異國菜單翻譯與美食具象化

Colab:
https://colab.research.google.com/drive/1CKVoWdz1PT0__Tz8r5BOV49uYcStQAsB?usp=sharing

Youtube:

https://youtu.be/ed201nbGWG4 **(為了符合影片時間限制有剪掉總生成等待時間 30 秒)**

一、 介紹你使用的模型

為了的能夠在不付費的基礎上達成，最終將原本規劃的須付費版模型（GPT-4o / DALL-E 3）換成為高效能的免費/開源替代方案：

1. **多模態分析模型 (Vision & Text Analysis)：**
    - **模型名稱：** Google Gemini 2.5 Flash (透過 `google.generativeai` API 存取)。
    - **用途：** 負責精準辨識菜單圖片中的文字，將異國語言翻譯成繁體中文，並結構化輸出為 JSON 格式。此外，它還負責根據菜色內容，撰寫專用的英文繪圖提示詞 (Image Prompt)。
2. **圖像生成模型 (Image Generation)：**
    - **模型名稱：** Pollinations.ai (Turbo Model)。
    - **用途：** 負責根據 prompt 生成圖片的功能。這是一個基於 SDXL Turbo 的高速生成模型。
    - **風格策略：** 雖然底層是通用模型，但我們透過提示詞工程強制注入了 **「Nano Banana 風格」**，使其生成的圖片盡可能具備該模型的視覺效果。

二、 說明你在這次專案想達成的目的

本專案旨在解決旅客在異國餐廳面臨的「雙重困境」：

1. **看不懂文字：** 面對法文、德文等純文字菜單，無法理解菜色內容 。
2. **無法想像菜色：** 即使翻譯出文字，因為文化差異，仍難以憑空想像餐點的實際樣貌（份量、擺盤、食材）。

核心目標： 打造一個「拍菜單 -> 翻譯 -> 看美食」的網頁應用 。透過 AI 即時生成擬真的美食預覽圖，輔助使用者快速做出點餐決策，消除語言隔閡與資訊落差 。

三、 專案重點呈現：調整過程與技術引導

本專案在開發過程中經歷了重大的技術轉折，以下針對**模型選擇的成本考量**與**風格注入技術**進行重點說明。

1. 模型選擇的轉折：為什麼最終放棄 DALL-E 與 Nano Banana Pro？

在專案初期與中期，我們評估了兩種主流方案，但最終發現它們對於學生專題來說都有致命的成本問題：

- **方案 A (原始計畫)：OpenAI DALL-E 3**
  - **優點：** 語意理解強，畫質極高。
  - **缺點：** **成本過高**。依據官方定價，每生成一張圖片需花費約 $0.04 - $0.08 美金。若在 Demo 期間頻繁測試，費用將難以負擔 。
- **方案 B (替代構想)：Nano Banana Pro**
  - **優點：** 擁有獨特的 2.5D 鮮豔畫風，非常適合呈現令人垂涎的美食。
  - **缺點：**要使用 API 需要有 PRO。這同樣意味著需要支付昂貴的算力費用。
- **最終方案 (現在的方法)：Pollinations.ai + 風格模擬**

解決方案： 我們改用免費的 **Pollinations.ai API**。為了重現 Nano Banana Pro 的獨特畫風，我們採用了 **「風格遷移提示 (Style Transfer Prompting)」** 技術。我們不直接運行那個模型，而是告訴 AI 「請模仿 Nano Banana 的風格畫圖」。這個方式盡可能達成了 **0 成本** 且 **保有特定風格** 的雙重目標。**調整與實作：** 我們將架構重構為 **「混合模型策略 (Hybrid Model Strategy)」**：

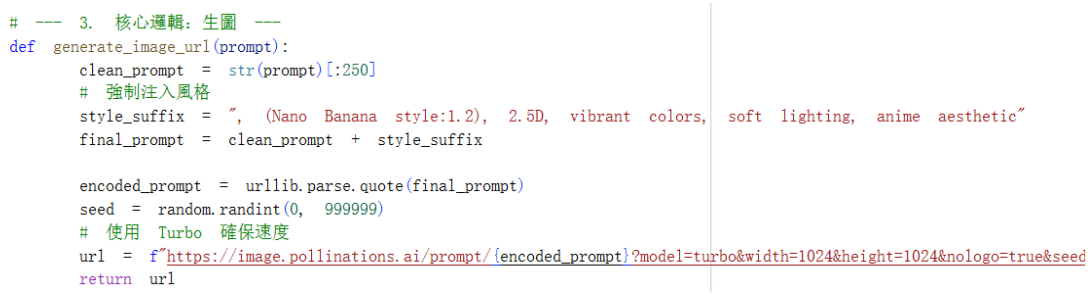
- **大腦升級：** 改用 **Google Gemini 2.5 Flash**。這不僅免費，且在處理中文語意與 OCR 辨識上的速度比 GPT-4o 更快，能達成「秒級翻譯」的體驗。
- **畫筆轉型：** 改用 **Pollinations.ai (Turbo)**。雖然是免費模型，但透過我們後端的優化，其生成速度從原本 DALL-E 的 10 秒縮短至 1-2 秒，大幅提升使用者體驗。

2. 技術解密：如何為每個 Prompt 注入風格修飾詞？

在開發過程中，我們發現若只給 AI 簡單的菜名（如 "Chicken"），生成的圖片會非常單調。因此，我們在程式碼後端實作了「強制風格注入 (Style Injection)」機制。

實作方式： 我們在 generate_image_url 函式中定義了一個 style_suffix (風格後綴) 變數。

- **程式碼邏輯：**



```
# --- 3. 核心邏輯：生圖 ---
def generate_image_url(prompt):
    clean_prompt = str(prompt)[:250]
    # 強制注入風格
    style_suffix = ", (Nano Banana style:1.2), 2.5D, vibrant colors, soft lighting, anime aesthetic"
    final_prompt = clean_prompt + style_suffix

    encoded_prompt = urllib.parse.quote(final_prompt)
    seed = random.randint(0, 999999)
    # 使用 Turbo 確保速度
    url = f"https://image.pollinations.ai/prompt/{encoded_prompt}?model=turbo&width=1024&height=1024&nologo=true&seed
    return url
```

```
# 來自核心程式碼片段
def generate_image_url(prompt):
    clean_prompt = str(prompt)[:250]
    # 強制注入風格：這裡就是關鍵！
    # 無論 Gemini 寫了什麼，我們都在最後面加上這串指令
    style_suffix = ", (Nano Banana style:1.2), 2.5D, vibrant colors, soft lighting, anime aesthetic"
    final_prompt = clean_prompt + style_suffix

    # ... (後續編碼與呼叫 API)
```

- **效果：** 系統會自動將使用者的菜色描述與這串風格關鍵字結合。
  - *原始輸入：* "Grilled Chicken"
  - *注入後：* "Grilled Chicken**, (Nano Banana style:1.2), 2.5D, vibrant colors...**" 這盡可能使每一張生成的圖片都擁有一致、高品質的 Nano Banana 視覺風格。

4. 網路架構突破：Cloudflare Tunnel 的應用

- **實作挑戰：** 由於專案運行於 Google Colab 的虛擬環境中，本機的網頁無法直接讓外部存取。

- **解決方案：** 引用 **Cloudflare Tunnel (`cloudflared`)** 技術。
    - 這項技術能在 Colab 與外部網路之間建立一條安全隧道，生成一個公開的 HTTPS 網址 (`trycloudflare.com`)。
    - 這不僅解決了 Demo 時的連線問題，也模擬了真實網頁上線的部署情境。

四、 專案最終成果

本專案成功建構了一個基於 Streamlit 的互動式網頁應用。

1. 文字生成與分析成果

下圖展示了系統上傳菜單後的分析介面。Gemini 模型成功辨識了圖片中的文字 "`Poulet Sauté, Sauce Chasseur`"，並將其翻譯為「香煎雞肉佐獵人醬」，加上文字描述詳細的口感介紹。

- **操作介面截圖：**

2. 圖像生成成果

透過上述的「風格注入」技術，當使用者點擊生成按鈕時，系統會即時產生對應的美食圖，該網址在手機上也可操作生成。

- **生成效果說明：**生成的圖片具備以下 **Nano Banana 特徵**：
    - **高飽和度 (Vibrant Colors)：** 食物色澤鮮豔誘人。
    - **2.5D 質感：** 介於寫實照片與動漫插畫之間的獨特風格。
    - **光影強化：** 強調食物的光澤感。

- 操作介面截圖：


設定
請輸入 API Key
清除重來
£19.75
12. 香煎雞肉佐獵人醬 (0.0)
原文： Poulet Sauté, Sauce Chasseur
介紹：香煎至金黃的雞肉，搭配經典的法式獵人醬，醬汁通常由蘑菇、青蔥、白酒和番茄製成，風味濃郁。此為主菜選項之一，為「套餐 (Formule)」的一部分，該套餐價格為£19.75。


FeastVision (Elite Access)
https://...ed.trycloudflare.com
9. 芥末兔肉 (星期六特餐) (15.75)
原文： Lapin à la Moutarde
介紹： 傳統法式菜餚，將兔肉以濃郁的芥末醬汁燉煮，肉質鮮嫩入味，為星期六的 Plats du Jour。

四、 程式碼說明

先確認自己的 gemini api 可用的模型有哪些，以調整後續操作(目前預設 `models/gemini-2.5-flash`)

```
# API Key 診斷程式
import google.generativeai as genai
import getpass

print("請貼上你的 Gemini API Key 按下 Enter:")
my_key = getpass.getpass()

try:
        genai.configure(api_key=my_key)
        print("\n正在嘗試連線 Google 伺服器...")

        # 1. 測試列出所有可用模型
        print("📋 你的帳號可用的模型清單：")
        count = 0
        for m in genai.list_models():
            if 'generateContent' in m.supported_generation_methods:
                print(f" - {m.name}")
                count += 1

        if count > 0:
            print(f"\n API Key 是有效的，且抓到了 {count} 個模型。")
            print("執行 app.py 的步驟了 (記得 API Key 要貼剛剛這組)。")
        else:
            print("\n API Key 沒有權限存取任何模型。")

except Exception as e:
        print("\n 測試失敗！你的 API Key 有問題，或是套件版本太舊。")
        print(f"完整錯誤訊息: {e}")
```

套用模型辨識文字並固定 prompt 與輸出的 json 格式，確保後續生成圖片的指令(prompt 有請 AI 稍微調整，json 格式請 AI 幫忙統一)

```

# --- 2. 核心邏輯：Gemini 識圖與分析 ---
def analyze_menu_gemini(api_key, image):
        genai.configure(api_key=api_key)

        candidate_models = [
            "models/gemini-2.5-flash"
        ]

        prompt = """
        你是一位專業的美食評論家。請分析這張菜單圖片。
        識別每一道菜，並輸出嚴格的 JSON 格式。

        JSON 結構必須包含 "items" 列表，每個物件包含：
        - "original_name": 原文菜名
        - "price": 價格
        - "translated_name": 繁體中文翻譯
        - "description": 簡短介紹
        - "image_prompt": 英文繪圖提示詞。

        【重要指令】
        請為 "Nano Banana" (Stable Diffusion model) 撰寫最佳提示詞。
        風格要求：'Nano Banana style', '2.5D', 'vibrant colors', 'soft lighting', 'highly detailed', 'semi-realistic'.

        直接輸出 JSON string。
        """

        for model_name in candidate_models:
            try:
                print(f"嘗試模型: {model_name}...")
                model = genai.GenerativeModel(model_name)
                response = model.generate_content([prompt, image])
                text = response.text.replace("```json", "").replace("```", "").strip()
```

圖片生成並下載，避免生成圖片無法回傳至顯示介面輸出

```
# --- 3. 核心邏輯：生圖 ---
def generate_image_url(prompt):
    clean_prompt = str(prompt)[:250]
    # 強制注入風格
    style_suffix = ", (Nano Banana style:1.2), 2.5D, vibrant colors, soft lighting, anime aesthetic"
    final_prompt = clean_prompt + style_suffix

    encoded_prompt = urllib.parse.quote(final_prompt)
    seed = random.randint(0, 999999)
    # 使用 Turbo 確保速度
    url = f"https://image.pollinations.ai/prompt/{encoded_prompt}?model=turbo&width=1024&height=1024&nologo=true&seed={seed}"
    return url

# --- 安全下載圖片函式 ---
def download_image(url):
    for attempt in range(2): # 試 2 次就好，失敗就切換方案 B
        try:
            headers = {'User-Agent': 'Mozilla/5.0'}
            response = requests.get(url, headers=headers, timeout=10)
            if response.status_code == 200:
                return Image.open(BytesIO(response.content))
        except Exception:
            time.sleep(1)
    return None
```

介面撰寫設計

```
# --- 5. 前端介面 ---
st.title("✨ 異國菜單具象化")
st.caption("Accessing: Gemini 2.5 / 3.0 / Nano Banana Preview")

with st.sidebar:
    st.header("⚙️ 設定")
    api_key = st.text_input("請輸入 API Key", type="password", key="api_key_final")

    if st.button("清除重來"):
        st.session_state.menu_data = None
        st.session_state.generated_images = {}
        st.rerun()

uploaded_file = st.file_uploader("📷 上傳菜單 (JPG/PNG)", type=["jpg", "jpeg", "png"])

if uploaded_file and api_key:
    image = Image.open(uploaded_file)
    st.image(image, caption="原始菜單", use_container_width=True)

    if st.button("🚀 開始分析"):
        with st.spinner("正在分析中..."):
            result = analyze_menu_gemini(api_key, image)
            if result:
                st.session_state.menu_data = result.get("items", [])
                st.success(f"成功辨識 {len(st.session_state.menu_data)} 道菜！")
                st.rerun()

if st.session_state.menu_data:
    st.divider()
    st.subheader("🍽️ 分析結果")
    for idx, item in enumerate(st.session_state.menu_data):
        with st.container():
            col1, col2 = st.columns([3, 2])
```

```
        with col1:
            st.markdown(f"### {idx+1}. {item['translated_name']} ({item['price']})")
            st.markdown(f"**原文：** `{item['original_name']}`")
            st.markdown(f"**介紹：** {item['description']}")
        with col2:
            if idx in st.session_state.generated_images:
                content = st.session_state.generated_images[idx]

                # 方案 A: 圖片物件 (伺服器下載成功)
                if isinstance(content, Image.Image):
                    st.image(content, caption="AI 預覽圖", use_container_width=True)

                # 方案 B: 網址字串 (伺服器下載失敗，改用瀏覽器顯示)
                elif isinstance(content, str):
                    # 使用 Markdown 語法顯示圖片，這會由你的瀏覽器去抓圖，絕對不會被擋
                    st.markdown(f"![AI 預覽圖]({content})")
                    st.caption("AI 預覽圖 (瀏覽器載入)")
            else:
                if st.button(f"✨ 生成圖片", key=f"btn_{idx}"):
                    status_box = st.empty()
                    status_box.info("AI 繪圖中...")

                    img_url = generate_image_url(item['image_prompt'])

                    # 嘗試下載
                    img = download_image(img_url)

                    if img:
                        st.session_state.generated_images[idx] = img
                    else:
                        # 下載失敗沒關係，我們存網址，讓前端 Markdown 自己去抓
                        st.session_state.generated_images[idx] = img_url

                    status_box.empty()
```

Streamlit 啟用並生成網址

```
# Cell 3: 啟動 Streamlit 並使用 Cloudflare Tunnel (推薦)
import subprocess
import time

print("正在安裝 Cloudflare Tunnel...")
# 下載並安裝 cloudflared
!curl -L --output cloudflared.deb https://github.com/cloudflare/cloudflared/releases/latest/download/cloudflared-linux-amd64.deb
!dpkg -i cloudflared.deb > /dev/null 2>&1
!rm cloudflared.deb

print("正在啟動伺服器...")
# 背景執行 Streamlit
!streamlit run app.py > /dev/null 2>&1 &

# 等待 Streamlit 啟動
time.sleep(3)

print("正在建立隧道 (這可能需要 10-15 秒)...")
# 背景執行 Cloudflare Tunnel 並將輸出導向檔案以便讀取網址
!cloudflared tunnel --url http://localhost:8501 > tunnel_log.txt 2>&1 &

# 讀取並顯示網址
time.sleep(5)
print("👇 請點擊下方連結進入你的專題 (無需密碼)：")
!grep -o 'https://.*\.trycloudflare.com' tunnel_log.txt | head -n 1
```