

gemini-conversation

Exported on: 11/26/2025, 11:52:36 AM

User

use streamlit to build an app

Gemini

這是一個非常好的想法！將這個 Jupyter Notebook 轉換為 Streamlit 應用程式，可以讓沒有程式背景的使用者也能輕鬆上傳檔案並建立自己的向量資料庫 (Vector Database)。

以下是完整的 Streamlit 應用程式程式碼。這個應用程式保留了原本 Notebook 的邏輯（使用 `intfloat/multilingual-e5-small` 模型和 FAISS），並加入了圖形化介面來處理檔案上傳與下載。

1. 安裝必要套件

首先，你需要建立一個 `requirements.txt` 或是直接在終端機安裝以下套件：

Bash



```
pip install streamlit langchain langchain-community pypdf python-docx sentence-transformers faiss-cpu
```

2. Streamlit 應用程式代碼 (app.py)

請將以下程式碼儲存為 `app.py`：

Python



```
import streamlit as st
import os
import shutil
import tempfile

from langchain_community.document_loaders import TextLoader, PyPDFLoader, UnstructuredWordDocumentLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS

# --- 設定頁面 ---
st.set_page_config(page_title="RAG 向量資料庫產生器", page_icon="📁")

st.title("📁 RAG 向量資料庫產生器")
st.markdown("""
這個應用程式可以協助你將 PDF, DOCX, TXT 文件轉換為 **FAISS 向量資料庫**。
底層使用 `intfloat/multilingual-e5-small` 模型進行 Embedding。
""")

# --- 類別定義（保留原 Notebook 邏輯） ---
# 自訂支援 E5 的 embedding 模型（加上 "passage:" / "query:" 前綴）
class CustomE5Embedding(HuggingFaceEmbeddings):
    def embed_documents(self, texts):
        texts = [f"passage: {t}" for t in texts]
```

```

        return super().embed_documents(texts)

    def embed_query(self, text):
        return super().embed_query(f"query: {text}")

# --- 快取模型載入 (提升效能) ---
@st.cache_resource
def load_embedding_model():
    return CustomE5Embedding(model_name="intfloat/multilingual-e5-small")

# --- 側邊欄設定 ---
with st.sidebar:
    st.header("⚙️ 參數設定")
    chunk_size = st.number_input("Chunk Size (切分大小)", min_value=100, max_value=2000, value=500,
    chunk_overlap = st.number_input("Chunk Overlap (重疊大小)", min_value=0, max_value=500, value=10

# --- 主要邏輯 ---

# 1. 檔案上傳
uploaded_files = st.file_uploader(
    "請上傳您的文件 (.txt, .pdf, .docx)",
    type=["txt", "pdf", "docx"],
    accept_multiple_files=True
)

if uploaded_files and st.button("🚀 開始建立資料庫"):

    embedding_model = load_embedding_model()
    documents = []

    # 建立臨時資料夾來處理上傳的檔案
    with tempfile.TemporaryDirectory() as temp_dir:

        progress_bar = st.progress(0, text="正在處理檔案...")

        # 2. 儲存並載入文件
        for i, uploaded_file in enumerate(uploaded_files):
            file_path = os.path.join(temp_dir, uploaded_file.name)

            # 將上傳的 Bytes 寫入臨時檔案
            with open(file_path, "wb") as f:
                f.write(uploaded_file.getbuffer())

            # 根據副檔名選擇 Loader
            try:
                if uploaded_file.name.endswith(".txt"):

```

```

        loader = TextLoader(file_path, encoding="utf-8")
    elif uploaded_file.name.endswith(".pdf"):
        loader = PyPDFLoader(file_path)
    elif uploaded_file.name.endswith(".docx"):
        loader = UnstructuredWordDocumentLoader(file_path)
    else:
        continue

    documents.extend(loader.load())
except Exception as e:
    st.error(f"讀取檔案 {uploaded_file.name} 時發生錯誤: {e}")

# 更新進度條
progress = (i + 1) / len(uploaded_files) * 0.3 # 載入佔 30% 進度
progress_bar.progress(progress, text=f"已載入: {uploaded_file.name}")

if not documents:
    st.error("沒有成功載入任何文件，請檢查檔案內容。")
    st.stop()

# 3. 切分文件
progress_bar.progress(0.4, text="正在切分文本...")
splitter = RecursiveCharacterTextSplitter(chunk_size=chunk_size, chunk_overlap=chunk_overlap)
split_docs = splitter.split_documents(documents)

st.info(f"共載入 {len(documents)} 份文件，切分為 {len(split_docs)} 個片段。")

# 4. 建立向量資料庫 (最耗時)
progress_bar.progress(0.5, text="正在進行 Embedding 與建立向量庫 (這可能需要一點時間)...")
vectorstore = FAISS.from_documents(split_docs, embedding_model)

# 5. 儲存與壓縮
progress_bar.progress(0.9, text="正在壓縮檔案...")

# 定義輸出的資料夾名稱
output_folder = "faiss_db_output"
if os.path.exists(output_folder):
    shutil.rmtree(output_folder) # 清除舊的

vectorstore.save_local(output_folder)

# 壓縮資料夾
zip_path = shutil.make_archive("faiss_db", 'zip', output_folder)

progress_bar.progress(1.0, text="完成!")

```

```
# 6. 提供下載按鈕
with open(zip_path, "rb") as fp:
    btn = st.download_button(
        label="📄 下載 faiss_db.zip",
        data=fp,
        file_name="faiss_db.zip",
        mime="application/zip"
    )

st.success("向量資料庫建立完成！請點擊上方按鈕下載。")

# 清理生成的 zip 檔（選擇性，或保留給下次）
# os.remove(zip_path)
# shutil.rmtree(output_folder)
```

如何執行

1. 開啟終端機 (Terminal) 或命令提示字元。
2. 進入存放 `app.py` 的資料夾。
3. 執行以下指令：

Bash

```
streamlit run app.py
```

主要改動說明 (從 Notebook 到 App)

1. 檔案處理 (`tempfile`)：

- Notebook 直接讀取硬碟路徑。
- Streamlit 的 `file_uploader` 是將檔案存在記憶體 (RAM) 中。為了配合 LangChain 的 Loaders (通常需要檔案路徑)，我使用了 Python 的 `tempfile` 模組，先將檔案暫存到臨時目錄，讀取完後會自動清除，確保不會留下垃圾檔案。

2. 使用者介面 (UI)：

- 加入了 `st.file_uploader` 讓使用者直覺地上傳多個檔案。
- 加入了 `st.sidebar` 讓使用者可以調整 `chunk_size` 和 `chunk_overlap`，增加了靈活性。
- 加入了 `st.progress` 進度條，讓使用者知道目前處理到哪一個階段（因為 Embedding 通常比較慢）。

3. 下載功能：

- Notebook 使用 Linux 指令 `!zip`。
- App 改用 Python 內建的 `shutil.make_archive` 來進行跨平台的壓縮，並使用 `st.download_button` 讓瀏覽器觸發下載。

4. 模型快取 (`@st.cache_resource`)：

- 為了避免每次按下按鈕都重新載入 Embedding 模型 (很花時間)，使用了 Streamlit 的快取功能，讓模型只載入一次。

User

```
ner.py:640640 in in code_to_exec code_to_exec
```

```
C:\Users\ss348\Desktop\AloThw4\main.py :C:\Users\ss348\Desktop\AloThw4\main.py:6: in 6<module> in <module>
```

```
3 3 importimport shutilshutil
```

```
4 4 importimport tempfiletempfile
```

```
5 5 fromfrom langchain_communitylangchain_community.document_loadersdocument_loaders importimport  
TextLoader, PyPDFLoader, Unst TextLoader, PyPDFLoader, Unst
```

```
6 6 fromfrom langchainlangchain.text_splittertext_splitter importimport RecursiveCharacterTextSplitter  
RecursiveCharacterTextSplitter
```

```
7 from 7 fromlangchain .langchainembeddings. importembeddings HuggingFaceEmbeddings import  
HuggingFaceEmbeddings 8 from 8 langchain_communityfrom. langchain_communityvectorstores .import FAISS  
vectorstores import FAISS 9 9
```

```
ModuleNotFoundError:
```

```
-----No module named 'langchain.text_splitter' ModuleNotFoundError: No module named  
'langchain.text_splitter'
```

Gemini

這個錯誤 (`ModuleNotFoundError: No module named 'langchain.text_splitter'`) 是因為 **LangChain** 最近進行了重大更新，將許多模組拆分成了獨立的套件。

要解決這個問題，你需要做兩個步驟：**安裝缺少的套件** 並 **修改程式碼的 `import` 路徑**。

第一步：安裝缺少的套件

請在你的終端機 (Terminal) 或命令提示字元 (CMD) 中執行以下指令：

```
Bash
```



```
pip install langchain-text-splitters
```

第二步：修改 `main.py` 的程式碼

請打開你的 `main.py` (位於 `C:\Users\ss348\Desktop\AloThw4\main.py`)，找到第 6 行到第 8 行，將舊的寫法改為新的寫法。

請將這一段 (舊的)：

```
Python
```



```
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
```

修改為這一段（新的）：

Python



```
# 修改後的 import
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
```

為什麼會這樣？

LangChain 為了讓套件更輕量化，現在建議從專門的子套件引入功能：

1. `RecursiveCharacterTextSplitter` 現在位於 `langchain_text_splitters` 。
2. `HuggingFaceEmbeddings` 建議從 `langchain_community.embeddings` 引入（原本的 `langchain.embeddings` 也即將被廢棄）。

修改完並存檔後，再次執行 `streamlit run main.py` 應該就能成功了！

User

proceed to build a rag system. And generate some sample documents. allow users to input things.

Gemini

這是一個完整的 RAG 系統解決方案。為了讓你能夠馬上測試，我整合了三個功能到同一個 Streamlit App 中：

1. **生成範例文件**：一鍵產生假的社團資料與規章，讓你不用找檔案。
2. **建立資料庫**：修復了之前的 Import 錯誤，並整合建立流程。
3. **對話系統 (RAG)**：讓你可以輸入問題，系統會從資料庫中找出答案。

1. 更新安裝清單

為了支援對話功能，我們需要更新一下套件（請在終端機執行）：

Bash

```
pip install streamlit langchain-community langchain-text-splitters faiss-cpu sentence-transformers
```

(註：如果你有安裝 Ollama 並想用本地 LLM，請額外安裝 `langchain-ollama`，下面程式碼我預設使用「純檢索」模式，確保你沒有 GPU 也能跑得動。)

2. 完整的 RAG 應用程式 (main.py)

請將原本的 `main.py` 全部覆蓋為以下程式碼。這份程式碼使用了新版的 **Import** 路徑，不會再報錯。

Python

```
import streamlit as st
import os
import shutil
import tempfile

from langchain_community.document_loaders import TextLoader, PyPDFLoader, UnstructuredWordDocumentLoader
# --- 修正後的 Import (解決 ModuleNotFoundError) ---
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS

# 設定頁面
st.set_page_config(page_title="本地 RAG 系統演示", layout="wide")
st.title("🚗 本地 RAG 系統 (Retrieval-Augmented Generation)")

# --- 初始化 Session State ---
if "messages" not in st.session_state:
    st.session_state.messages = []

# --- 核心設定 ---
DB_PATH = "faiss_db_output"
```

```

SAMPLE_DIR = "sample_docs"

# 載入 Embedding 模型 (快取以提升速度)
@st.cache_resource
def load_embedding_model():
    # 使用輕量級的中文/多語言模型
    return HuggingFaceEmbeddings(model_name="intfloat/multilingual-e5-small")

# --- 功能函式 ---

def generate_sample_data():
    """生成範例文件"""
    if not os.path.exists(SAMPLE_DIR):
        os.makedirs(SAMPLE_DIR)

    samples = {
        "AI研究社_介紹.txt": "AI研究社成立於2023年，社長是王小明。社課時間為每週五晚上7點，地點在資訊大樓306室。",
        "登山社_活動規章.txt": "登山社安全規章：1. 參加百岳行程需具備基礎體能證明。 2. 裝備檢查未通過者不得參加。",
        "圖書館_借閱規則.txt": "圖書館開放時間為週一至週五 08:00-22:00。大學部學生可借閱10本書，借期30天。",
    }

    created_files = []
    for filename, content in samples.items():
        path = os.path.join(SAMPLE_DIR, filename)
        with open(path, "w", encoding="utf-8") as f:
            f.write(content)
        created_files.append(filename)

    return created_files

def process_documents(files):
    """處理上傳的或生成的檔案並建立向量庫"""
    embedding_model = load_embedding_model()
    documents = []

    with tempfile.TemporaryDirectory() as temp_dir:
        for file in files:
            # 判斷是 UploadedFile 物件 (上傳的) 還是字串 (生成的路徑)
            if hasattr(file, "name"):
                # 處理使用者上傳的檔案
                file_path = os.path.join(temp_dir, file.name)
                with open(file_path, "wb") as f:
                    f.write(file.getbuffer())
                file_name = file.name
            else:
                # 處理生成的範例檔案 (傳入的是路徑字串)

```

```

        file_name = os.path.basename(file)
        file_path = file # 已經是路徑了

    try:
        if file_name.endswith(".txt"):
            loader = TextLoader(file_path, encoding="utf-8")
        elif file_name.endswith(".pdf"):
            loader = PyPDFLoader(file_path)
        else:
            continue
        documents.extend(loader.load())
    except Exception as e:
        st.error(f"讀取 {file_name} 失敗: {e}")

    if not documents:
        return False, "沒有有效的文件被讀取"

    # 切分文本
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=300, chunk_overlap=50)
    split_docs = text_splitter.split_documents(documents)

    # 建立資料庫
    vectorstore = FAISS.from_documents(split_docs, embedding_model)
    vectorstore.save_local(DB_PATH)
    return True, f"成功建立資料庫！共 {len(split_docs)} 個片段。"

def query_rag(query_text):
    """查詢向量資料庫"""
    if not os.path.exists(DB_PATH):
        return "⚠️ 請先建立資料庫（請至左側 '建立知識庫' 分頁）"

    embedding_model = load_embedding_model()
    # 載入現有的資料庫
    vectorstore = FAISS.load_local(DB_PATH, embedding_model, allow_dangerous_deserialization=True)

    # 搜尋最相似的 3 個片段
    results = vectorstore.similarity_search(query_text, k=3)

    # 組合回應（因為沒有 LLM，我們直接整理檢索到的結果）
    response = "根據資料庫檢索，我找到了以下相關資訊：\n\n"
    for i, doc in enumerate(results):
        response += f"**來源 {i+1}**：{doc.page_content}\n\n---\n"

    return response

# --- 介面佈局 ---

```

```

tab1, tab2 = st.tabs(["📁 建立知識庫", "💬 AI 助手對話"])

with tab1:
    st.header("步驟 1：準備資料")

    col1, col2 = st.columns(2)

    with col1:
        st.subheader("選項 A：生成範例資料")
        if st.button("🚀 生成測試用文件"):
            files = generate_sample_data()
            st.success(f"已生成 {len(files)} 份文件於 {SAMPLE_DIR} 資料夾。")
            # 自動列出生成的檔案路徑供後續處理
            st.session_state['generated_files'] = [os.path.join(SAMPLE_DIR, f) for f in files]

    with col2:
        st.subheader("選項 B：上傳自己的檔案")
        uploaded_files = st.file_uploader("上傳 .txt 或 .pdf", accept_multiple_files=True)

    st.divider()

    st.header("步驟 2：建立向量索引")
    if st.button("🚀 開始建立 RAG 資料庫"):
        files_to_process = []

        # 加入上傳的檔案
        if uploaded_files:
            files_to_process.extend(uploaded_files)

        # 加入生成的檔案（如果有的話）
        if 'generated_files' in st.session_state:
            files_to_process.extend(st.session_state['generated_files'])

        if not files_to_process:
            st.warning("請先上傳檔案或生成範例資料！")
        else:
            with st.spinner("正在分析文件並建立向量索引..."):
                success, msg = process_documents(files_to_process)
                if success:
                    st.success(msg)
                else:
                    st.error(msg)

with tab2:
    st.header("與你的文件對話")

```

```

st.caption("輸入問題，系統會從剛剛建立的資料庫中找答案。")

# 顯示歷史訊息
for message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.markdown(message["content"])

# 處理使用者輸入
if prompt := st.chat_input("請輸入問題 (例如：AI社社長是誰?)"):
    # 1. 顯示使用者訊息
    st.session_state.messages.append({"role": "user", "content": prompt})
    with st.chat_message("user"):
        st.markdown(prompt)

    # 2. 呼叫 RAG 邏輯
    with st.chat_message("assistant"):
        with st.spinner("正在搜尋資料庫..."):
            response = query_rag(prompt)
            st.markdown(response)

    # 3. 儲存 AI 回應
    st.session_state.messages.append({"role": "assistant", "content": response})

```

如何使用這個 Demo

1. 執行程式：在終端機輸入：`streamlit run main.py`
2. 生成資料 (Tab 1)：
 - 進入 "📁 建立知識庫" 分頁。
 - 點擊 "🔥 生成測試用文件"。這會在你的資料夾中自動產生包含社團介紹、圖書館規則的假文件。
 - 接著點擊下方的大按鈕 "🚀 開始建立 RAG 資料庫"。看到綠色成功訊息表示資料庫建好了。
3. 開始對話 (Tab 2)：
 - 切換到 "💬 AI 助手對話" 分頁。
 - 在下方輸入框試著問：
 - "AI社的社長是誰？"
 - "圖書館逾期怎麼罰款？"
 - "登山社有什麼安全規章？"
 - 系統會找出最相關的文字片段並顯示出來。

關於「真正的」RAG (加上 LLM)

目前的程式碼是 **Retrieval (檢索)** 部分。為了避免你在設定 LLM (如 OpenAI API Key 或 Ollama) 時遇到困難，我先讓程式回傳「檢索到的原文」。

如果你希望它像 ChatGPT 一樣用流暢的語句回答，而不只是複製貼上原文，你需要串接 LLM。如果你已經安裝了 [Ollama](#)，可以告訴我，我可以給你一行程式碼來啟用它。

User

[illegible]

During handling of the above exception, another exception occurred:

[illegible]

Gemini

這兩個問題的解決方案如下。

問題 1：解決 Streamlit 與 Torch 的衝突 (RuntimeError)

這個錯誤 (Examining the path of torch.classes raised...) 是 Streamlit 的「檔案監控功能 (File Watcher)」與 PyTorch 的內部機制發生衝突所導致的，特別常見於 Windows 環境。

解決方法： 你需要建立一個 Streamlit 的設定檔來關閉或調整檔案監控。

1. 在你的專案資料夾（也就是 `main.py` 所在的資料夾）中，建立一個新資料夾叫做 `.streamlit`。
2. 在這個資料夾內，建立一個檔案叫做 `config.toml`。
3. 用記事本開啟 `config.toml`，貼上以下內容並存檔：

Ini, TOML


```
[server]  
fileWatcherType = "poll"
```

(註：如果 `poll` 還是會報錯，請改為 `none`，但這樣你修改程式碼後需要手動重整網頁。)

問題 2：讓使用者瀏覽文件 (Browse Docs)

為了讓使用者能「瀏覽」文件，我們需要改變一下邏輯：將上傳的檔案永久存入一個資料夾（而不只是暫存在記憶體中），然後讀取該資料夾來顯示列表。

以下是更新後的 `main.py`。我新增了：

1. **source_data** 資料夾：所有上傳或生成的檔案都會存在這裡，方便瀏覽。
2. 「 檢視目前文件」區塊：在左側欄位新增了一個區域，可以展開查看所有已存入文件的內容。

請將 `main.py` 全部替換為以下程式碼：

Python

```

import streamlit as st
import os
import shutil
import tempfile
# 引入 shutil 以便移動檔案
from langchain_community.document_loaders import TextLoader, PyPDFLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS

# --- 設定頁面 ---
st.set_page_config(page_title="RAG 知識庫系統", layout="wide", page_icon="🤖")
st.title("🤖 本地 RAG 系統 (含文件瀏覽)")

# --- 核心路徑設定 ---
DB_PATH = "faiss_db_output"      # 向量資料庫儲存位置
DOCS_DIR = "source_data"         # 原始文件儲存位置 (新增：為了讓使用者可以瀏覽)

# 確保資料夾存在
os.makedirs(DOCS_DIR, exist_ok=True)

# --- 初始化 Session State ---
if "messages" not in st.session_state:
    st.session_state.messages = []

# --- 載入模型 (快取) ---
@st.cache_resource
def load_embedding_model():
    return HuggingFaceEmbeddings(model_name="intfloat/multilingual-e5-small")

# --- 功能函式 ---

def save_uploaded_file(uploaded_file):
    """將上傳的檔案儲存到 DOCS_DIR"""
    file_path = os.path.join(DOCS_DIR, uploaded_file.name)
    with open(file_path, "wb") as f:
        f.write(uploaded_file.getbuffer())
    return file_path

def generate_sample_data():
    """生成範例文件並存入 DOCS_DIR"""
    samples = {
        "AI研究社_介紹.txt": "AI研究社成立於2023年，社長是王小明。社課時間為每週五晚上7點，地點在資訊大樓301室。",
        "登山社_活動規章.txt": "登山社安全規章：1. 參加百岳行程需具備基礎體能證明。 2. 裝備檢查未通過者不得參加。",
        "圖書館_借閱規則.txt": "圖書館開放時間為週一至週五 08:00-22:00。大學部學生可借閱10本書，借期30天。"
    }

```



```

for filename, content in samples.items():
    path = os.path.join(DOCS_DIR, filename)
    with open(path, "w", encoding="utf-8") as f:
        f.write(content)

return list(samples.keys())

def build_vector_db():
    """讀取 DOCS_DIR 中的所有檔案並建立向量庫"""
    embedding_model = load_embedding_model()
    documents = []

    # 掃描 DOCS_DIR 資料夾
    files = [f for f in os.listdir(DOCS_DIR) if f.endswith(('.txt', '.pdf'))]

    if not files:
        return False, "資料夾中沒有文件，請先上傳或生成資料。"

    progress_bar = st.progress(0, text="正在讀取檔案...")

    for i, file in enumerate(files):
        file_path = os.path.join(DOCS_DIR, file)
        try:
            if file.endswith(".txt"):
                loader = TextLoader(file_path, encoding="utf-8")
            elif file.endswith(".pdf"):
                loader = PyPDFLoader(file_path)
            else:
                continue
            documents.extend(loader.load())
        except Exception as e:
            st.error(f"讀取 {file} 失敗: {e}")

        progress_bar.progress((i + 1) / len(files), text=f"已讀取: {file}")

    if not documents:
        return False, "沒有有效內容可建立索引。"

    # 切分與向量化
    progress_bar.progress(0.8, text="正在切分文本與建立索引...")
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=300, chunk_overlap=50)
    split_docs = text_splitter.split_documents(documents)

    vectorstore = FAISS.from_documents(split_docs, embedding_model)
    vectorstore.save_local(DB_PATH)

```

```

progress_bar.progress(1.0, text="完成!")
return True, f"成功建立資料庫！共包含 {len(files)} 份文件，切分為 {len(split_docs)} 個片段。"

def query_rag(query_text):
    """查詢向量資料庫"""
    if not os.path.exists(DB_PATH):
        return "⚠️ 請先建立資料庫（請至左側 '建立知識庫' 分頁）"

    embedding_model = load_embedding_model()
    vectorstore = FAISS.load_local(DB_PATH, embedding_model, allow_dangerous_deserialization=True)
    results = vectorstore.similarity_search(query_text, k=3)

    response = "根據資料庫檢索，我找到了以下相關資訊：\n\n"
    for i, doc in enumerate(results):
        # 嘗試從 metadata 獲取檔名
        source = doc.metadata.get('source', '未知來源')
        source_name = os.path.basename(source)
        response += f"**來源 {i+1} ({source_name})*: \n{doc.page_content}\n\n---\n"
    return response

# --- 介面佈局 ---

tab1, tab2 = st.tabs(["📁 管理與瀏覽文件", "💬 AI 助手對話"])

with tab1:
    col1, col2 = st.columns([1, 1])

    with col1:
        st.subheader("1. 新增資料")
        st.info("上傳或生成檔案後，檔案會存入 `source_data` 資料夾。")

        # 生成範例按鈕
        if st.button("🌟 生成測試用文件"):
            files = generate_sample_data()
            st.success(f"已生成 {len(files)} 份文件。")
            st.rerun() # 重新整理以更新列表

        # 上傳按鈕
        uploaded_files = st.file_uploader("上傳新文件 (.txt, .pdf)", accept_multiple_files=True)
        if uploaded_files:
            for u_file in uploaded_files:
                save_uploaded_file(u_file)
            st.success(f"已儲存 {len(uploaded_files)} 份新文件。")
            st.rerun()

```

```

st.divider()

st.subheader("2. 建立/更新 資料庫")
if st.button("🚀 重建 RAG 索引"):
    with st.spinner("正在處理..."):
        success, msg = build_vector_db()
        if success:
            st.success(msg)
        else:
            st.error(msg)

with col2:
    st.subheader("📁 檢視目前文件")
    st.caption(f"資料夾路徑: {DOCS_DIR}")

    # 列出 source_data 中的檔案
    existing_files = os.listdir(DOCS_DIR)

    if not existing_files:
        st.write("目前沒有任何文件。")
    else:
        # 使用 Expander 讓使用者可以展開看內容
        for f in existing_files:
            file_path = os.path.join(DOCS_DIR, f)
            with st.expander(f"📄 {f}"):
                # 刪除按鈕
                if st.button("刪除", key=f"del_{f}"):
                    os.remove(file_path)
                    st.rerun()

                # 顯示內容預覽 (僅限 txt, pdf 顯示路徑)
                if f.endswith(".txt"):
                    with open(file_path, "r", encoding="utf-8") as _f:
                        st.text(_f.read())
                else:
                    st.write("PDF 檔案僅支援預覽檔名與路徑。")

with tab2:
    st.header("與你的文件對話")

    # 檢查是否有資料庫
    if not os.path.exists(DB_PATH):
        st.warning("⚠️ 尚未偵測到向量資料庫，請先至「管理與瀏覽文件」分頁建立索引。")

    # 顯示歷史訊息
    for message in st.session_state.messages:

```

```

        with st.chat_message(message["role"]):
            st.markdown(message["content"])

# 處理輸入
if prompt := st.chat_input("請輸入問題..."):
    st.session_state.messages.append({"role": "user", "content": prompt})
    with st.chat_message("user"):
        st.markdown(prompt)

    with st.chat_message("assistant"):
        with st.spinner("搜尋中..."):
            response = query_rag(prompt)
            st.markdown(response)

    st.session_state.messages.append({"role": "assistant", "content": response})

```

這次修改的重點：

1. 瀏覽功能 (col2 區塊)：

- 我加入了一個邏輯來讀取 `source_data` 資料夾。
- 使用 `st.expander` 製作摺疊選單，點開就可以看到 `.txt` 的內容。
- 加入了簡單的刪除按鈕，如果你不想某個文件被搜尋到，刪除後記得按一下「重建 RAG 索引」。

2. 檔案持久化：

- 之前是傳完就刪（使用 `tempfile`），現在會將檔案 `write` 到 `source_data` 資料夾，這樣才能實現「瀏覽」功能。

3. 錯誤修復：

- 記得搭配上上面提到的 `config.toml` 來解決 Torch 報錯的問題。