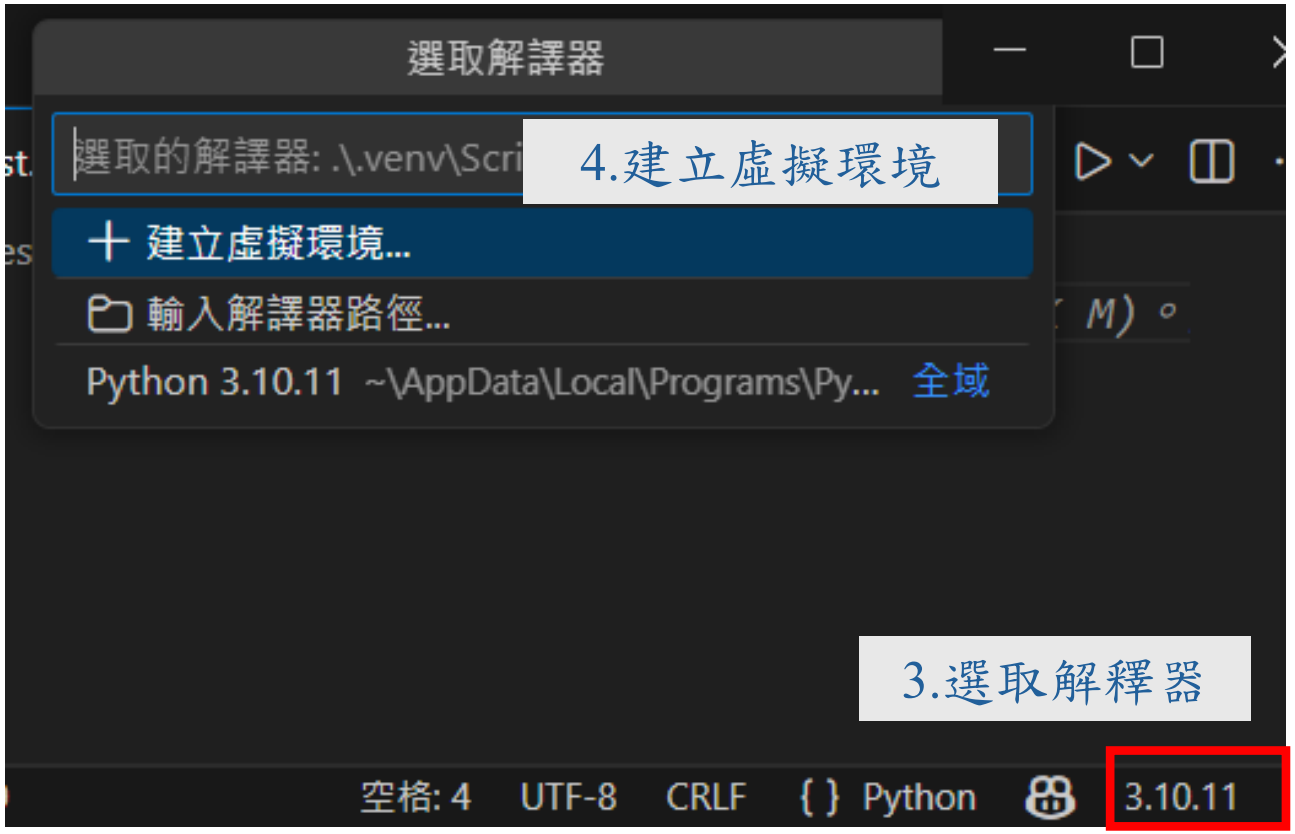
- test.py
- config.ini



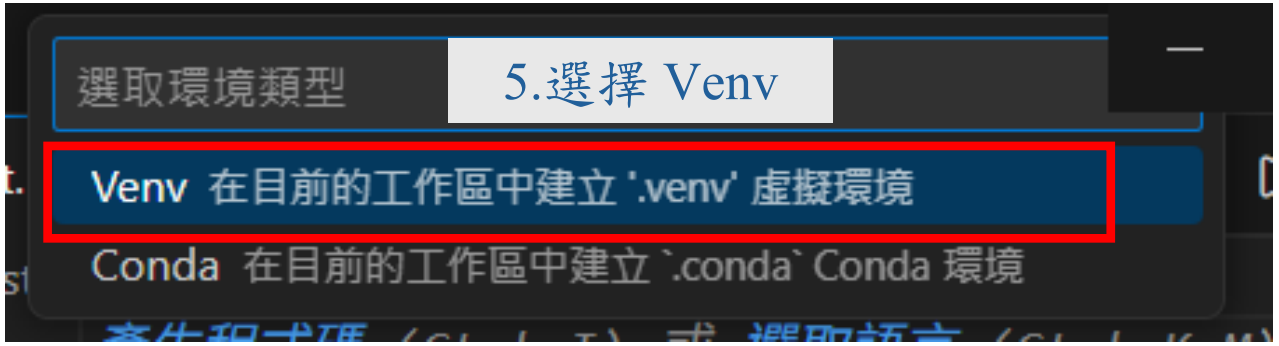
全域環境的套件安裝

透過VS Code 建立虛擬環境



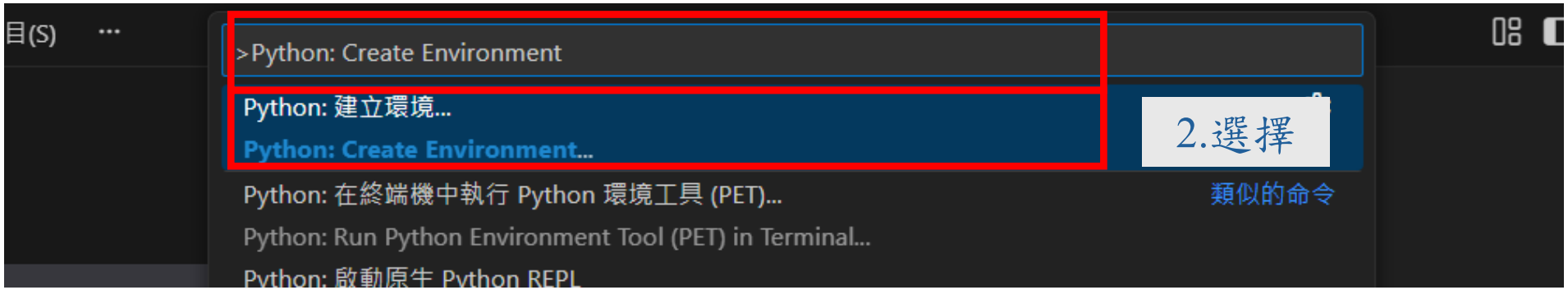


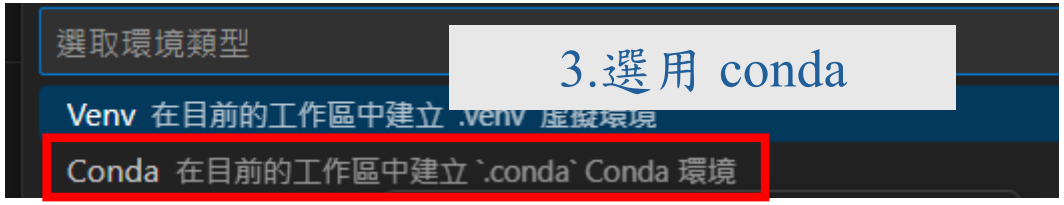
全域環境的套件安裝(用Venv建環境)

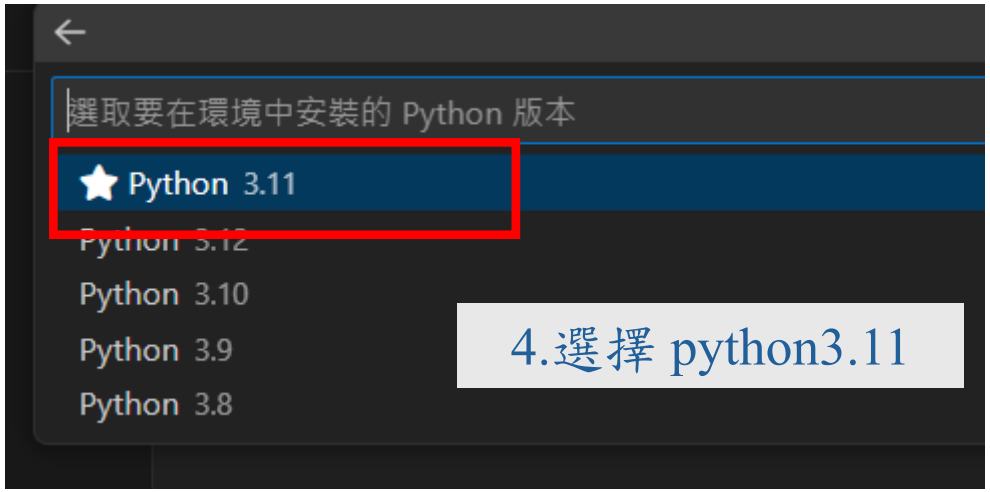


全域環境的套件安裝 (用conda建環境) (今天用這個來建環境)

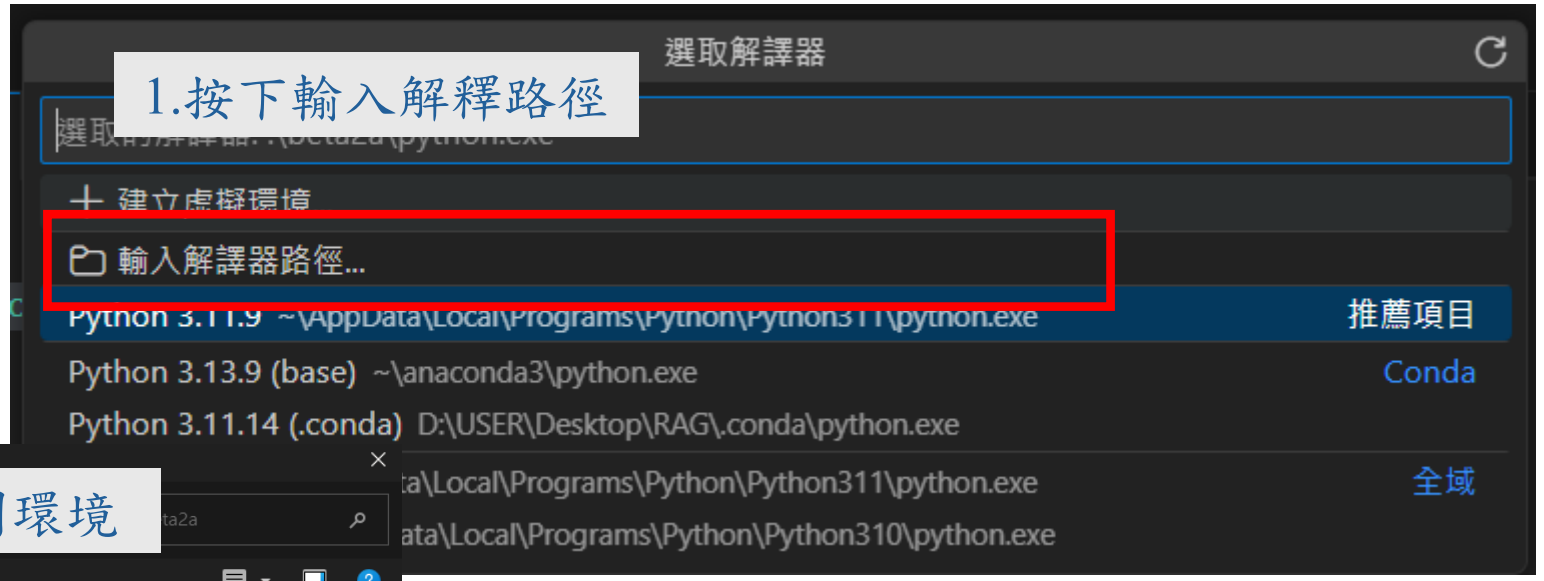
1.按下 ctrl + shift + p 輸入: Python: Create Environment

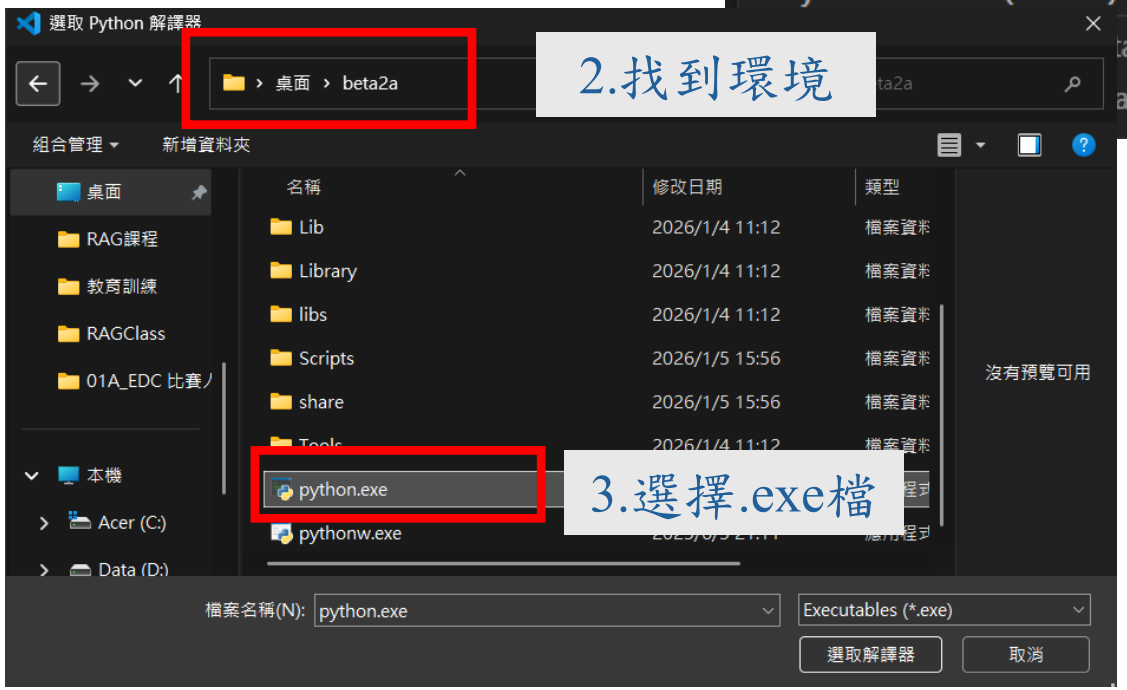






全域環境的套件安裝 (導入建環境)





新增 config.ini

```
[Gemini]
API_KEY = XXXXXXX (自己key)
```

安裝套件

https://github.com/BONE-22/RAG..io/blob/main/test/test_API

1.出現黃色底線，表示系統找不到套件

```python
# Set up the config parser
config = ConfigParser()
config.read("config.ini")

from langchain_google_genai import ChatGoogleGenerativeAI
llm_gemini = ChatGoogleGenerativeAI(
    model="gemini-2.0-flash-lite",
    google_api_key=config["Gemini"]["API_KEY"]
)

user_input = "人生的意義是什麼？"

role_description = """
你是一個哲學家，請用繁體中文回答。
"""

messages = [
    ("system", role_description),
    ("human", user_input),
]

response_gemini = llm_gemini.invoke(messages)

print(f"問 : {user_input}")
print(f"Gemini : {response_gemini.content}")
```

2.選取要執行的程式碼(按 Shift + enter 執行)

已連線至 .venv (Python 3.10.11)

```python
from configparser import ConfigParser

# Set up the config parser
config = ConfigParser()
config.read("config.ini")

from langchain_google_genai import ChatGoogleGenerativeAI
llm_gemini = ChatGoogleGenerativeAI(
    model="gemini-2.0-flash-lite",
    google_api_key=config["Gemini"]["API_KEY"]
)
```

3.跳出 Error

```
ModuleNotFoundError                       Traceback (most recent call last)
File d:\USER\Desktop\RAGClass\test.py:7
      4 config = ConfigParser()
      5 config.read("config.ini")
----> 7 from langchain_google_genai import ChatGoogleGenerativeAI
      8 llm_gemini = ChatGoogleGenerativeAI(
      9     model="gemini-2.0-flash-lite",
     10     google_api_key=config["Gemini"]["API_KEY"]
     11 )

ModuleNotFoundError: No module named 'langchain_google_genai'
```

4.需要安裝套件

python ---- pip install 欲安裝的套件名稱

```
pip install -U langchain-google-genai
✓ 1m 17.1s
```

安裝套件完成

```
Collecting langchain-google-genai
  Downloading langchain_google_genai-4.1.2-py3-none-any.whl.metadata (2.7 kB)
Collecting filetype<2.0.0,>=1.2.0 (from langchain-google-genai)
  Using cached filetype-1.2.0-py2.py3-none-any.whl.metadata (6.5 kB)
Collecting google-genai<2.0.0,>=1.56.0 (from langchain-google-genai)
  Downloading google_genai-1.56.0-py3-none-any.whl.metadata (53 kB)
Collecting langchain-core<2.0.0,>=1.2.2 (from langchain-google-genai)
  Downloading langchain_core-1.2.5-py3-none-any.whl.metadata (3.7 kB)
Collecting pydantic<3.0.0,>=2.0.0 (from langchain-google-genai)
  Downloading pydantic-2.12.5-py3-none-any.whl.metadata (90 kB)
```

LangChain初探

test.py > ...

```python
from langchain_google_genai import ChatGoogleGenerativeAI
from configparser import ConfigParser

config = ConfigParser()
config.read("config.ini")

llm_gemini = ChatGoogleGenerativeAI(
    model="gemini-2.5-flash",
    google_api_key=config["Gemini"]["API_KEY"],
    max_retries=5
)


messages = [
    ("system", "你是一個哲學家，請用繁體中文回答。"),
    ("human", "人生的意義是什麼？"),
]

try:
    print("正在請求 Google API...")
    response = llm_gemini.invoke(messages)
    print(f"回答：\n{response.content}")
except Exception as e:
    print(f"連線失敗，請檢查 API Key 或模型名稱。錯誤訊息：\n{e}")
```

中斷 | 全部清除 | 重新啟動 | Jupyter 變數 | 儲存 | 匯出 | 展開 | 摺疊 | .venv (Python 3.10.11)

```
    response = llm_gemini.invoke(messages)
    print(f"回答：\n{response.content}")
except Exception as e:
    print(f"連線失敗，請檢查 API Key 或模型名稱。錯誤訊息：\n{e}")
```

[4] ✓ 14.9s

```
正在請求 Google API...
回答：
人生的意義是什麼？這是一個古老而又永恆的追問，自人類有意識以來便縈繞心頭，卻也始終沒有一個普世皆準的定論。作為一

以下是一些主要的哲學觀點，它們共同構成了我們對「人生意義」理解的複雜畫卷：

1. **意義的創造（存在主義）：**
   存在主義哲學家如沙特（Jean-Paul Sartre）認為，「存在先於本質」（existence precedes essence）。這意味著，我

2. **意義的賦予（宗教與超越性）：**
   對一部分人而言，人生的意義由超越性的存在所賦予。各種宗教信仰都提供了一套關於宇宙、生命起源與終極目的的敘事，

3. **意義的追求（目的論與幸福論）：**
   古希臘哲學家如亞里斯多德（Aristotle）提出，人生的最終目的是「幸福」（eudaimonia），這並非感官的愉悅，而是指

4. **意義的連結與關係（人本主義與社群主義）：**
   許多人本主義思想家認為，人生的意義深植於我們的關係之中。透過愛、友誼、家庭和社群的連結，我們感受歸屬、被愛和

5. **意義的體驗與過程（現象學與當下）：**
   另一種觀點則認為，意義不在於終極的目標或外在的賦予，而在於生命本身的體驗過程。每一個當下、每一次的感受、每一

**總結來說：**

人生的意義，或許不是一個單一、客觀的答案，而是一個由我們每個個體**主觀建構**的過程。它可能源於我們的信仰，植根於

正因為沒有一個絕對的答案，我們才擁有最大的自由去探索、去提問、去創造、去選擇，最終為自己的生命畫上獨特的色彩。這
```

API 回答問題

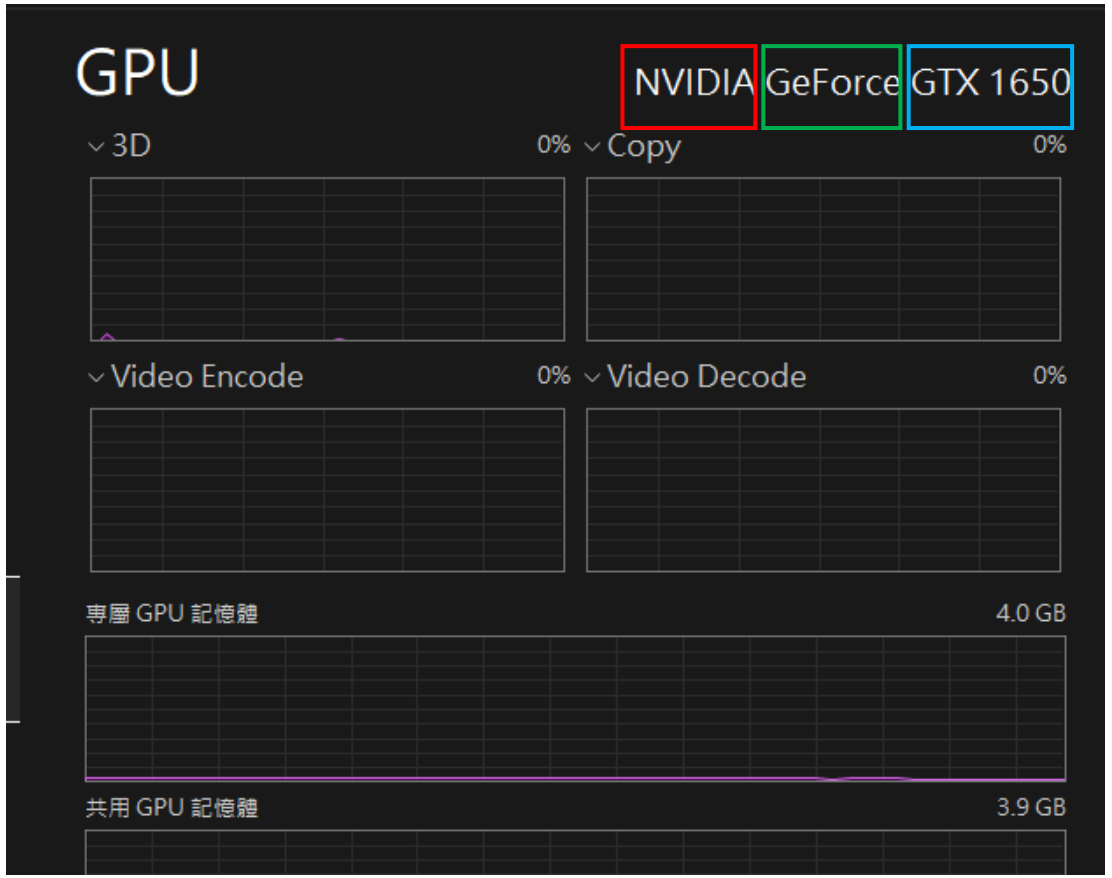
更新 NVIDIA 顯示卡

主講：陳昱丞

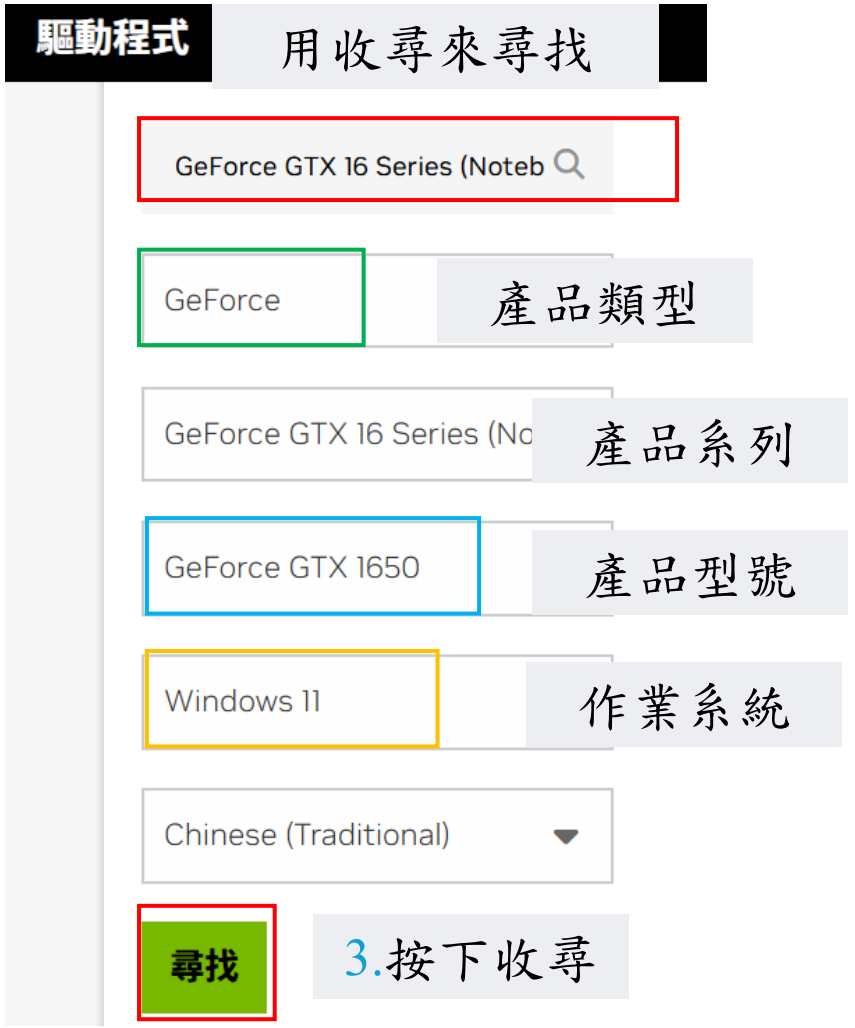
https://www.nvidia.com/zh-tw/drivers/

更新 NVIDIA 顯示卡 (下載驅動程式)

1. 打開工作管理員 看顯卡型號



2. 在官網找到電腦對應GPU型號並尋找



https://www.nvidia.com/zh-tw/drivers/

更新 NVIDIA 顯示卡 (下載驅動程式)

4. 選擇 ready 版本

5. 下載
安裝即可

更新 NVIDIA 顯示卡 (下載驅動程式)

https://github.com/BONE-22/RAG..io/blob/main/test/test_GPU

呼叫 Ollama 地端模型

主講：陳昱丞

https://ollama.com/

國立聯合大學
NATIONAL UNITED UNIVERSITY

節能診斷服務中心
AI培訓課程

呼叫 Ollama 地端模型 (LangChain初探)

安裝好後，會在右下角的工作列看到一個羊頭圖示，這代表 Ollama 已經在後台運行。

1. 開啟終端機 (按 ctrl + `)
2. 輸入 ollama pull bge-m3
3.開始安裝地端的模型

```
(.venv) D:\USER\Desktop\RAGClass>ollama pull bge-m3
pulling manifest
pulling daec91ffb5dd: 100%
pulling a406579cd136: 100%
pulling 0c4c9c2a325f: 100%
verifying sha256 digest
writing manifest
success
```

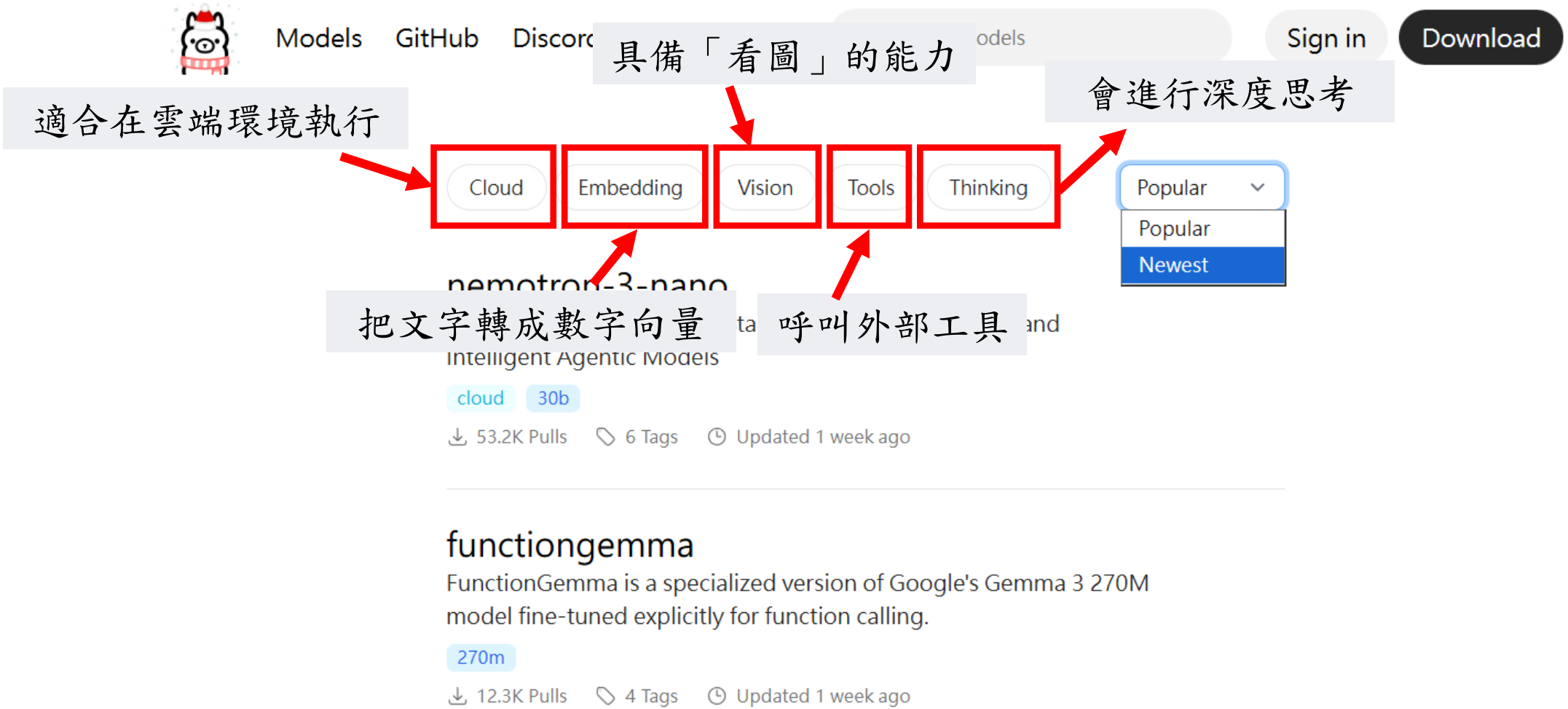
安裝成功

ollama pull beg-m3

呼叫安裝軟體名稱　執行動作　模型名稱(可變換)

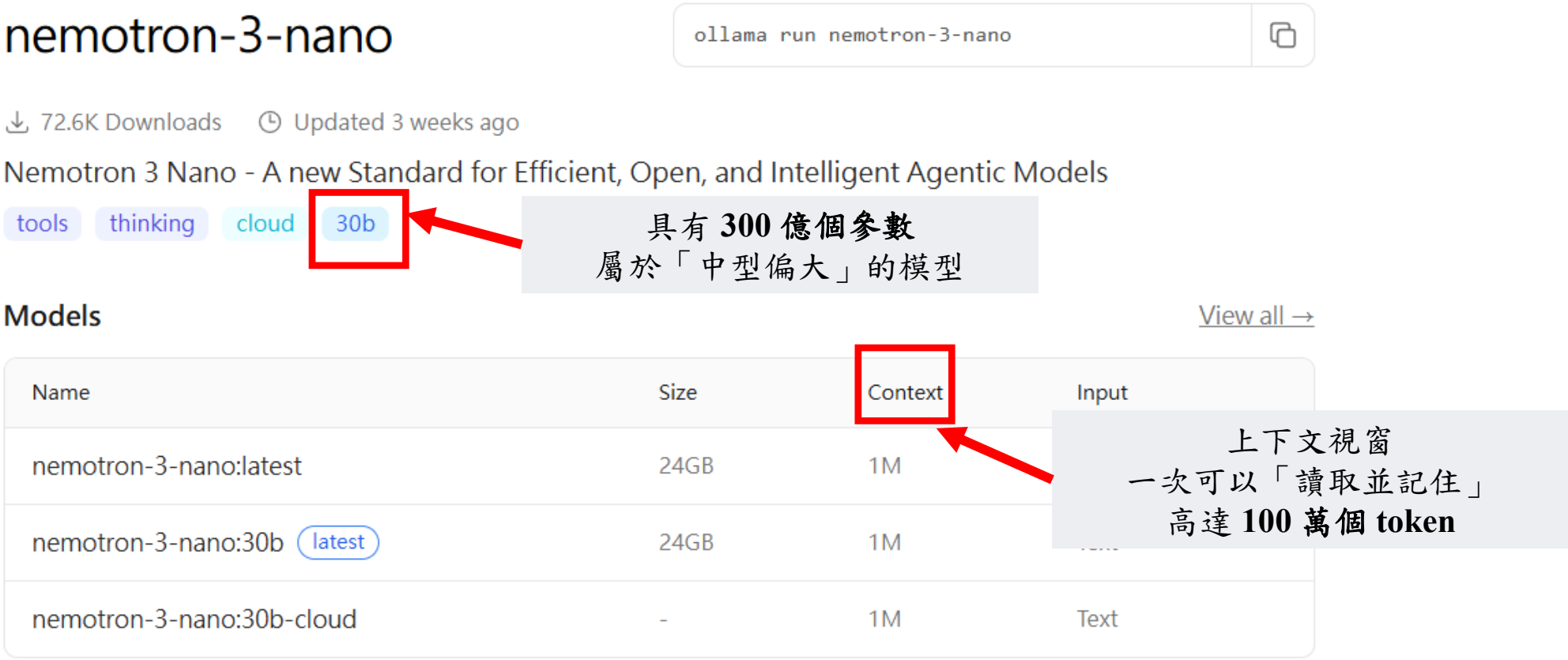
https://ollama.com/search

呼叫 Ollama 地端模型（LangChain初探）



https://ollama.com/search

呼叫 Ollama 地端模型（LangChain初探）



https://ollama.com/search

動物趣味事實問答(Excel)

主講：陳昱丞

https://github.com/BONE-22/RAG..io/tree/main/excel

國立聯合大學
NATIONAL UNITED UNIVERSITY

動物趣味事實問答（ 需求 ）

將資料轉換成向量資料庫

使用者輸入一句話後，能找到最相近的資料

建立專案

app_csv.py

requirements.txt

animal-fun-facts-dataset.csv

Excel 檔案下載：https://github.com/BONE-22/RAG..io/blob/main/excel/animal-fun-facts-dataset.csv

動物趣味事實問答（requirements.txt）

安裝需求模組

langchain	開發LLM應用的框架，負責串接不同功能。
pandas	讀取 CSV、Excel 或整理表格數據。
sentence-transformers	將文字轉換為「向量（Vector）」，讓電腦理解語義。
faiss-cpu	快速搜尋相似的文字內容。

直接執行一行指令，讓電腦自動安裝裡面列出的所有套件：pip install -r requirements.txt

一個一個安裝，可以在終端機輸入：pip install langchain pandas sentence-transformers faiss-cpu

動物趣味事實問答（app_csv. py）

https://github.com/BONE-22/RAG..io/blob/main/excel/app_csv.py

```
from langchain_community.vectorstores import FAISS
import pandas as pd
from langchain_huggingface import HuggingFaceEmbeddings
```

載入外掛模組

```
#讀取資料集
animal_data = pd.read_csv("animal-fun-facts-dataset.csv")
```

讀取檔案資料

```
#初始化 Embedding Function ( 使用模型將文字轉為向量 )
embedding_function = HuggingFaceEmbeddings(
    model_name="sentence-transformers/all-MiniLM-L6-v2"
)
```

Embedding(嵌入) 資料

```
# 準備 Metadata
metadatas = []
for i, row in animal_data.iterrows():
    metadatas.append(
        {
            "Animal Name": row["animal_name"],
            "Source URL": row["source"],
            # "Media URL": row["media_link"],
            # "Wikipedia URL": row["wikipedia_link"],
        }
    )
# 確保文本欄位是字串格式
animal_data["text"] = animal_data["text" ].astype(str)
```

幫資料貼上「**分類標籤**」
與內容相關的「額外資訊」

```
# 建立 FAISS 向量資料庫
faiss = FAISS.from_texts(animal_data["text"].to_list(), embedding_function, metadatas)
```

儲存並**快速搜尋**向量的工具

動物趣味事實問答（app_csv. py）

```
query = "What is ship of the desert?“
```

問題

```
k_count = 3  # 設定要找回前幾名結果
```

設定K值 (回傳 K 筆最相近的答案)

```
results = faiss.similarity_search_with_score(query, k=k_count)
```

顯示K值

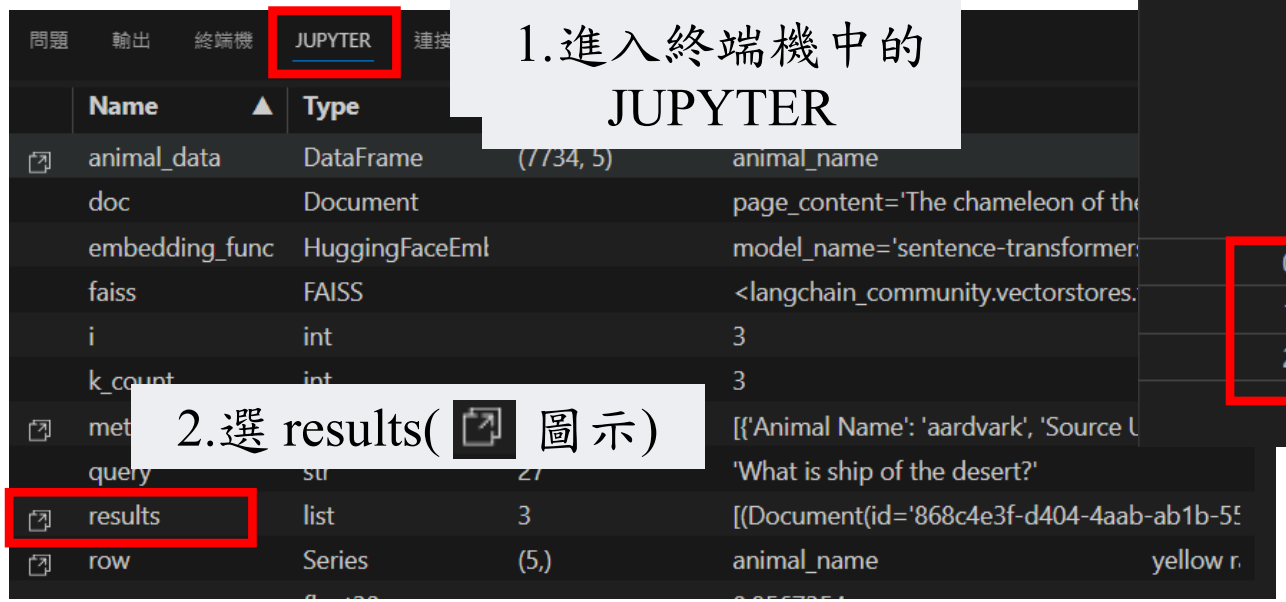
```
print(f"問題: '{query}'")
# 使用 enumerate 來標示 K 值 (從 1 開始)
for i, (doc, score) in enumerate(results, 1):
    print(f"【排名 K = {i}】")
    print(f"相似度分數：{score:.4f}")
    print(f"動物名稱: {doc.metadata.get('Animal Name')}")
    print(f"資料來源: {doc.metadata.get('Source URL')}")
)
```

顯示答案

動物趣味事實問答（結果確認）

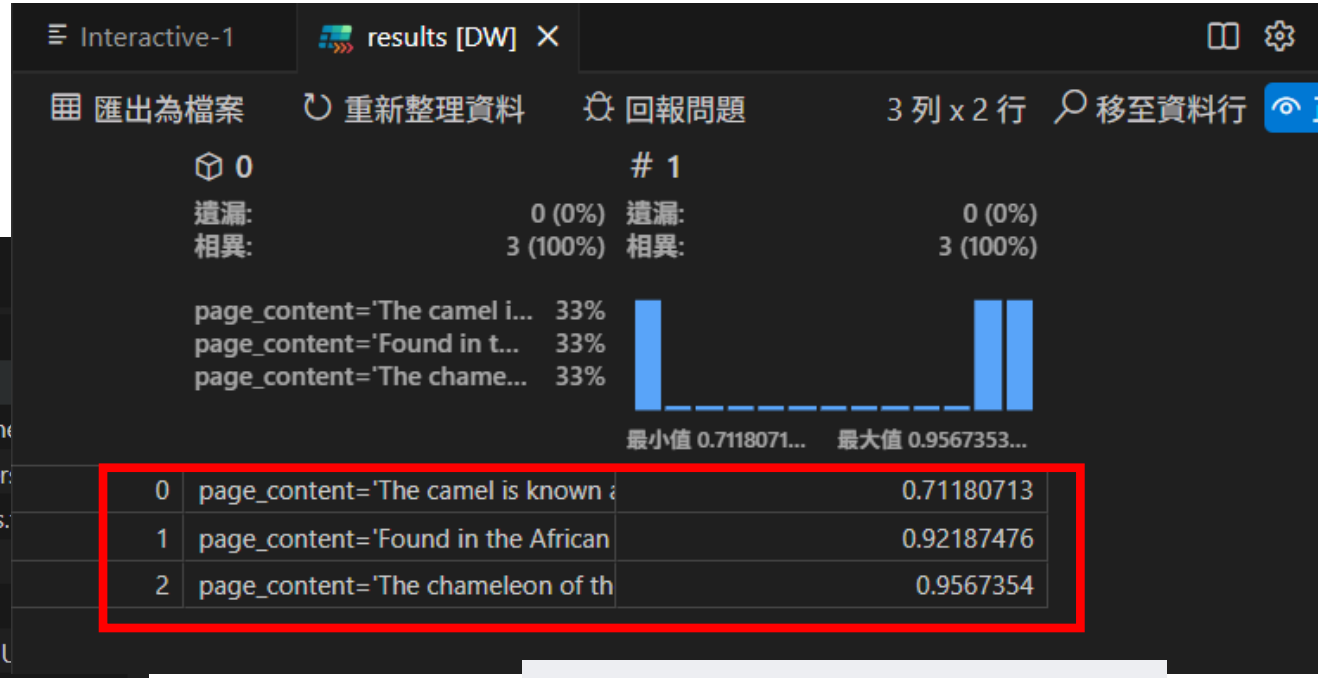
```
【排名 K = 1】
相似度分數 : 0.7118
動物名稱: camel
資料來源: https://www.animalfactsencyclopedia.com/Camel-facts.html
【排名 K = 2】
相似度分數 : 0.9219
動物名稱: fennec fox
資料來源: https://a-z-animals.com/animals/fennec-fox/
【排名 K = 3】
相似度分數 : 0.9567
動物名稱: summer flounder
資料來源: https://a-z-animals.com/animals/fluke-fish/
```

排名(k值越小表示越接近正確答案)



1.進入終端機中的JUPYTER

2.選 results(圖示)



進入後會顯示詳細的K值

動物趣味事實問答（MTEB）

Rank (Bor…	Model	Zero-shot	Memory Us…	Number of P…	Embedding D…	Max Tokens	Mean (T…	Mean (TaskT…	Bitext …	Classification
1	KaLM-Embedding-Gemma3-12B-2511	73%	44884	11.8	3840	32768	**72.32**	**62.51**	**83.76**	**77.88**
2	llama-embed-nemotron-8b	99%	28629	7.5	4096	32768	69.46	61.09	81.72	73.21
3	Qwen3-Embedding-8B	99%	14433	7.6	4096	32768	70.58	61.69	80.89	74.00
4	gemini-embedding-001	99%			3072	2048	68.37	59.59	79.28	71.82
5	Qwen3-Embedding-4B	99%	7671	4.0	2560	32768	69.45	60.86	79.36	72.33
6	Octen-Embedding-8B	99%	14433	7.6	4096	32768	67.85	60.28	80.35	66.68
7	Seed1.6-embedding-1215	89%			2048	32768	70.26	61.34	78.68	76.75
8	Qwen3-Embedding-0.6B	99%	1136	0.596	1024	32768	64.34	56.01	72.23	66.83
9	gte-Qwen2-7B-instruct	⚠ NA	29040	7.6	3584	32768	62.51	55.93	73.92	61.55
10	Linq-Embed-Mistral	99%	13563	7.1	4096	32768	61.47	54.14	70.34	62.24
11	multilingual-e5-large-instruct	99%	1068	0.560	1024	514	63.22	55.08	80.13	64.94

https://huggingface.co/spaces/mteb/leaderboard

PDF 問答

主講：陳昱丞

https://github.com/BONE-22/RAG..io/tree/main/pdf

國立聯合大學
NATIONAL UNITED UNIVERSITY

節能診斷服務中心
AI培訓課程

PDF 問答（app_pdf.py）

https://github.com/BONE-22/RAG..io/blob/main/pdf/app_pdf.py

```
from langchain_community.document_loaders import PyPDFLoader #讀取檔案(pdf)
from langchain_community.vectorstores import FAISS #向量資料庫
from langchain_huggingface import HuggingFaceEmbeddings #嵌入模型
from langchain_text_splitters import RecursiveCharacterTextSplitter #切分文字
from langchain_google_genai import ChatGoogleGenerativeAI #Google提供Embedding模型
from langchain_core.prompts import ChatPromptTemplate #提示詞模板
from langchain_core.output_parsers import StrOutputParser #輸出解析器
from configparser import ConfigParser #讀取與管理設定檔
# 1. 讀取配置
config = ConfigParser()
config.read("config.ini")

# 2. 載入 PDF
loader = PyPDFLoader("貿特198診斷報告final.pdf") # 替換為你的檔案名
data = loader.load()

# 3. 切分文本
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=400,
    chunk_overlap=50,
    separators=["\n\n", "\n", " ", ""]
)
docs = text_splitter.split_documents(data)
# 4. 向量化
embeddings = HuggingFaceEmbeddings(model_name="paraphrase-multilingual-MiniLM-L12-v2")
db = FAISS.from_documents(docs, embeddings)
```

PDF 問答（app_pdf.py）

https://github.com/BONE-22/RAG..io/blob/main/pdf/app_pdf.py

```
# 5. 設定 LLM
llm_gemini = ChatGoogleGenerativeAI(
    model="gemini-2.5-flash",
    api_key=config["Gemini"]["API_KEY"]
)
# 6. 優化 Prompt
prompt = ChatPromptTemplate.from_template(
    """
    你是一個專業的導讀助手。請根據以下提供的上下文資訊來回答問題。
    如果資訊不足以回答，請誠實告知。
    回答要求：
    1. 使用繁體中文。
    2. 除了回覆答案外，請列出比較的清單（條列式，不要表格）。
    上下文：
    {context}
    問題：{input}
    """
)
# 7. 執行檢索與生成
query = "空壓系統做怎麼樣的改善"
docs_related = db.similarity_search(query, k=4)
# 將檢索到的 Document 物件轉換為純文字
context_combined = "\n---\n".join([doc.page_content for doc in docs_related])
chain = prompt | llm_gemini | StrOutputParser()
result = chain.invoke({
    "input": query,
    "context": context_combined
})
print("Question:", query)
print("LLM Answer:\n", result)
```

word 問答
ollama / API 建資料庫

主講：陳昱丞

https://github.com/BONE-22/RAG..io/tree/main/word

國立聯合大學
NATIONAL UNITED UNIVERSITY

節能診斷服務中心
AI培訓課程

word 問答（建向量資料庫 app_word_ingest_API.py）

```python
from configparser import ConfigParser #讀取與管理設定檔
from langchain_community.document_loaders import Docx2txtLoader #讀取檔案(word)
from langchain_text_splitters import CharacterTextSplitter #切分文字
from langchain_google_genai import GoogleGenerativeAIEmbeddings #Google提供Embedding模型
from langchain_community.vectorstores import Chroma #向量資料庫

# --- 1. 設定區 ---
FILE_PATH = "貿特198診斷報告Final.docx"
EMBED_MODEL = "models/text-embedding-004"
PERSIST_DIRECTORY = "./chroma_db_medical"

# --- 2. 載入金鑰 ---
config = ConfigParser()
config.read("config.ini")
api_key = config["Gemini"]["API_KEY"]

# --- 3. 處理文件 ---
print(f"正在載入檔案: {FILE_PATH}...")
loader = Docx2txtLoader(FILE_PATH)
data = loader.load()

# 切割文字
text_splitter = CharacterTextSplitter(
    chunk_size=800,
    chunk_overlap=100,
    separator="\n"
)
docs = text_splitter.split_documents(data)
```

https://github.com/BONE-22/RAG..io/blob/main/word/app_word_ingest_API.py

word 問答（建向量資料庫 app_word_ingest_API.py）

```
# --- 4. 建立 Embedding 與儲存資料庫 ---
embeddings = GoogleGenerativeAIEmbeddings(
    model=EMBED_MODEL,
    google_api_key=api_key,
)

print(f"正在建立資料庫並儲存至 {PERSIST_DIRECTORY}...")

# 建立 Chroma 資料庫並持久化儲存
db = Chroma.from_documents(
    documents=docs,
    embedding=embeddings,
    persist_directory=PERSIST_DIRECTORY
)

print("--- 資料庫建立完成！ ---")
```

word 問答（回答問題 app_word_quert_API.py）

```
from configparser import ConfigParser  #讀取與管理設定檔
from langchain_google_genai import GoogleGenerativeAIEmbeddings, ChatGoogleGenerativeAI  #Google提供Embedding模型
from langchain_chroma import Chroma  #向量資料庫
from langchain_core.prompts import ChatPromptTemplate  #提示詞模板
from langchain_core.output_parsers import StrOutputParser  #輸出的訊息對象轉換成純字串

# --- 1. 設定區 ---
EMBED_MODEL = "models/text-embedding-004"
PERSIST_DIRECTORY = "./chroma_db_medical"
k_value=5

# --- 2. 載入金鑰 ---
config = ConfigParser()
config.read("config.ini")
api_key = config["Gemini"]["API_KEY"]

# --- 3. 讀取現有資料庫 ---
embeddings = GoogleGenerativeAIEmbeddings(
    model=EMBED_MODEL,
    google_api_key=api_key,
)

# 直接載入EMBEDDING 資料庫
db = Chroma(
    persist_directory=PERSIST_DIRECTORY,
    embedding_function=embeddings
)
```

https://github.com/BONE-22/RAG..io/blob/main/word/app_word_quert_API.py

word 問答（回答問題 app_word_quert_API.py）

```
# --- 4. 設定問答鏈---
llm = ChatGoogleGenerativeAI(
    model="gemini-2.5-flash",
    google_api_key=api_key
)
prompt = ChatPromptTemplate.from_template(
    """
    請根據提供的診斷報告內容回答問題。
    如果報告中找不到答案，請回答「根據提供的報告內容，無法找到相關資訊」。
    <報告內容>
    {context}
    </報告內容>
    問題：{input}
    """
    )
chain = prompt | llm | StrOutputParser()
# --- 5. 進行提問 ---
query = "這份報告寫作者是誰" #
# 檢索最相關的片段
docs = db.similarity_search(query, k=k_value)
# --- 6. 除錯資訊：印出檢索到的內容 ---
print(f"資料庫檢索回傳了 {len(docs)} 個片段")
for i, d in enumerate(docs):
    content_snippet = d.page_content.replace('\n', ' ')[:100] # 取前100字方便查看
    print(f"片段 {i+1}: {content_snippet}...")
# 執行 LLM 回答
result = chain.invoke({"input": query, "context": docs})
print(f"問：{query}")
print(f"答：\n{result}")
```

word 問答（地端建向量資料庫 app_word_ingest_ollama.py）

```
import os
import sys
import time
import torch
import warnings
from tqdm import tqdm
from langchain_community.document_loaders import Docx2txtLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_ollama import OllamaEmbeddings
from langchain_community.vectorstores import Chroma
warnings.filterwarnings("ignore", category=DeprecationWarning)
# --- 1. 設定區 ---
FORCE_DEVICE = "cuda"  # "cpu" 或 "cuda"
FILE_PATH = "貿特198診斷報告Final.docx"
EMBED_MODEL = "nomic-embed-text"
PERSIST_DIRECTORY = "./dbollama_“

if not os.path.exists(FILE_PATH):
    print(f"❌ 找不到檔案: {FILE_PATH}")
    sys.exit()

# --- 2. 硬體判定 ---
if FORCE_DEVICE == "cuda" and torch.cuda.is_available():
    current_device = "GPU (CUDA)"
elif FORCE_DEVICE == "cpu":
    current_device = "CPU"
else:
    current_device = "GPU (CUDA)" if torch.cuda.is_available() else "CPU"
print(f"目前適用裝置: {current_device}")
```

https://github.com/BONE-22/RAG..io/blob/main/word/app_word_ingest_ollama.py

word 問答（地端建向量資料庫 app_word_ingest_ollama.py）

```
# --- 3. 載入與切分文件 ---
loader = Docx2txtLoader(FILE_PATH)
raw_docs = loader.load()
text_splitter = RecursiveCharacterTextSplitter(chunk_size=800, chunk_overlap=100)
all_splits = text_splitter.split_documents(raw_docs)
total_chunks = len(all_splits)
# --- 4. 初始化地端 Embedding 模型 ---
embeddings = OllamaEmbeddings(model=EMBED_MODEL)
print(f"\n正在進行地端向量化 (總計 {total_chunks} 個段落)...")
start_time = time.time()
try:
    # 建立空的資料庫
    vectorstore = Chroma.from_documents(
        documents=[all_splits[0]],
        embedding=embeddings,
        persist_directory=PERSIST_DIRECTORY
    )
    # 分批處理並顯示進度條
    batch_size = 5
    for i in tqdm(range(1, total_chunks, batch_size), desc=f"地端 {current_device} 處理中"):
        batch = all_splits[i : i + batch_size]
        vectorstore.add_documents(batch)
    duration = time.time() - start_time
    print(f"✅ 地端向量資料庫建置完成！")
    print(f"⏱️ 總耗時: {duration:.2f} 秒")
except Exception as e:
    print(f"❌ 發生錯誤: {e}")
    print("提示：請檢查 Ollama 是否已啟動，且已執行過 'ollama pull nomic-embed-text'")
```

word 問答（地端建向量資料庫 app_word_ingest_ollama.py）

```
# --- 3. 載入與切分文件 ---
loader = Docx2txtLoader(FILE_PATH)
raw_docs = loader.load()
text_splitter = RecursiveCharacterTextSplitter(chunk_size=800, chunk_overlap=100)
all_splits = text_splitter.split_documents(raw_docs)
total_chunks = len(all_splits)
# --- 4. 初始化地端 Embedding 模型 ---
embeddings = OllamaEmbeddings(model=EMBED_MODEL)
print(f"\n正在進行地端向量化 (總計 {total_chunks} 個段落)...")
start_time = time.time()
try:
    # 建立空的資料庫
    vectorstore = Chroma.from_documents(
        documents=[all_splits[0]],
        embedding=embeddings,
        persist_directory=PERSIST_DIRECTORY
    )
    # 分批處理並顯示進度條
    batch_size = 5
    for i in tqdm(range(1, total_chunks, batch_size), desc=f"地端 {current_device} 處理中"):
        batch = all_splits[i : i + batch_size]
        vectorstore.add_documents(batch)
    duration = time.time() - start_time
    print(f"✅ 地端向量資料庫建置完成！")
    print(f"⏱️ 總耗時: {duration:.2f} 秒")
except Exception as e:
    print(f"❌ 發生錯誤: {e}")
    print("提示：請檢查 Ollama 是否已啟動，且已執行過 'ollama pull nomic-embed-text'")
```

word 問答（回答問題 app_word_ quert _ollama-1.py）

沒有 LLM 的

```
import os
import torch #GPU 加速
from langchain_chroma import Chroma # 向量資料庫
from langchain_ollama import OllamaEmbeddings  # 向量化

# --- 1. 設定區 ---
FORCE_DEVICE = "cuda"  # 預期使用的裝置
EMBED_MODEL = "nomic-embed-text"
PERSIST_DIRECTORY = "./db_ollama"
k_value=5

# --- 2. 硬體判定與資訊獲取 ---
cuda_available = torch.cuda.is_available()

if FORCE_DEVICE == "cuda" and cuda_available:
    device_info = f"GPU (CUDA)"
elif FORCE_DEVICE == "cuda" and not cuda_available:
    device_info = "CPU (雖然設定為 cuda，但系統未偵測到可用 GPU，已切換至 CPU)"
else:
    device_info = "CPU“

# --- 3. 初始化 Embedding 模型 ---
embeddings = OllamaEmbeddings(model=EMBED_MODEL)

if not os.path.exists(PERSIST_DIRECTORY):
    print(f"❌ 找不到資料庫目錄: {PERSIST_DIRECTORY}，請先執行建庫程式。")
    exit()
```

https://github.com/BONE-22/RAG..io/blob/main/word/app_word_%20quert%20_ollama-1.py

word 問答（回答問題 app_word_ quert _ollama-1.py）

```
# --- 4. 載入本地存好的 Chroma 資料 ---
db = Chroma(
    persist_directory=PERSIST_DIRECTORY,
    embedding_function=embeddings
)

# --- 5. 測試檢索 ---
query = "報告撰寫人是誰？"
# 檢索最相關的片段
docs = db.similarity_search(query, k=k_value)

# --- 6. 輸出結果 ---
print(f"【系統硬體資訊】: {device_info}")
print(f"【使用模型名稱】: {EMBED_MODEL}")
print(f"【資料庫路徑】: {PERSIST_DIRECTORY}")
print("-" * 30)
print(f"問題：{query}\n")
print("--- 檢索到的報告內容如下 ---")

for i, doc in enumerate(docs):
    clean_content = " ".join(doc.page_content.split())
    source = doc.metadata.get("source", "未知來源")
    print(f"[{i+1}] (來源: {source}) {clean_content}\n")
```

word 問答（回答問題 app_word_ quert _ollama-2.py）

有 LLM 的

```
import os
from langchain_ollama import OllamaEmbeddings, ChatOllama # 理解與尋找、回答與總結
from langchain_chroma import Chroma # 向量資料庫
from langchain_core.prompts import ChatPromptTemplate # 提示詞模板
from langchain_core.output_parsers import StrOutputParser # 輸出解析器

# --- 1. 設定區 ---
EMBED_MODEL = "nomic-embed-text"
LLM_MODEL = "llama3.2:1b"
PERSIST_DIRECTORY = "./db_ollama"
k_value=5

# --- 2. 檢查資料庫是否存在 ---
if not os.path.exists(PERSIST_DIRECTORY):
    print(f"❌ 找不到資料庫目錄: {PERSIST_DIRECTORY}，請先執行建庫程式。")
    exit()

# --- 3. 初始化地端模型 ---
print(f"正在載入地端模型 ({LLM_MODEL})...")

# Embedding 模型 (用來把問題轉成向量去搜尋)
embeddings = OllamaEmbeddings(model=EMBED_MODEL)

# LLM 模型 (用來閱讀搜尋結果並回答)
llm = ChatOllama(
    model=LLM_MODEL,
    temperature=0,
    num_gpu=1 # 設為 1 通常代表啟用 GPU 加速
)
```

https://github.com/BONE-22/RAG..io/blob/main/word/app_word_%20quert%20_ollama-2.py

word 問答（回答問題 app_word_ quert _ollama-2.py）

```
# 載入現有資料庫
db = Chroma(
    persist_directory=PERSIST_DIRECTORY,
    embedding_function=embeddings
)
# --- 4. 設定 Prompt (針對地端模型優化) ---
prompt = ChatPromptTemplate.from_template("""
你是一個專業的醫療報告分析助手。請根據下方提供的報告內容來回答問題。
答案請使用「繁體中文」回答。如果你無法從內容中找到答案，請回答不知道，不要胡編亂造。
<報告內容>
{context}
</報告內容>
問題：{input}
""")
# 建立執行鏈
chain = prompt | llm | StrOutputParser()
# --- 5. 執行提問 ---
query = "這份診斷報告的報告撰寫人是誰？"
print(f"\n🔍 正在檢索資料並生成回答...")
# 檢索最相關的 3 個片段
docs = db.similarity_search(query, k=k_value)
# 執行
try:
    result = chain.invoke({
        "input": query,
        "context": docs
    })
    print("\n" + "="*30)
    print(f"問：{query}")
    print(f"答：\n{result}")
    print("="*30)
except Exception as e:
    print(f"❌ 執行失敗: {e}")
    print("提示：請確保 Ollama 伺服器正在執行中。")
```