

期末專案

——作業時間排程器——

台北大學 711483120 資工研一 簡瀚佑

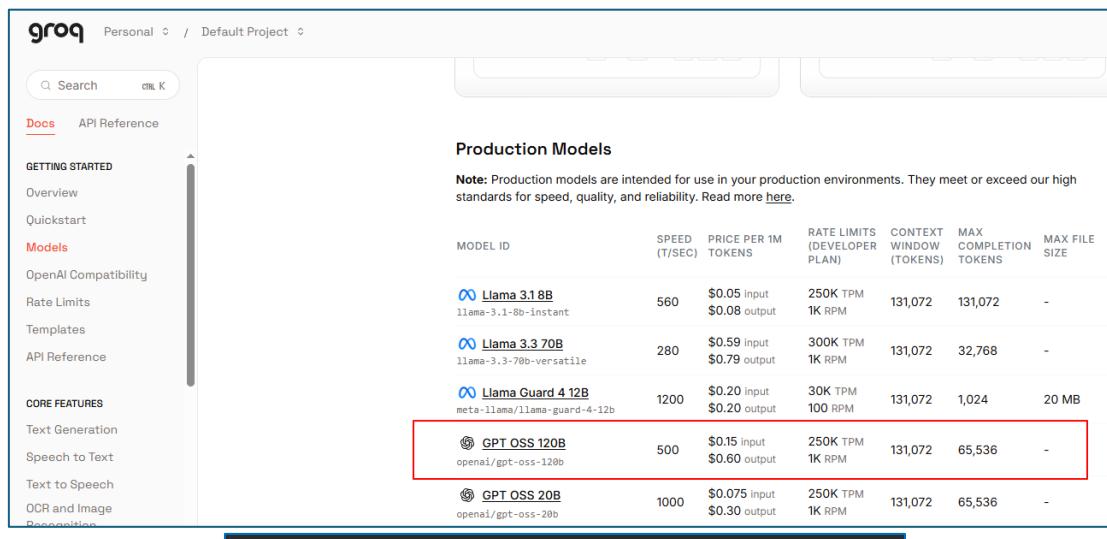
==== 檔案連結 ===

Youtube 影片連結：<https://youtu.be/VfkdVOY82lw>

GitHub 實作與程式碼連結：<https://github.com/JY-soldier/Task-Scheduler>

==== 專案使用模型 ===

本次專案使用 Groq 的免費模型 "openai/gpt-oss-120b"。



MODEL ID	SPEED (T/SEC)	PRICE PER 1M TOKENS	RATE LIMITS (DEVELOPER PLAN)	CONTEXT WINDOW (TOKENS)	MAX COMPLETION TOKENS	MAX FILE SIZE
Llama 3.1.8B llama-3.1-8b-instant	560	\$0.05 input \$0.08 output	250K TPM 1K RPM	131,072	131,072	-
Llama 3.3.70B llama-3.3-70b-versatile	280	\$0.59 input \$0.79 output	300K TPM 1K RPM	131,072	32,768	-
Llama Guard 4.12B meta-llama/llama-guard-4-12b	1200	\$0.20 input \$0.20 output	30K TPM 100 RPM	131,072	1,024	20 MB
GPT OSS 120B openai/gpt-oss-120b	500	\$0.15 input \$0.60 output	250K TPM 1K RPM	131,072	65,536	-
GPT OSS 20B openai/gpt-oss-20b	1000	\$0.075 input \$0.30 output	250K TPM 1K RPM	131,072	65,536	-

```
# 想用哪個 Groq 模型改這行即可
#GROQ_MODEL_NAME = "llama-3.1-8b-instant" # Groq LLM 使用的模型名稱
GROQ_MODEL_NAME = "openai/gpt-oss-120b"

# 呼叫 Groq 的 chat.completions API 產生回應
completion = client.chat.completions.create(
    model=GROQ_MODEL_NAME, # 指定模型名稱
    messages=[
        {"role": "system", "content": system_prompt2}, # system 訊息, 定義任務與格式
        {"role": "user", "content": raw_text}, # user 訊息, 放使用者輸入的原始文字
    ],
    temperature=0.2, # 溫度設比較低, 結果更穩定
)
```

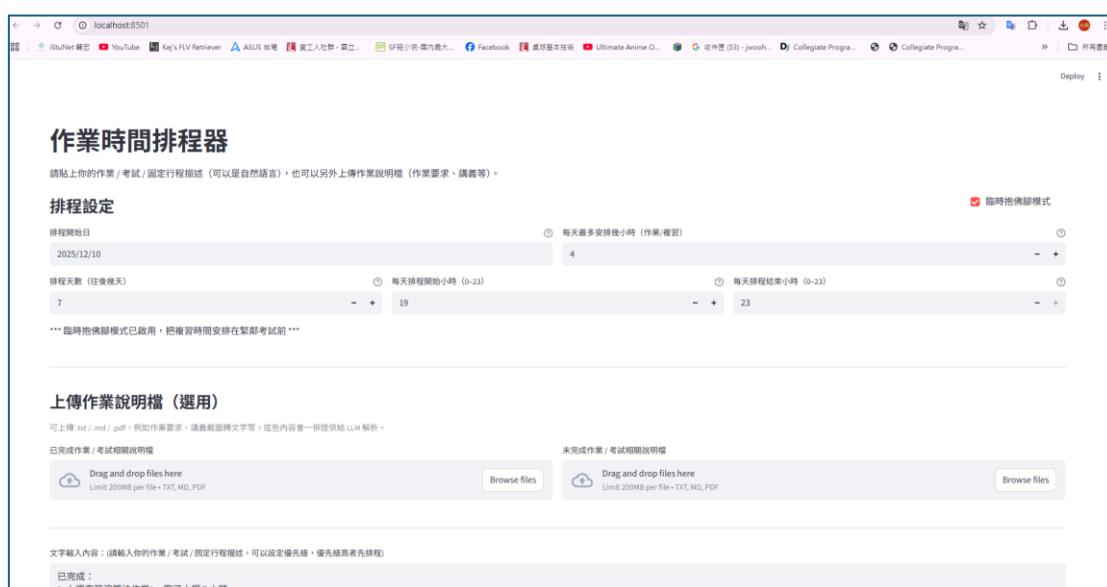
==== 專案目標 ===

製作一個可以幫助使用者安排作業/報告準備/考試複習等行程的排程器。

使用者可以輸入已完成、未完成(待安排)和固定(不能更動)的行程資訊，由 LLM 估算時間並透過演算法安排生成行事曆。

==== 排程器下載說明 ====

- 前往 GitHub 下載整個 launcher 資料夾(也可以直接下載整個專案.zip)。
- 前往 <https://console.groq.com/keys> 申請並記下 API key。
- 打開 Windows PowerShell 將 API key 設定成環境變數。
 - 輸入 :`setx GROQ_API_KEY "你的 Groq 金鑰"`
 - 重新開啟 Windows PowerShell 並輸入：
`echo $Env:GROQ_API_KEY`
- 若回傳內容是你的 key 值，表示設定成功。
- 打開第一步下載的 launcher 資料夾並雙擊啟動 launcher.exe 打開網頁
- 開始使用作業時間排程器 !!



==== 排程器使用說明 ====

排程設定：設定排程條件。



- 臨時抱佛腳模式：將考試複習行程安排在緊鄰考試的時間段複習。
- 排程開始日：從哪一天開始排 "未完成行程"。
- 每天最多安排幾小時：每天安排的 "未完成行程" 上限。
- 排程天數：從排程開始日開始計算。
- 每天排程開始/結束小時：每天安排的時間段。

說明檔上傳：可以上傳相關文字檔案給 LLM，協助估算所需時間。

上傳作業說明檔（選用）

可上傳 .txt / .md / .pdf，例如作業要求、議義範例轉文字等。這些內容會一併提供給 LLM 解析。

已完成作業 / 考試相關說明檔

Drag and drop files here
Limit 200MB per file • TXT, MD, PDF

Browse files

數位影像處理hw1.pdf 1.4MB

高等演算法_Exercise_1.pdf 161.5KB

未完成作業 / 考試相關說明檔

Drag and drop files here
Limit 200MB per file • TXT, MD, PDF

Browse files

高等演算法_Exercise2_.pdf 114.0KB

×

×

文字輸入：輸入已完成、未完成、固定行程描述，包含日期、耗時、優先級等等。

文字輸入內容：(请输入你的作業 / 考試 / 固定行程描述，可以設定優先級。優先級最高者先排程)

已完成：

- 上傳高等演算法作業1，寫了大概7小時。
- 上傳數位影像處理作業1，寫了大概12小時。
- 上次線代小考前複習3小時。
- 上次計算機圖學期中考複習8小時。

未完成：

- 12/09 早上9點要考線性代數小考1。
- 12/11 晚上11:59要交高等演算法作業2。

生成排程

- 生成排程：以上都設定好就可以按下按鈕開始排程。

逾期任務：截止時間已經超過當前時間，不計入排程。

逾期任務

以下任務的截止時間已經過去，因此不會被排入未來的時程中：

標題	科目	截止時間	預估時間(分鐘)
0 線性代數小考 1	線性代數	2025-12-09 09:00	120

無法完全安排的任務：截止時間前無法排完的任務，"已排時間"大於 0 表示該行程仍有部分排入行事曆。

無法完全安排的任務

以下任務在目前設定的排程天數 / 每日作業上限 / 每天排程時段內，無法完全排完預估所需時間：

標題	科目	截止時間	預估時間(分鐘)	已排時間(分鐘)	尚未排入(分鐘)
0 數位影像處理作業 2	數位影像處理	2025-12-12 23:59	720	600	120
1 高等網路期末考	高等網路	2025-12-18 13:00	180	120	60

排程列表：排入行事曆的行程，按照時間順序排列。

排程列表

標題	開始	結束	種類
0 高等演算法作業 3	2025-12-10 12:00	2025-12-10 13:00	作業/複習
1 高等演算法作業 3	2025-12-10 13:00	2025-12-10 14:00	作業/複習
2 高等演算法作業 3	2025-12-10 14:00	2025-12-10 15:00	作業/複習
3 高等演算法作業 3	2025-12-10 15:00	2025-12-10 16:00	作業/複習
4 高等演算法作業 3	2025-12-10 16:00	2025-12-10 17:00	作業/複習
5 高等演算法作業 3	2025-12-10 17:00	2025-12-10 18:00	作業/複習
6 高等演算法作業 3	2025-12-10 18:00	2025-12-10 19:00	作業/複習

ICS 檔下載：下載可以匯入日曆的 ICS 檔。由於 ICS 檔不支援顏色設定，故提供固定行程和未完成行程分開的版本，供使用者分開匯入不同日曆產生不同顏色，方便查看。

匯入 Google Calendar
(或者其他支援.ics檔的日曆)

單一日曆下載
【全部行程】.ics

分成兩個日曆下載
方便分開匯入不同日曆，以顯示不同顏色
【固定行程】.ics
【作業/考試複習】.ics

Google Calendar匯入流程

我的日曆
JY小兵
Tasks
生日
其他日曆
台灣的節慶假日

其他日曆
台灣的節慶假日

Step 1

Step 2

訂閱日曆
建立新日曆
瀏覽感興趣的日曆
加入日曆網址

匯入 Step 3 選擇匯入檔案和日曆

fixed_events.ics

新增至日曆
固定行程

你可以使用 iCalendar (ICS/VCS) 或 CSV 格式的檔案匯入活動資訊。

匯入 Step 4

• 單一日曆：

9 (廿十)	10 (廿一)	11 (廿二)	12 (廿三)	13 (廿四)
<ul style="list-style-type: none"> ● 下午12點 高等演算法作業 3 ● 下午1點 高等演算法作業 3 ● 下午2點 高等演算法作業 3 ● 下午3點 高等演算法作業 3 <p>還有 4 個</p>	<ul style="list-style-type: none"> ● 下午12點 高等演算法作業 2 ● 下午1點 高等演算法作業 2 ● 下午2點 高等演算法作業 2 ● 下午3點 高等演算法作業 2 <p>還有 4 個</p>	<ul style="list-style-type: none"> ● 下午12點 數位影像處理作業 2 ● 下午1點 數位影像處理作業 2 ● 下午2點 數位影像處理作業 2 ● 下午3點 數位影像處理作業 2 <p>還有 4 個</p>	<ul style="list-style-type: none"> ● 下午6點 和家人吃飯 	
16 (廿七)	17 (廿八)	18 (廿九)	19 (三十)	20 (十一月)
<ul style="list-style-type: none"> ● 下午7點 補習 ● 下午9點 線性代數小考 2 ● 下午10點 線性代數小考 2 	<ul style="list-style-type: none"> ● 下午9點 高等網路期末考 ● 下午10點 高等網路期末考 		<ul style="list-style-type: none"> ● 下午8點 高等演算法期末考 ● 下午9點 高等演算法期末考 ● 下午10點 高等演算法期末考 	

• 分開日曆：

9 (廿十)	10 (廿一)	11 (廿二)	12 (廿三)	13 (廿四)
<ul style="list-style-type: none"> ● 下午12點 高等演算法作業 3 ● 下午1點 高等演算法作業 3 ● 下午2點 高等演算法作業 3 ● 下午3點 高等演算法作業 3 <p>還有 4 個</p>	<ul style="list-style-type: none"> ● 下午12點 高等演算法作業 2 ● 下午1點 高等演算法作業 2 ● 下午2點 高等演算法作業 2 ● 下午3點 高等演算法作業 2 <p>還有 4 個</p>	<ul style="list-style-type: none"> ● 下午12點 數位影像處理作業 2 ● 下午1點 數位影像處理作業 2 ● 下午2點 數位影像處理作業 2 ● 下午3點 數位影像處理作業 2 <p>還有 4 個</p>	<ul style="list-style-type: none"> ● 下午6點 和家人吃飯 	
<ul style="list-style-type: none"> ● 下午7點 補習 ● 下午9點 線性代數小考 2 ● 下午10點 線性代數小考 2 	<ul style="list-style-type: none"> ● 下午9點 高等網路期末考 ● 下午10點 高等網路期末考 		<ul style="list-style-type: none"> ● 下午8點 高等演算法期末考 ● 下午9點 高等演算法期末考 ● 下午10點 高等演算法期末考 	20 (十一月)

==== 排程器實作流程 ===

首先使用 Chat GPT 生成整個專案的基礎框架程式碼。

我想使用生成式AI製作一個作業時間排程應用，以下是條件：

1. 可以輸入的內容(過去完成的作業(包含題目、結果、消耗時間，也可以是複習考試內容)、尚未完成的作業(包含題目、截止日期，也可以是複習考試內容)、已有安排不能更改的日期、時間(包含上課、聚餐、考試、回家等)等使用者認為已經有事的資訊)

2. 輸出內容--> 請程安排表(內容必須包含所有輸入的未完成作業和有安排不能更改事項，以不同顏色標示)

3. 生成式AI根據使用者輸入內容來執行時程安排表*

*輸入內容可以是複習或者文字輸入或二選皆有
請幫我構造一個實際的框架，輸入生成式AI則輸出這一段過程要點做調整(不一定是單一模型輸出結果)

已思考 15s >

先直接給你一個「模組化 pipeline」的整體架構，從「輸入 → 生成式 AI → 排程結果」整條路都可以拆開調整，不會堵死在單一模組。

一、整體概念：把問題拆成 4 個階段

你要的東西可以拆成四段（每一段都可以換模式 / 檢測算法）：

- 資料輸入與前處理
 - 接收文字、檔案、行事曆等各種輸入。
- 語義理解與結構化 (LLM 模組 A)
 - 把「作業描述：已排好行程」轉成統一的結構化資料 (JSON)。
- 排程規劃 (演算法 + LLM 模組 B)
 - 根據歷史 + 劍止日 + 占用時間，做拆分。
- 根據歷史 + 劍止日 + 占用時間，做拆分。

一、config.py : 設定

```
python
# config.py
import os

OPENAI_API_KEY = os.getenv("OPENAI_API_KEY") # export OPENAI_API_KEY
DEFAULT_STUDY_START_HOUR = 19 # 每天 19:00-23:00 都可讀書時間
DEFAULT_STUDY_END_HOUR = 23
BLOCK_MINUTES = 60 # 每隔 1 小時
```

四、llm_parser.py : 用 LLM 把文字 → JSON

這裡示範一個簡單 prompt，實際你可以再加強。

```
python
# llm_parser.py
from openai import OpenAI
from datetime import datetime
from models import ParsedInput
from config import OPENAI_API_KEY

client = OpenAI(api_key=OPENAI_API_KEY)

SYSTEM_PROMPT = """
你是一個能自動排學業行程的助理。
使用者會黏貼出自訂的說明檔說明他的作業、考試和已經排好的行程。
請回答：JSON 構式：
```

接著讓 Chat GPT 調整並補上一些額外內容(說明檔上傳、輸出 ics 檔、把程式包裝成 exe 檔)。

增加輸入權限功能，包括已完成和未完成作業的說明檔(作業要求之類的)

已思考 1m 58s >

可以，加「上傳作業說明檔」很適合但這個很難。

我們只需要在 app.py 上加 file_uploader，然後把檔案內容一起丟給 LLM 做解析就好。

下面給你一個完整的 app.py，你可以直接複製複製現在的 app.py 並使用。

(其它檔案像 llm_parser.py、scheduler.py 都不動)

我有幫你做這個件事：

- 新增兩組上傳區：
- 已完成作業相關說明檔 (例如作業要求、講義)
- 未完成作業 / 考試相關說明檔
- 支援檔案格式：.txt, .md, .pdf
- 請在內容後，用清楚的標註放在原本文字裡後，一起送進 parse_input_with_llm()

需要額外安裝 (用來讀 PDF)

```
bash
pip install PyPDF2
```

改好的 app.py (整份覆蓋即可)

能把排程結果放到google calendar上嗎？

已思考 13s >

可以，而且用你現在的架構 (Streamlit + Python) 很好處理：

把排程結果轉成 .ics 檔 → 讓使用者下載 → 填入 Google Calendar 就好了，不一定要直接串 Google API (那會牽涉 OAuth，會比較麻煩)。

我先給你「能直接用在你專案裡」的版本：

- 新增一個 ics_export.py
- 修改 app.py，在行程後多一個下載 .ics 的按鈕
- 最後教教你怎麼在 Google Calendar 填入

新增 ics_export.py : 把排程轉成 .ics

在你的專案資料夾 /scheduler/ 底下加一個檔案 ics_export.py :

```
python
# ics_export.py
from datetime import datetime
from typing import List
from models import ScheduledBlock
```

我的意思是把所有程式包裝成一個檔案，例如exe檔雙擊即動，類似 visual studio 工程產生一個exe檔，我使用的pycharm

已思考 19s >

你的意思是：你是想要：

像 Visual Studio 那樣做出一個 .xxx.exe，在 Windows 上雙擊就能把 Streamlit 項頁跑起來，而不是只在 PyCharm 按 run 就好。

因為你現在是 Streamlit 項頁，所以就拿個 exe，本質上還是：

```
exe 發動 = 默認一個本機的 Streamlit server → 自動在瀏覽器開啟 http://localhost:8501 *
```

下面給你兩種方法：

做法一 (推薦) : 用 PyInstaller + launcher.py

讓原本的 app.py 完全不用改，只多一個「啟動器」，用 PyInstaller 打包。

第 0 步：確認你現在的結構

假設你的專案是這樣：

```
test
D:\ASUS\Desktop\Program\Python\VisualScheduler\
    app.py
    .venv\ + 你目前的 Streamlit 主程式
    .venv\ ...
    ...
```

第 1 步：在同一個資料夾建立 launcher.py

最後手動修改程式碼調整並添加額外功能，主要修改網頁 UI 設計、排程條件設定、Prompt 提示內容與流程、排程演算法。

```
# ----- 排程設定 -----
col1, col2, col3 = st.columns([1, 3, 1])

with col1:
    st.subheader("排程設定") # 小標題：排程設定區塊

with col3: # 右欄：把考試複習時間安排在緊部考試前
    cram_or_not: bool = st.checkbox("隨時泡佛腳模式", value=True)

col_cfg1, col_cfg2= st.columns(2) # 建立兩個欄，左邊啟始時間，右邊啟每天最多小時

with col_cfg1: # 左欄：排程開始日設定
    today = datetime.datetime.today().date() # 取得今天日期 (不含時間)
    start_date: date = st.date_input(
        "排程開始日", # 欄位標題
        value=today, # 預設值為今天
        min_value=today, # 不允許選擇今天以前
        help="從這一天開始往後排作業/複習時段 (不可早於今天)。", # 指示文字
    )

with col_cfg2: # 右欄：每天最多安排幾小時作業/複習
    max_hours_per_day: int = st.number_input(
        "每天最多安排幾小時 (作業/複習)", # 欄位標題
        min_value=1, # 最小值 1 小時
        max_value=10, # 此大值 10 小時
        value=4, # 設定 4 小時
        step=1, # 每次調整步階為 1
        help="只計算作業/複習時間，不包含固定行程。", # 指示文字
    )

# 定義規則：
# 1. "title" 檢查說明檔的欄名設定。
# 2. "difficulty" 必須是正整數，難度越大則 difficulty 越大，與完成作業所需時間成反比。
# 3. 第一個判斷的說明檔的 "difficulty" 必須設定為 10，其他說明檔的 difficulty 以第一個說明檔的難易度為基準做判斷。
# 4. 只輸出合法 JSON，不要加註解或多餘文字。
"""

# 定義 Graq 的 system prompt，根據說明檔判斷難易度
system_prompt = """
你是一個幫忙代辦學業作業、報告和考試的難易程度的助理。
現在因為學生判斷作業、報告和考試的難易程度。
使用者點出自然語言說明他的作業：報告和考試的難易程度。
請你根據「已完成作業 / 考試說明檔(past_tasks)」和「未完成作業 / 考試說明檔(todos)」輸出JSON。格式為：
{
    "past_tasks": [
        {
            "title": "string",
            "difficulty": 1
        }
    ],
    "todos": [
        {
            "title": "string",
            "difficulty": 1
        }
    ]
}

重要規則：
1. "title" 檢查說明檔的欄名設定。
2. "difficulty" 必須是正整數，難度越大則 difficulty 越大，與完成作業所需時間成反比。
3. 第一個判斷的說明檔的 "difficulty" 必須設定為 10，其他說明檔的 difficulty 以第一個說明檔的難易度為基準做判斷。
4. 只輸出合法 JSON，不要加註解或多餘文字。
"""

# 定義可用的讓使用者點選中間的每個滑鼠移動事件
for slot_start in _iter_study_slots:
    start_date = f"請選擇開始日期 (當天日期) + {slot_start}"
    study_start_hour = f"請選擇開始時間 (當天時間) + {slot_start.hour}"
    study_end_hour = f"請選擇結束時間 (當天時間) + {slot_start.hour}"

    if slot_start > todo.deadline:
        break # 已達過期作業的截止時間，不往下延伸

    if remaining <= 0:
        break # 代表已經用完的分鐘所剩時間，不需要再計算

# 因時間限制的問題
if cram_or_not == True and todo.start_time == today:
    num_of_days = today - start_date
    num_of_minutes = num_of_days * num_minutes_per_day
    num_of_minutes += max((study.end_hour - slot.start.hour), 0) * 60
    num_of_minutes += max(todo.deadline.hour - study.start.hour, 0) * 60
    for slot in _iter_study_slots[slot_start:, num_of_days+2, study.start.hour, study.end.hour]:
        if slot.is_free(slot.slot_estimated.minutes+BLOCK_MINUTES):
            num_of_minutes -= BLOCK_MINUTES
    if num_of_minutes <= 0:
        continue

slot_end = slot_start + timedelta(minutes=blocks.append(ScheduleBlock))

# 檢查時間是否超過一天
if today != start_date:
    day_key = slot_start.date() # 取得該時間點的一天
    used = day_todo_blocks.get(day_key, 0) # 該天是否有已經用掉的 todo 分數
    if used <= BLOCK_MINUTES:
        blocks.append(ScheduleBlock)
    else:
        continue # 不在該地帶停，直接跳下一個

if is_free(slot_start, slot_end): # 若時間範圍沒有被其他 block 占用
    blocks.append(ScheduleBlock)
```

==== 排程器運作邏輯 ===

使用者設定完排程條件並輸入資料後，對輸入資料進行以下處理：

1. 將輸入檔案整理成字串，和輸入文字分開輸入 LLM。

```
# ----- 文字輸入 -----
# 主要文字輸入區，使用者輸上首段語言描述
raw_text = st.text_area("文字輸入內容：(請輸入你的作業 / 考試 / 請定期程描述，可以設定優先級，優先跳高等排程)", value=default_example, height=250)

# 按按下「生成行程」按鈕時，更新 session_state
if st.button("生成行程"): # 建立一間新行程，按下時執行以下區域
    # 把檔案內容讀出來，判斷是否是什麼樣型
    extra_sections = [] # 用來累積上層檔案文字內容的 list

    if done_files: # 給有上層已完成作業說明檔
        for f in done_files: # 逐一處理每個檔案
            content = read_uploaded_file(f) # 讀取並轉為文字
            # 加上說明檔原來是怎麼樣檔案
            extra_sections.append(
                f"[已完成作業 / 考試說明檔: {f.name}]\n{content}"
            )

    if todo_files: # 給有上層半完成作業 / 考試說明檔
        for f in todo_files: # 逐一處理每個檔案
            content = read_uploaded_file(f) # 讀取並轉為文字
            # 加上說明檔原來是怎麼樣檔案
            extra_sections.append(
                f"[半完成作業 / 考試說明檔: {f.name}]\n{content}"
            )

    # 合併成送給 LLM 的完整文字
    assignment_file = "" # 先從文字輸入內容開始
    if extra_sections: # 若有任何上層檔案
        assignment_file += ( # 在原文字後加上直線和各檔案內容
            "\n\n*** 以下為上層的作業說明檔內容 ***\n"
            + "\n\n---\n".join(extra_sections)
        )

    # 呼叫 LLM 解析使用者輸入
    with st.spinner("解析輸入內容中 (呼叫 Graq LLM ...)"):
        parsed = parse_input_with_llm(raw_text, assignment_file, start_date, schedule_days) # 使用 lla_parser 將文字解析成結構化的物件
        if not parsed: st.error("⚠️ 發生錯誤，請確認 API KEY 是否正確，或檢查輸入文字是否有誤。")
        else:
            st.session_state["parsed"] = parsed # 存進 session_state 以便繼續使用
```

2. 先將轉成字串的說明檔輸入 LLM，讓它判斷難易度，並按照 JSON 格式輸出。

如果 LLM 輸出要求以外的內容，則透過程式檢查並篩除。

最後將輸出結果和使用者輸入文字串接在一起。

```
# 定義給 Graq 的 system prompt，根據說明檔判斷難易度
system_prompt = """
你是一個幫忙代辦學業作業、報告和考試的難易程度的助理。
現在因為學生判斷作業、報告和考試的難易程度。
使用者點出自然語言說明他的作業：報告和考試的難易程度。
請你根據「已完成作業 / 考試說明檔(past_tasks)」和「未完成作業 / 考試說明檔(todos)」輸出JSON。格式為：
{
    "past_tasks": [
        {
            "title": "string",
            "difficulty": 1
        }
    ],
    "todos": [
        {
            "title": "string",
            "difficulty": 1
        }
    ]
}

重要規則：
1. "title" 檢查說明檔的欄名設定。
2. "difficulty" 必須是正整數，難度越大則 difficulty 越大，與完成作業所需時間成反比。
3. 第一個判斷的說明檔的 "difficulty" 必須設定為 10，其他說明檔的 difficulty 以第一個說明檔的難易度為基準做判斷。
4. 只輸出合法 JSON，不要加註解或多餘文字。
"""

# 叫叫 Graq 的 chat.completions API 產生回應
completion = client.chat.completions.create(
    model=GROQ_MODEL_NAME, # 指定模型名稱
    messages=[
        {"role": "system", "content": system_prompt}, # system 訊息，定義任務與格式
        {"role": "user", "content": assignment_file}, # user 訊息，敘述使用者輸入的原始文字
    ],
    temperature=0.2, # 溫度設比較低，結果要穩定
)

# 取得 LLM 回傳的主要文字內容
content = completion.choices[0].message.content
raw = content.strip() # 去掉前後空白

# ■ 如果有 `` json ... `` code block, 先去掉外層
if raw.startswith("```"): # 檢查是否以 `` `` 開頭 (包含 `` ``)
    # 去掉前面 `` `` 或 `` `` json 這一層
    raw = re.sub(pattern="````[\s\S]*````", repl="", raw)
    # 去掉最後的 `` ``，以及前面的空白換行
    raw = re.sub(pattern="````[\s\S]*````", repl="\n", raw).strip()

# ■ 只抓第一個 { 到最後一個 } 之間的內容
first = raw.find('{') # 找到第一個 '{' 的位置
last = raw.rfind("}") # 找到最後一個 '}' 的位置
if first == -1 or last == -1: # 若找不到大括號
    print("⚠️ Graq 回傳內容找不到大括號，原始內容: ", content) # 印出原始內容
    return None # 後面 None 是上層啟用 def
```

(1) LLM 輸入內容：

已完成作業 / 考試相關說明檔	未完成作業 / 考試相關說明檔
<div><p> Drag and drop files here Limit 200MB per file • TXT, MD, PDF</p><p> 數位影像處理hw1.pdf 1.4MB ×</p><p> 高等演算法_Exercise1.pdf 181.5KB ×</p></div>	<div><p> Drag and drop files here Limit 200MB per file • TXT, MD, PDF</p><p> 高等演算法_Exercise2_.pdf 114.0KB ×</p></div>

(2) LLM 輸出內容：

```
===== 難度解析結果 =====  
past_tasks=[DifficultyPastTask(title='高等演算法_Exercise_1.pdf', difficulty=10), DifficultyPastTask(title='數位影像處理hw1.pdf', difficulty=9)] todos=[DifficultyTodo(title='高等演算法_Exercise2_.pdf', difficulty=12)]
```

3. 將輸出結果和使用者輸入文字串接後再次輸入 LLM，讓它估算未完成行程的消耗時間。

輸出同樣按照 JSON 格式，並用程式檢查篩除要求以外的輸出。

(1) LLM 輸入內容：

<p>文字輸入內容：(請填入你的作業/考試/因旅行行程推遲，可以已完成；</p> <p>1. 上禮拜幾演算法作業，寫了大概 7 小時。 2. 上禮拜幾演算法作業 1，寫了大概 12 小時。 3. 上次縮小考題做練習 3 小時。 4. 上次計算機概論那題半數後 8 小時。</p> <p>未完成：</p> <p>1. 12/09 早上 8 點要交線性代數小考 1。</p>	<p>3. 12/12 晚上 11:59 要交數位影像處理作業 2。 4. 12/16 晚上 11:59 要交高等演算法作業 3。 5. 12/17 早上 9 點要考線性代數小考 2。 6. 12/22 早上 9 點要考計算機圖學期末考。 7. 12/18 下午 1 點要考高等網路期末考。 8. 12/20 早上 10 點要考高等運算符半期考。</p>
	<p>優先級：線性代數小考 1 > 線性代數小考 2 > 高等演算法作業</p> <p>固定行程：</p> <ol style="list-style-type: none"> 每週二晚上 7 點到 9 點補習。 12/13 晚上 6 點到 8 點和家人吃飯。

難度解析結果 ======
past_tasks=[DifficultyFastTask(title='高等演算法 Exercise_1.pdf', difficulty=10), DifficultyFastTask(title='數位影像處理hw1.pdf', difficulty=9)] todos=[DifficultyTodo(title='高等演算法 Exercise_2_.pdf', difficulty=12)]

(2) LLM 輸出內容：

LLM 解析結果 (debug 用)	todos	fixed_events
<pre> "past_tasks": [{ "title": "高等演算法作業1", "subject": "高等演算法", "time_spent_minutes": 420, "difficulty": 10 }, { "title": "前回影劇作業1", "subject": "影劇作業", "time_spent_minutes": 720, "difficulty": 9 }, { "title": "線性代數小考復習", "subject": "線性代數", "time_spent_minutes": 180, "difficulty": 1 }, { "title": "高等演算法作業2", "subject": "高等演算法", "deadline": "datetime.datetime(2025, 12, 9, 0)", "estimated_time_minutes": 180, "difficulty": -1, "priority": 1, "exam_or_not": true }], "now_tasks": [{ "title": "影劇作業", "subject": "影劇作業", "deadline": "datetime.datetime(2025, 12, 11, 23, 59)", "estimated_time_minutes": 504, "difficulty": 12, "priority": 101, "exam_or_not": false }, { "title": "計算機圖形學作業2", "subject": "計算機圖形學", "deadline": "datetime.datetime(2025, 12, 12, 23, 59)", "estimated_time_minutes": 720, "difficulty": -1, "priority": 101, "exam_or_not": false }], "future_tasks": [{ "title": "高等網路期末考", "subject": "高等網路", "deadline": "datetime.datetime(2025, 12, 18, 13, 0)", "estimated_time_minutes": 240, "difficulty": -1, "priority": 101, "exam_or_not": true }, { "title": "高等演算法期末考", "subject": "高等演算法", "deadline": "datetime.datetime(2025, 12, 20, 10, 0)", "estimated_time_minutes": 504, "difficulty": -1, "priority": 101, "exam_or_not": true }] </pre>	<pre> [{"id": 1, "title": "線性代數小考 1", "subject": "線性代數", "deadline": "datetime.datetime(2025, 12, 9, 0)", "estimated_time_minutes": 180, "difficulty": -1, "priority": 1, "exam_or_not": true}, {"id": 2, "title": "高等演算法作業 2", "subject": "高等演算法", "deadline": "datetime.datetime(2025, 12, 11, 23, 59)", "estimated_time_minutes": 504, "difficulty": 12, "priority": 101, "exam_or_not": false}, {"id": 3, "title": "計算機圖形學作業 2", "subject": "計算機圖形學", "deadline": "datetime.datetime(2025, 12, 12, 23, 59)", "estimated_time_minutes": 720, "difficulty": -1, "priority": 101, "exam_or_not": false}, {"id": 4, "title": "影劇作業", "subject": "影劇作業", "deadline": "datetime.datetime(2025, 12, 11, 23, 59)", "estimated_time_minutes": 504, "difficulty": 12, "priority": 101, "exam_or_not": false}, {"id": 5, "title": "高等演算法期末考", "subject": "高等演算法", "deadline": "datetime.datetime(2025, 12, 20, 10, 0)", "estimated_time_minutes": 504, "difficulty": -1, "priority": 101, "exam_or_not": true}], [{"start": "2025-12-18T13:00:00", "end": "2025-12-18T13:30:00", "title": "補習"}, {"start": "2025-12-23T19:00:00", "end": "2025-12-23T21:00:00", "title": "補習"}, {"start": "2025-12-13T18:00:00", "end": "2025-12-13T20:00:00", "title": "和家人吃飯"}] </pre>	<pre> [{"id": 1, "title": "影劇作業", "subject": "影劇作業", "deadline": "datetime.datetime(2025, 12, 11, 23, 59)", "estimated_time_minutes": 504, "difficulty": 12, "priority": 101, "exam_or_not": false}, {"id": 2, "title": "計算機圖形學作業 2", "subject": "計算機圖形學", "deadline": "datetime.datetime(2025, 12, 12, 23, 59)", "estimated_time_minutes": 720, "difficulty": -1, "priority": 101, "exam_or_not": false}, {"id": 3, "title": "高等演算法作業 2", "subject": "高等演算法", "deadline": "datetime.datetime(2025, 12, 11, 23, 59)", "estimated_time_minutes": 504, "difficulty": 12, "priority": 101, "exam_or_not": false}, {"id": 4, "title": "高等演算法期末考", "subject": "高等演算法", "deadline": "datetime.datetime(2025, 12, 20, 10, 0)", "estimated_time_minutes": 504, "difficulty": -1, "priority": 101, "exam_or_not": true}], [{"start": "2025-12-18T13:00:00", "end": "2025-12-18T13:30:00", "title": "補習"}, {"start": "2025-12-23T19:00:00", "end": "2025-12-23T21:00:00", "title": "補習"}, {"start": "2025-12-13T18:00:00", "end": "2025-12-13T20:00:00", "title": "和家人吃飯"}] </pre>

4. 將 LHM 最終輸出結果和使用者設定的排程條件一起輸入排程演算法進行排程。

排程邏輯如下：

- (1) 排程按照使用者設定的每日限定時段、上限時間等條件將行程以 60 分鐘為單位插入使用者設定的日期範圍。
 - (2) 超時行程會被篩除、固定行程時段不得更動。
 - (3) 如果使用者有設定優先級，則按照優先級順序排程。
 - (4) 未設定優先級則按照截止日期先後順序排程。
 - (5) 如果使用者開啟 "臨時抱佛腳模式"，則考試複習行程安排在緊鄰考試時間前。

```
# 根據解析結果進行排程
with st.spinner("排程中..."):
    if parsed:
        schedule = build_schedule(
            parsed, # 解析後的結構化資料
            days=schedule_days, # 往後排幾天，由使用者輸入
            start_date=start_date, # 排程開始日期
            max_hours_per_day=max_hours_per_day, # 每天最多作業/複習小時
            study_start_hour=int(study_start_hour), # 每天排程開始小時（轉成 int）
            study_end_hour=int(study_end_hour), # 每天排程結束小時（轉成 int）
            cram_or_not=cram_or_not, # 試算複習時間安排模式
        )
        st.session_state["schedule"] = schedule # 排好的結果存到 session_state

# 依 deadline 排序 todo (最近截止的先排)
todos = sorted(parsed.todos, key=lambda x: x.deadline)

# 依 priority 排序 todo (優先級最高的先排)
todos = sorted(todos, key=lambda x: X.priority)

# 依序處理每一個 todo 任務
for todo in todos:
    remaining = todo.estimated_time_minutes # 利用前面開的分鐘數

    # 刪掉已過期在 now 之前，就不排（之後在 UI 雖為強制）
    if todo.deadline <= now:
        continue # 直接跳過這個 todo

    # 在可用的課時時間間隙中嘗試排這個任務
    for slot_start in _iter_study_slots(
        start_base, # 課程起始時間 (台灣時間 + 00:00)
        days, # 請排幾天
        study_start_hour, # 每天排程開始小時
        study_end_hour, # 每天排程結束小時
    ):
        if slot_start >= todo.deadline:
            # 已經超過這段作業的截止時間，不再選擇
            break

        if remaining <= 0:
            break # 代表已經排到最後時間，不需要再找時間

        # 請勿使用中文關鍵字
        if cram_or_not == True and todo.exam_or_not == True:
            num_of_days = todo.deadline.day - slot.start_day - 1
            num_of_minutes = num_of_days * max_minutes_per_day
            num_of_minutes += max((slot.end_hour - slot.start_hour) * 60, 0) * 60
            num_of_minutes -= todo.estimated_time_minutes - (slot.end_hour - slot.start_hour) * 60
            if slot.end_hour == slot.start_hour:
                num_of_minutes -= BLOCK_MINUTES

            if not is_free(slot, slot.minutes_left(minutes=BLOCK_MINUTES)):
                continue

            if todo.estimated_time_minutes < num_of_minutes:
                continue
```

5. 在網頁上顯示排程結果並提供 ICS 檔案下載匯入第三方日曆程式。

```
# ✓ 顯示排程結果 * 排不完任務 * 下載 .ics
if st.session_state["schedule"] is not None: # 確已有排程結果
    st.subheader("排程結果") # 小標題：排程結果
    schedule = st.session_state["schedule"] # 取出排程結果列表
    parsed = st.session_state["parsed"] # 再取一次解析結果，方便後續對比
```

```

# ---- 2. 正常排程列表 (表格) ----
st.markdown("#### 排程列表") # 小標題：排程列表
# 將所有排程區塊整理成表格資料
table_data = [
    {
        "標題": b.title, # 事件標題
        "開始": b.start.strftime("%Y-%m-%d %H:%M"), # 開始時間字符串
        "結束": b.end.strftime("%Y-%m-%d %H:%M"), # 結束時間字符串
        "種類": "作業/複習" if b.kind == "todo" else "固定行程", # 依 kind 顯示中文種類
    }
    for b in schedule
]
st.table(table_data) # 用表格顯示所有已排事件

```

```

# ---- 3. 下載 .ics ----
st.subheader("匯入 Google Calendar") # 小標題：Google Calendar 區塊
st.caption("( 或者其他支援.ics檔的日曆 )")

st.markdown("#### 單一日曆下載") # 小標題：Google Calendar 區塊

# 3-1 全部一起版本
all_ics = schedule_to_ics(schedule) # 將完整排程轉成單一 .ics 字串
st.download_button(
    label="【全部行程】.ics", # 按鍵文字
    data=all_ics, # .ics 檔案內容
    file_name="study_schedule_all.ics", # 下載檔名
    mime="text/calendar", # MIME 型別
)

```

```

# 3-2 Google Calendar 兩個日曆版本
fixed_ics, todo_ics = split_schedule_to_ics_for_google(schedule) # 分拆兩份 .ics

st.markdown("#### 分成兩個日曆下載") # 小標題：Google Calendar 區塊
st.caption("方便分開匯入不同日曆，以顯示不同顏色")

# 下載固定行程 .ics
st.download_button(
    label="【固定行程】.ics", # 按鍵文字
    data=fixed_ics, # 固定行程 .ics 內容
    file_name="fixed_events.ics", # 檔名
    mime="text/calendar", # MIME 型別
)

# 下載作業 / 複習 .ics
st.download_button(
    label="【作業/考試複習】.ics", # 按鍵文字
    data=todo_ics, # 作業 / 複習 .ics 內容
    file_name="tasks_events.ics", # 檔名
    mime="text/calendar", # MIME 型別
)

```

6. 排程器輸出結果

排程結果					
無法完全安排的任務					
以下任務在目前設定的排程天數 / 每日作業上限 / 每日排程時段內，無法完全排完預估所需時間：					
標題	科目	截止時間	預估時間(分鐘)	已排時間(分鐘)	尚未排入(分鐘)
0 高等演算法作業 2	高等演算法	2025-12-11 23:59	504	420	84
1 數位影像處理作業 2	數位影像處理	2025-12-12 23:59	720	480	240
2 高等演算法期末考	高等演算法	2025-12-20 10:00	504	480	24

排程列表					
標題	開始	結束	種類	作業/複習	作業/複習
0 高等演算法作業 3	2025-12-10 13:00	2025-12-10 14:00	作業	作業	複習
1 高等演算法作業 3	2025-12-10 14:00	2025-12-10 15:00	作業	作業	複習
2 高等演算法作業 3	2025-12-10 15:00	2025-12-10 16:00	作業	作業	複習
3 高等演算法作業 3	2025-12-10 16:00	2025-12-10 17:00	作業	作業	複習
4 高等演算法作業 3	2025-12-10 17:00	2025-12-10 18:00	作業	作業	複習
5 高等演算法作業 3	2025-12-10 18:00	2025-12-10 19:00	作業	作業	複習
6 高等演算法作業 3	2025-12-10 19:00	2025-12-10 20:00	作業	作業	複習
7 高等演算法作業 3	2025-12-10 20:00	2025-12-10 21:00	作業	作業	複習
8 高等演算法作業 3	2025-12-11 13:00	2025-12-11 14:00	作業	作業	複習
9 高等演算法作業 2	2025-12-11 14:00	2025-12-11 15:00	作業	作業	複習
10 高等演算法作業 2	2025-12-11 15:00	2025-12-11 16:00	作業	作業	複習
11 高等演算法作業 2	2025-12-11 16:00	2025-12-11 17:00	作業	作業	複習
12 高等演算法作業 2	2025-12-11 17:00	2025-12-11 18:00	作業	作業	複習

匯入 Google Calendar

(或者其他支援.ics檔的日曆)

單一日曆下載

分成兩個日曆下載

方便分開匯入不同日曆，以顯示不同顏色