靜 宜 大 學

資訊工程學系

**畢 業 專 題 成 果 報 告 書**

專題名稱

Real-time Japanese Screen Translator Communication Project Plan

學 生：

資工四B 411170341 林家畯

指導教授：林耀鈴 (Yaw-Ling Lin) 教授

**西 元 二O二五 年 十二 月**

靜

宜

大

學

資

訊

工

程

學

系

專

題

題

目

Real-time Japanese Screen Translator Communication Project Plan

西

元

二

O

二

五

年

十

二

月

靜宜大學資訊工程學系

專題實作指導教師確認書

茲確認專題書面報告之格式及內容符合本系之規範

畢業專題實作名稱 : Real-time Japanese Screen Translator Communication Project Plan

畢業專題實作分組名單 : 共計 \_\_1\_\_ 人

|  |  |
| --- | --- |
| 組員姓名 | 學號 |
| 林家畯 | 411170341 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

指導教師簽章 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**西 元 2025 年 12 月 1 日**

靜宜大學資訊工程學系

專題實作授權同意書

本人具有著作財產權之論文全文資料，授予靜宜大學資工系，為學術研究之目的以各種方法重製，或為上述目的再授權他人以各種方法重製，不限地域與時間，惟每人以一份為限。授權內容均無須訂立讓與及授權契約書。依本授權之發行權為非專屬性發行權利。依本授權所為之收錄、重製、發行及學術研發利用均為無償。

指導教授 \_\_\_\_林耀鈴\_\_\_\_\_\_\_

|  |
| --- |
| 學生簽名: 學號:411170341 日期:西元2025 年12 月 1 日 |
| 學生簽名: 學號: 日期:西元 年 月 日 |
| 學生簽名: 學號: 日期:西元 年 月 日 |
| 學生簽名: 學號: 日期:西元 年 月 日 |
| 學生簽名: 學號: 日期:西元 年 月 日 |

指導教師簽章 \_\_\_\_\_\_\_\_\_\_\_\_\_\_

**西 元 2025 年 12 月 1 日**

XXXXXXXXXX即時本地多語言翻譯器XXXXXXXXX

指導教授：林耀鈴

學生：林家畯

靜宜大學資訊工程學系

# 摘要

在日常生活中，文字掃描與翻譯的需求愈加普遍，例如課堂拍攝的題目、紙本原文書或跨語言資料理解。然而，現有工具多依賴雲端翻譯 API，不僅受到用量與網路限制，更涉及隱私問題。為解決此困境，本專題開發一套可於 Windows 與 Linux 平台執行的本地化桌面應用程式，結合 OCR 與翻譯功能，兼顧便利性與安全性。

本研究主要使用 PaddleOCR v5 進行文字擷取，並整合 Meta NLLB-200 多語言翻譯模型，以支援超過兩百種語言。同時預留 API 模式，讓使用者可依需求切換至 DeepL、Google Translate 等線上翻譯服務。系統採模組化設計，分為 OCR 模組、翻譯模組與 GUI 模組；其中 GUI 以 PyQt6 實作，提供直覺操作，並整合歷史紀錄查詢與簡易除錯等功能。研究過程亦針對效能進行優化，例，以提升處理效率。

實驗結果顯示，本系統能在無網路環境下完成多語言文字辨識與翻譯，OCR 在中英文測試中的正確率達九成以上，翻譯品質亦獲得穩定表現。相較於僅依賴雲端 API 的工具，本系統具備 隱私保護、跨平台支援與靈活選擇 等優勢。整體而言，本專題不僅展現了人工智慧在多語言應用上的實用性，也為未來的功能延伸與教育研究應用奠定基礎。

# 誌謝

在本專題研究得以順利完成之際，首先要誠摯感謝指導教授 林耀鈴 老師在整個過程中給予的專業指導與寶貴建議。其次，感謝本系各位師長與同學在研究討論與資源提供上的協助，使本研究得以順利進行。最後，感謝家人一直以來的鼓勵與支持，讓我們能專心投入研究工作。謹此誌謝。

# 目錄

[摘要 i](#_Toc210060074)

[誌謝 ii](#_Toc210060075)

[目錄 iii](#_Toc210060076)

[第一章 緒論 - 1 -](#_Toc210060077)

[1-1專題動機： - 1 -](#_Toc210060078)

[1-2 專題目的： - 1 -](#_Toc210060079)

[1-3專題範圍以及限制： - 1 -](#_Toc210060080)

[第二章 專題內容與研究方法 - 3 -](#_Toc210060081)

[2-1系統需求分析： - 3 -](#_Toc210060082)

[2-2專題技術背景： - 3 -](#_Toc210060083)

[第三章 專題流程與架構 - 5 -](#_Toc210060084)

[3-1系統架構： - 5 -](#_Toc210060085)

[3-2系統資料流： - 5 -](#_Toc210060086)

[3-3 OCR原理及偵測： - 6 -](#_Toc210060087)

[3-4 NLLB Models 架構原理： - 7 -](#_Toc210060088)

[3-5 NLLB Models 訓練及優化： - 7 -](#_Toc210060089)

[3-6 NLLB Models優勢： - 8 -](#_Toc210060090)

[第四章 專題成果展示 - 9 -](#_Toc210060091)

[4-1 GUI 展示： - 9 -](#_Toc210060092)

[4-2 使用部署： - 11 -](#_Toc210060093)

[4-3 操作結果範例： - 12 -](#_Toc210060094)

[第五章 專案學習歷程 - 13 -](#_Toc210060095)

[5-1 專案開發初期： - 13 -](#_Toc210060096)

[5-2 專案中期開發日誌 - 28 -](#_Toc210060097)

[5-3 專題後期學習歷程 - 37 -](#_Toc210060098)

[第六章 結論與未來展望 - 42 -](#_Toc210060099)

[6-1 結論總結 - 42 -](#_Toc210060100)

[6-2 未來展望 - 43 -](#_Toc210060101)

[參考文獻 - 45 -](#_Toc210060102)

# 第一章 緒論

## 1-1專題動機：

隨著數位化時代的來臨，日常生活中對於文字掃描與即時翻譯的需求日益增加，例如課堂上拍攝的題目照片、紙本原文書籍的理解、以及跨語言資訊檢索等。然而，現有的解決方案多仰賴雲端翻譯 API，不僅受到免費用量限制與網路環境穩定度影響，更涉及使用者需將圖片上傳至伺服器的隱私疑慮。因此，本專題旨在開發一套可於 Windows 與 Linux 平台執行的本地化桌面應用程式，結合文字光學辨識（OCR）與多語言翻譯功能，以提供使用者兼具便利性與隱私性的完整解決方案。

本專題研製之基於Windows和Linux平台結合PaddleOCRv5以及Meta’s NLLB LLMs開發之桌面應用。可基於PaddleOCR捕獲螢幕畫面辨識文字並使用本地部署之語言模型或API串接線上翻譯工具進行翻譯，提供用戶多種根據硬體效能自訂義使用模型，並將OCR工具及語言模型效能優化實現效益最大化。提供多種獨立工具並使用基於PyQt6開發並貫徹User-friendly的圖形化操作介面整合使用戶可以一目瞭然，工具包括歷史喚回及查詢、簡易除錯等便利工具。

## 1-2 專題目的：

開發一款具有本地及API文字辨識及翻譯的應用，並使用GUI整合讓使用者一目瞭然，盡量簡化操作步驟以及對專業知識的需求，並能依照使用者硬體進行效能微調。

## 1-3專題範圍以及限制：

系統環境OS ：Windows / Linux

開發系統環境：Windows 11 (Laptop)

使用版本：python 3.12+

硬體要求：CPU

記憶體要求：取決於NLLB Models(最少16GB)

硬碟空間需求：取決於使用Models(最少4GB)

API Key：BYOK(支援DeepL、Gemini)

# 第二章 專題內容與研究方法

## 2-1系統需求分析：

截圖器、OCR處理、本地翻譯處理、紀錄logging、GUI、API處理、主程式

## 2-2專題技術背景：

PaddleOCRv5是一款基於RNN+CNN實現的光學字符識別系統，該方案聚焦餘多場景、多類型的文字識別。在文字類型方面，PP-OCRv5支援簡體中文、中文拼音、繁體中文、英文、日文5大主流文字類型，在場景方面，PP-OCRv5升級了中英複雜手寫體、豎排文本、生僻字等多種挑戰性場景的識別能力。在內部多情境複雜評估集上，PP-OCRv5較PP-OCRv4端對端提升13個百分點。

本專題使用模型包括：文本檢測模型、文本識別模型、圖像校正模型、文本行方向辨識模型、文檔方向辨識模型

NLLB-200（No Language Left Behind 200）是 Meta AI 於 2022 年提出的大規模多語言機器翻譯模型。

其核心基於 Transformer 架構，但在訓練與設計上做了專門優化：

* 多語言共享編碼器—解碼器架構：不同語言共享大部分的參數，以提升低資源語言的泛化能力。
* 語言識別 Token：在翻譯任務中於輸入端添加目標語言標記，引導模型輸出對應語言結果。
* 多層注意力機制（Multi-Head Attention）：支援長序列與跨語言語境建模，能處理多樣化的語法與詞彙。
* 型詞彙表與子詞單位（SentencePiece）：減少稀有詞問題，讓模型更好地處理黏著語與罕見字。

根據 Meta 的公開測試結果，其在 高資源語言(英文、中文、法文、德文、日文等)的翻譯品質接近甚至優於 Google Translate，而在低資源語言上也顯著超越以往模型，達到 BLEU 分數 44.3（平均提升 44%）。

本專題使用model為蒸餾版600M、1.3B及3.3B參數版

# 第三章 專題流程與架構

## 3-1系統架構：

|---main.py

|---MainGUI.py

|---ocr\_process.py

|---ocr\_paths.py

|---region\_capture.py

|---NLLB\_translator.py

|---logging\_utils.py

|---log\_viewer.py

|---history\_views.py

|---API\_mode.py

|---workers.py

|---requirements.txt

## 3-2系統資料流：

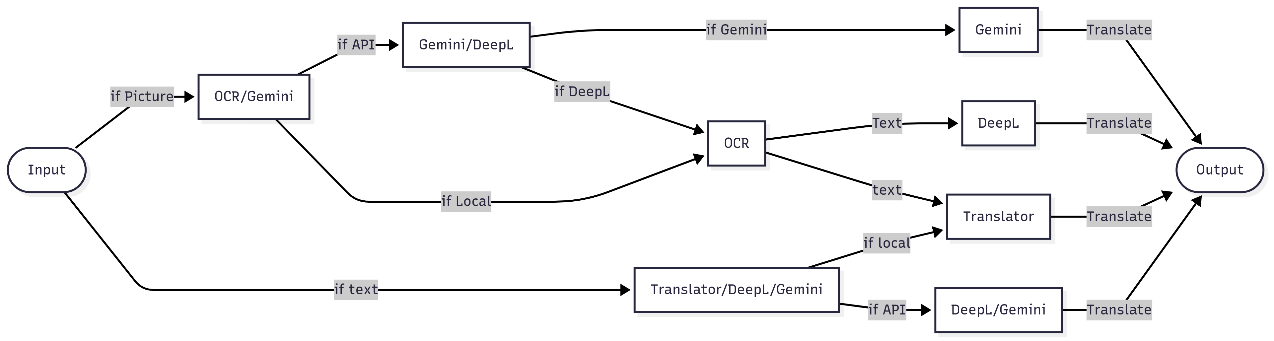


圖3-2、系統資料流

## 3-3 OCR原理及偵測：

1. 文字框偵測：

演算法首先透過 MobileNetV5 等輕量骨幹網路提取影像特徵，並由偵測頭輸出文字機率圖（probability map）與自適應閾值圖（threshold map）。接著利用 可微二值化（DB） 將連續機率轉換為二值化結果，生成候選文字區域。隨後進行後處理步驟：

* 1. 連通區域分析以擷取文字區域輪廓
  2. 最小外接矩形或多邊形近似生成文字框
  3. 外擴與非極大值抑制（NMS）以確保框能完整覆蓋文字且避免重疊。

此方法相較傳統固定閾值分割，能更精準貼合文字邊界，並保持高效率。

1. 文字方向校正：

偵測出的文字框經透視變換轉正為水平矩形，並由輕量分類器判斷是否需要旋轉或翻轉，以確保輸入辨識器的文字方向正確

1. 文字行分割：

為了將多個文字框組合成具可讀性的行，本系統採用簡單而有效的規則式演算法：

* 1. 依垂直位置與高度重疊率將文字框分群，形成同一行
  2. 在每行內再依語言書寫方向排序（如左至右、右至左或直排）
  3. 適用於表格或多欄時，可依間距或投影分析進一步切分。

1. 特點：

該演算法具備以下優點：

1. 高效能：透過模型剪枝與量化，能在一般 CPU/GPU 上運行。
2. 精準度高：DB 模型提供更準確的邊界偵測，提升 OCR 成功率。
3. 適應性強：支援多語言與多版面，能應對不同應用情境。

## 3-4 NLLB Models 架構原理：

NLLB 採用改良後的 Transformer 編碼器—解碼器架構，其主要設計如下：

* 1. 多語言共享參數：所有語言共用同一套模型參數，僅透過輸入端的「語言標記（Language Token）」指定目標語言，避免傳統多模型架構的冗餘。
  2. 子詞單位 (SentencePiece/BPE)：將不同語言的詞彙切割為子詞單位，有效減少稀有詞與字形變化帶來的困難。
  3. 大規模訓練語料：Meta 蒐集並清理了數百億對多語言平行語料，並特別增強低資源語言的資料，確保模型能學習更平衡的語言能力。
  4. 知識蒸餾 (Knowledge Distillation)：利用強大的教師模型指導學生模型訓練，提升翻譯品質並降低推理成本。

## 3-5 NLLB Models 訓練及優化：

1. 資料增強：包含回譯 (Back-Translation) 與過濾噪音資料，以增加低資源語言的有效訓練量。
2. 平衡取樣 (Balanced Sampling)：避免高資源語言在訓練中壓過低資源語言，確保翻譯品質的一致性。
3. 多階段訓練：先以大規模語料進行預訓練，再針對低資源語言做微調，達到最佳效果。

## 3-6 NLLB Models優勢：

1. 廣泛語言覆蓋：支援 200 種語言，遠超過傳統翻譯系統常見的數十種。
2. 低資源語言表現優異：透過跨語言參數共享與資料增強，顯著改善以往低資源語言翻譯效果不佳的問題。
3. 開源共享：Meta 公開完整模型權重（600M、1.3B、3.3B 參數版本）與訓練語料，使研究者與開發者能自由應用與部署。

# 第四章 專題成果展示

## 4-1 GUI 展示：

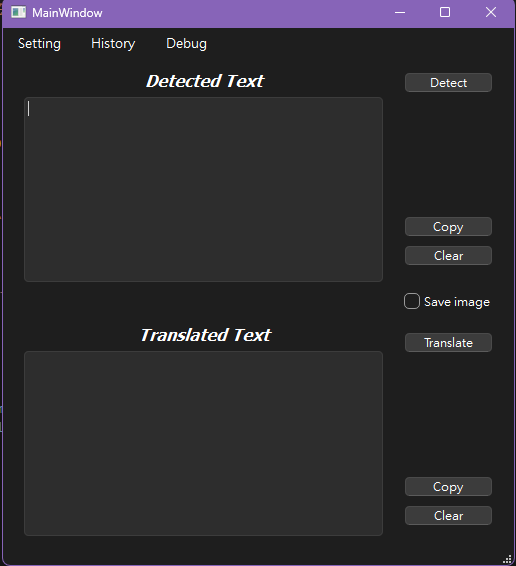


圖4-1-1、主介面

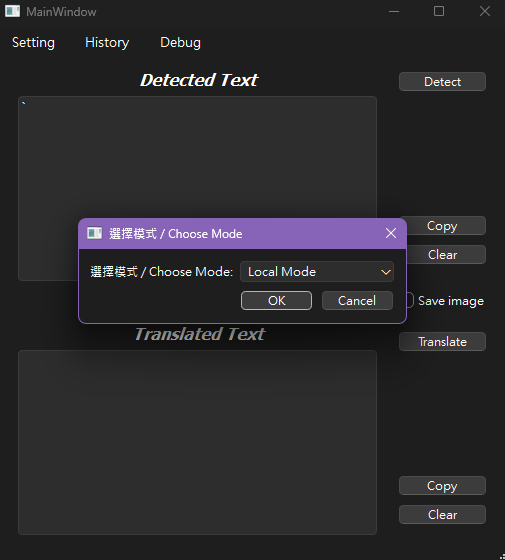


圖4-1-2、模式選擇(本地)

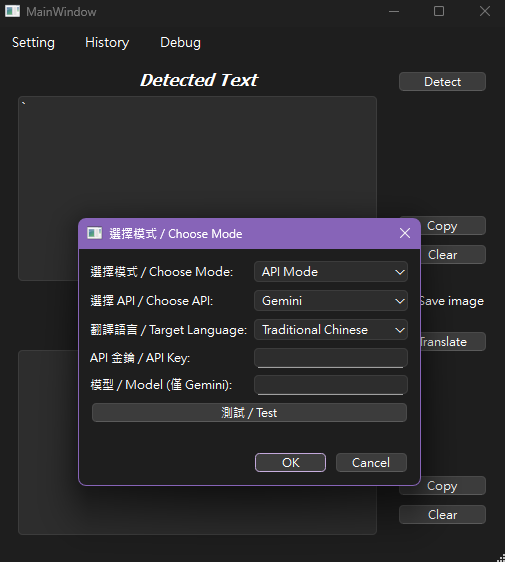


圖4-1-3、模式選擇(API)

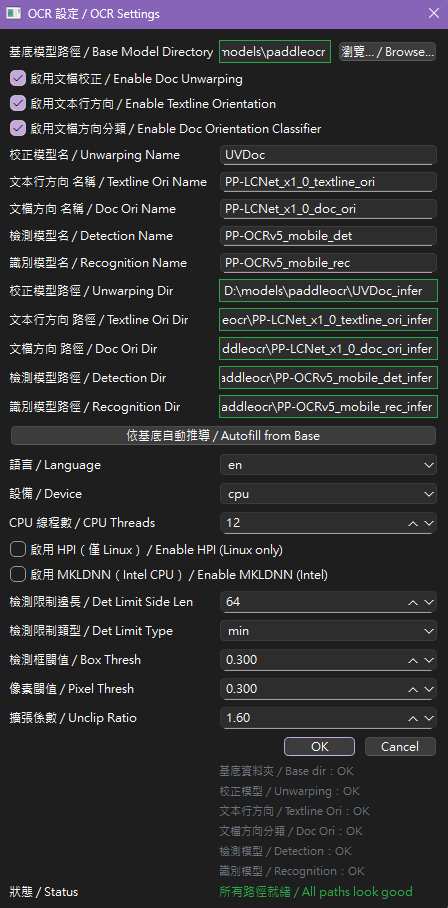


圖4-1-4、OCR設定

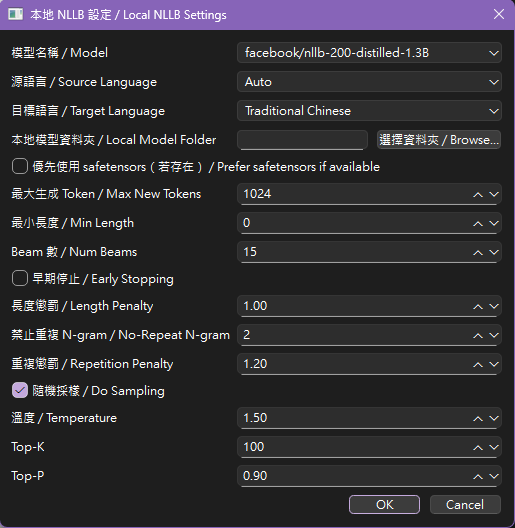


圖4-1-5、Translator設定

## 4-2 使用部署：

* 1. 從github clone專案：<https://github.com/NitanCos/local_translator.git>
  2. 使用conda 建立虛擬環境：conda create –name Translator python==3.12
  3. 啟動環境並執行依賴庫安裝：pip install -r requirements.txt
  4. 下載NLLB Models(600M、1.3B、3.3B)
  5. 下載PPOCRv5 models
  6. 啟動main.py：python main.py

## 4-3 操作結果範例：

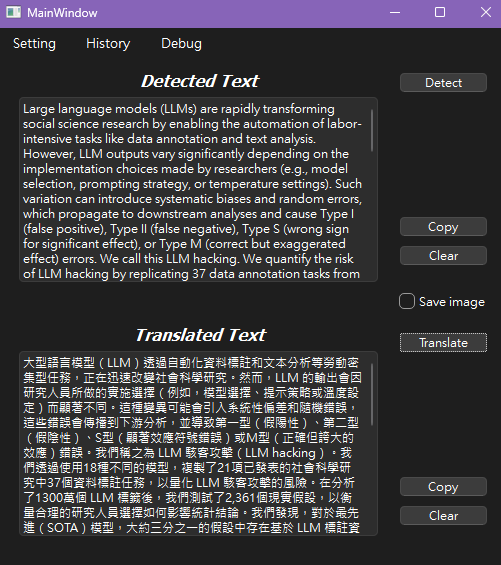


圖4-3-1、操作範例(1)

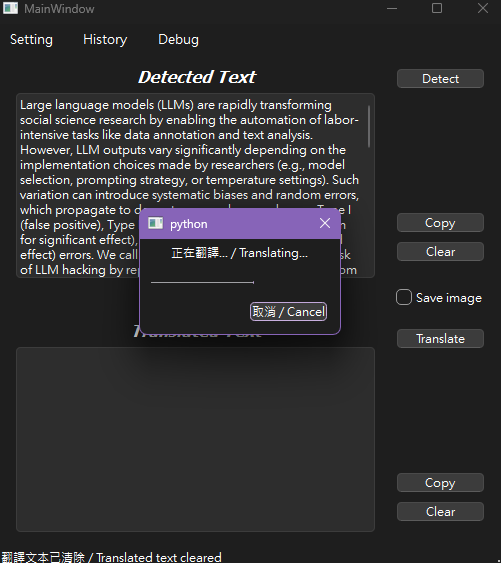


圖4-3-2、操作範例(2)

# 第五章 專案學習歷程

## 5-1 專案開發初期：

1. 最初發想：

最一開始的發想是源自於我在遊玩的一款知名的線上角色扮演RPG遊戲──《Final Fantasy XIV》。《Final Fantasy XIV》（簡稱 FFXIV）是由 Square Enix 開發與營運的大型多人線上角色扮演遊戲（MMORPG）。作為《Final Fantasy》系列中持續營運的線上作品，它自 2013 年推出《重生之境》後便廣受好評，並透過多部資料片不斷擴展世界觀與玩法，包括《蒼穹之禁城》、《紅蓮之狂潮》、《漆黑的反叛者》、《曉月的終焉》以及最新的《黃金的遺產》。



圖5-1-1 Final Fantasy XIV 7.0宣傳照

此遊戲的一大特色在於劇情導向的設計。不同於許多 MMORPG 僅將劇情作為背景，FFXIV 的故事結構與演出品質堪比單機 RPG，被眾多玩家譽為系列中最優秀的劇情之一。同時，角色職業系統具備高度自由性，玩家在同一角色中即可切換不同職業，無須重新建立新角色，這大幅降低了遊戲嘗試成本，並提升了玩法的多樣性。



圖5-1-2、FFXIV武士

不過，最吸引我的莫過於戰鬥及副本設計，FFXIV 採取「舞步型」戰鬥節奏，強調站位與機制解謎。各種試煉戰與團隊副本要求玩家精準掌握技能循環與場地變化，稍有失誤便可能導致整體挑戰失敗，因此為保證在野排體驗(即和不認識的人招募組隊)，需要玩家隊伍一起腦力激盪，在一次次嘗試即溝通後，不斷調整隊伍配置及職業選擇，以求完美通關。同時，這個遊戲也有豐富的社交系統，玩家甚至可以說是在裡面開創自己的第二人生，開店、投資、成為導師、從事生產等，這些豐富的系統使玩家可以暫時時拋開生活上的重擔，在虛擬世界中遨遊。

然而，這個對我來說是心靈寄託的寶藏遊戲有一個致命的缺點──他並沒有中文代理商，也就是說，我在裡面的一切必須都以外語溝通，再者，因為亞洲主要伺服器都位於日本，但日本人天生又有些排外，所以就算我能以英文介面勉強觀看遊戲劇情及介面，但我完全在攻略副本時與他人溝通，這對我來說一度造成非常大的困擾，雖然近年來AI蓬勃發展，語言翻譯已經不成問題，但仍沒有解決問題，因為幾乎所有的AI都需要付費，即使有免費額度，但也是一下就用完了，根本無法從頭解決我的根本問題，所以我就開始思考，由沒有什麼方法，可以滿足我的需求的同時又可以不花一毛錢解決問題，所以此專案──即時翻譯器便有了最初的樣子。

* 1. 專題規劃：

既然有了想法就是要付諸行動，但我這個剛轉進資工系的半桶水，幾乎什麼都不會，甚至連程式也只會最基本的運算使用，又要怎麼手搓出一個完整的系統應用呢？

於是我決定從基礎開始做，雖然在過去的大學課程中，我已經學過 C 與 C++，甚至稍微接觸過 Java，但 Python 給我的第一印象卻完全不同。幸好現在網路非常發達，網路上非常多的python教學，於是我選擇一部長達50小時的python 初學者教學開始看起。



圖5-1-3、python 學習影片

在 C++ 裡，輸出一句「Hello World」需要包含標頭檔、宣告 main() 函式，最後再搭配 cout 或 printf；而 Python 只要一行 print("Hello World") 就能完成。這樣的差異讓我驚訝之餘，也感受到一種前所未有的「親民感」。然而，這種簡潔並不代表容易上手。一開始，我經常因為「縮排」問題而卡關：少打一個空白，或是不小心混用了 tab 與空格，程式就會報錯。這在 C++ 的世界裡幾乎不會出現，因此我花了不少時間去重新培養「對縮排極度敏感」的習慣。

剛開始的學習，主要是跟著線上教學平台一步步操作：從變數型別、條件判斷、迴圈到函式撰寫，慢慢建立起對 Python 的基礎認知。雖然語法相對簡單，但我仍常常陷入「以 C++ 的習慣寫 Python」的陷阱，例如總想先宣告變數型別，或嘗試用分號結尾。這些「錯誤的移植習慣」花了不少時間才被糾正。

隨著練習累積，我漸漸熟悉 Python 的語感。舉例來說，我開始體會到 Python 的「一切皆物件」設計哲學：整數、字串、甚至函式都能被視為物件並操作，這讓我對程式語言的彈性有了新的認識。當我第一次把一個函式當成參數傳給另一個函式時，那種驚喜感到現在仍印象深刻。

在完成基礎語法的學習後，我進一步挑戰 Python 的模組化與套件生態系。這時候我才真正理解到 Python 為什麼能在 AI 領域如此流行。

我最先接觸的，是 numpy 與 pandas。這兩個套件是資料處理的基礎，幾乎所有後續的 AI 開發與資料科學應用都繞不開它們。numpy 讓我第一次體會到「矩陣運算」能被程式語言如此直觀地表達。以前在數學課上看到的 𝐴 × 𝐵 矩陣相乘，總是抽象又枯燥，而現在只需要一行 np.dot(A, B) 就能完成。當我看到結果正確輸出時，彷彿數學課本真的活了過來。

接著是 pandas，它則讓我學會如何處理表格型資料。最初我只用它來讀取 CSV 檔，但後來我逐漸學會了篩選、排序、群組等操作。這對後續專題中進行翻譯測試資料的分析有很大幫助。例如，我能快速統計某些翻譯任務的正確率，或比較不同模型在不同語言上的差異。

除此之外，我還學會了 matplotlib 與 seaborn，用來視覺化數據。剛開始只是簡單畫出折線圖與長條圖，但隨著需求增加，我開始嘗試自訂樣式、加上標籤與圖例。這不僅提升了報告的可讀性，也讓我在 debug 過程中能快速掌握模型表現。

更重要的是，這些工具讓我建立了一種「模組化解決問題」的思維方式。每當遇到新問題，我會先想：「是否已有現成套件能幫我處理？」這與我過去在 C++ 世界裡「什麼都得自己寫」的心態截然不同。Python 的生態，讓我第一次感覺自己可以站在巨人的肩膀上，而不是從零開始。

* 1. 理解 OCR：從概念到實作：

當 Python 的學習逐漸穩固後，我便開始探索專題的核心之一：OCR（光學文字辨識）。在最初的想像裡，OCR 就像是一個「黑盒子」：丟一張圖片進去，它就會吐出一段文字。但實際研究後，我才知道 OCR 其實是一個包含多個子任務的複雜流程。

OCR 的典型流程包含四個步驟：

* 1. 文字框偵測：從圖片中找出可能包含文字的區域。
  2. 文字行分割：將長段落拆分成一行一行文字，避免辨識器混淆。
  3. 字元辨識：透過卷積神經網路或序列模型，將影像轉換為文字。
  4. 後處理：利用語言模型或規則修正辨識錯誤，例如「1」與「l」的區分。

在挑選 OCR 工具時，我主要比較了 Tesseract 與 PaddleOCR。Tesseract 雖然歷史悠久，支援語言也廣，但它的準確率在複雜場景下常不理想，特別是對中日韓文字。而 PaddleOCR 則基於深度學習，結合 PP-OCR 系列模型，在多語言場景下有明顯優勢，並且持續更新到 PP-OCRv5。



圖5-1-4、PaddleOCR

最打動我的，是 PaddleOCR 的「輕量化」與「高效能」設計。它能在普通的電腦上流暢運行，同時提供良好的準確率。更重要的是，它的開源社群活躍，有豐富的範例程式碼與模型可直接套用。這讓我不必耗費過多時間在模型訓練，而能把精力集中在整合與應用上。

* 1. 邁向 Transformer：理解 LLM 的核心：

在 OCR 部分逐漸成形後，我開始專注於專題的另一個核心──翻譯。這就必須理解 Transformer 架構，因為它已經成為現代自然語言處理的基石。

一開始閱讀 Transformer 論文（《Attention Is All You Need》）時，滿滿的公式讓我感到壓力。Self-Attention、Multi-Head Attention、Position Encoding，這些名詞看似陌生，實際上卻蘊含著強大的概念。



圖5-1-5、Attention is all you need

Transformer 的突破點在於 注意力機制 (Attention)。過去的 RNN 與 LSTM，雖然能處理序列，但在長距離依賴上表現不佳。而 Attention 允許模型「同時考慮輸入序列中的所有位置」，從而有效捕捉上下文。舉例來說，在翻譯一句中文長句時，模型能同時參考前後文，而不會因為序列過長而「遺忘」。

我透過 Hugging Face 的 transformers 套件，實際操作過幾個翻譯模型，例如 MarianMT 與 M2M-100。雖然在初期實驗中，我的電腦常因為顯存不足而跑不動大模型，但這些失敗的經驗也讓我更清楚「資源限制」與「模型選擇」之間的關係。

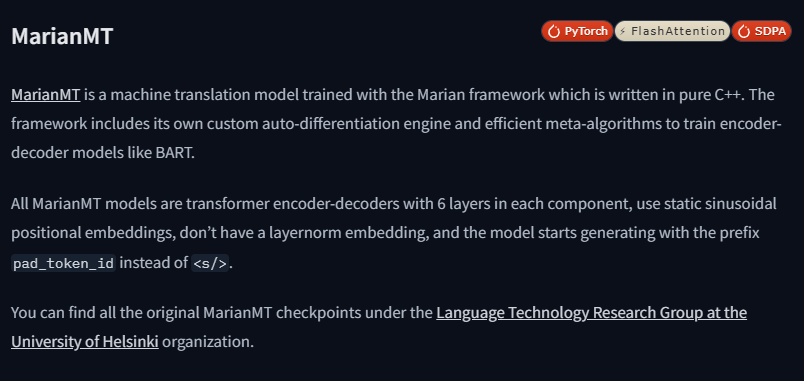


圖5-1-6、MarianMT

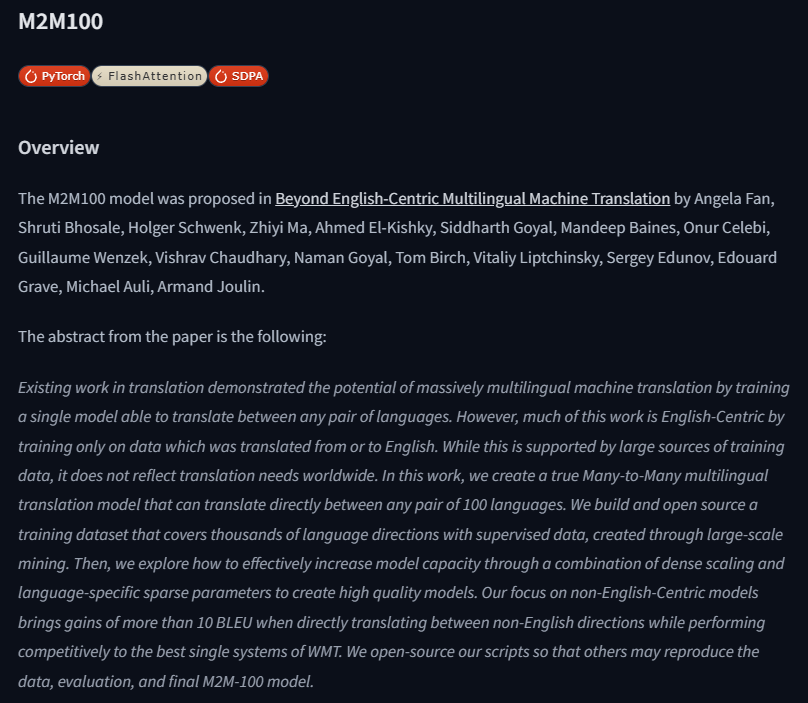


圖5-1-7 M2M-100

* 1. 語言模型的選擇基準：

在專題設計時，我必須面對現實的一個問題：專題不可能要求所有使用者都有高階 GPU。因此，在挑選語言模型時，我制定了幾個基準：

* 1. 能在 CPU 上運行：即使速度慢一點，也必須保證最低需求能執行。
  2. 滿足日常翻譯需求：至少能應付一般對話與文件翻譯，而非只適合學術用途。
  3. 語言覆蓋度廣：支援中文、英文、日文等常見語言，同時兼顧部分低資源語言。
  4. 開源可用：避免 API 金額限制，確保可長期使用與維護。

基於這些基準，我最初測試過 MarianMT 與 M2M-100，但它們在低資源語言上的表現不佳。最終，我選擇了 Meta 提出的 NLLB-200。它的優點在於提供多個規模的版本，例如 600M 可在 CPU 運行，而 3.3B 則能在 GPU 上提供更高翻譯品質。這種靈活性，正好符合專題「可依硬體資源自訂」的設計理念。

* 1. GUI 開發的整合與學習：

最後一個重要的學習歷程，就是 PyQt6 的 GUI 開發。

剛開始我對 GUI 的理解很單純，以為只要做出一個「輸入 → 輸出」的視窗就好。但實際操作後才發現，GUI 的挑戰不在於「能不能跑」，而在於「使用者體驗」。

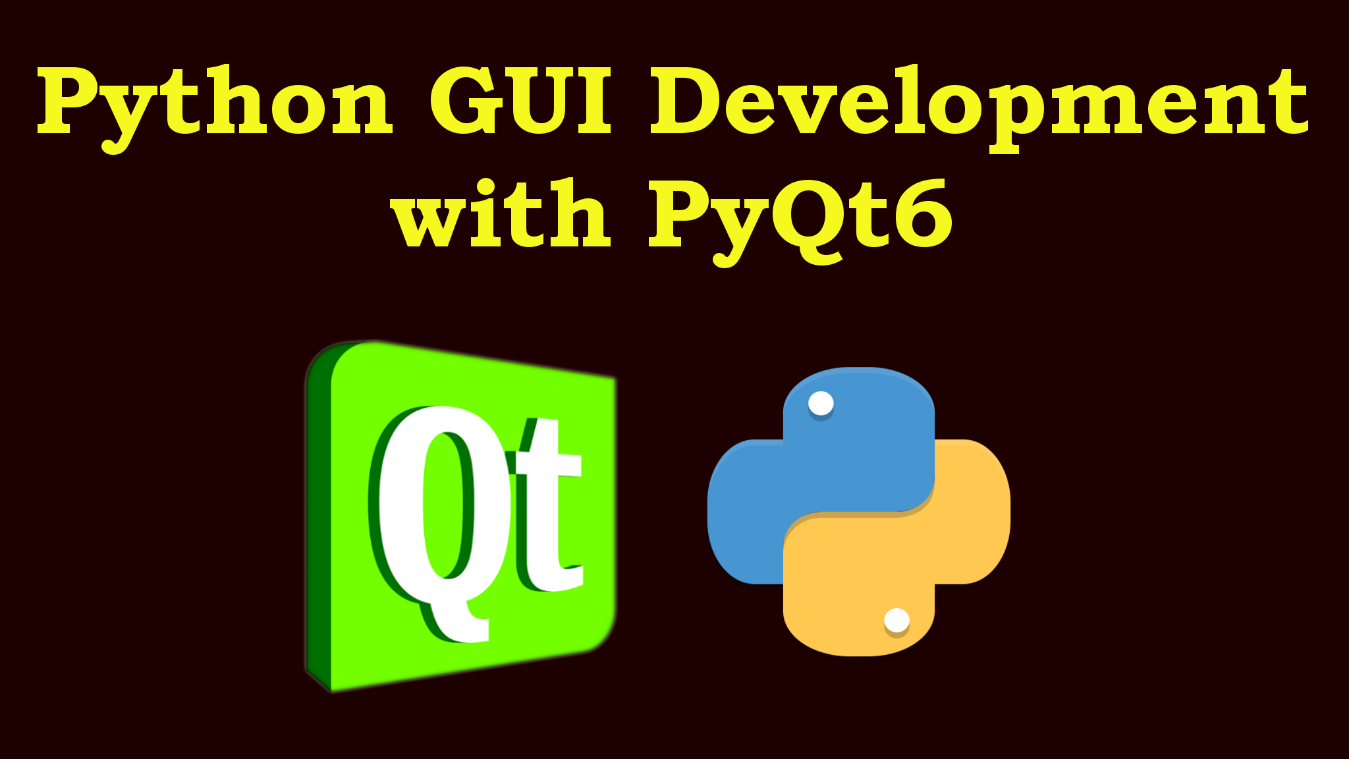


圖5-1-8、PyQt6 with Qtdesigner

我從最簡單的範例開始：建立一個主視窗，加入按鈕與文字框，實現「輸入文字 → 點擊翻譯 → 輸出結果」的流程。但隨著功能需求增加，我開始面臨更多挑戰：

* + - 1. 模式切換：如何在同一個介面中提供「API 模式」與「本地模式」？
      2. 歷史紀錄：如何讓使用者能查詢之前的翻譯？
      3. 錯誤提示：如何在翻譯失敗時提供明確訊息，而不是讓程式崩潰？

這些問題迫使我深入學習 事件驅動程式設計。我逐漸理解，GUI 不僅是前端顯示，它與後端邏輯的結合方式，決定了使用者是否覺得「順手」。

學習 PyQt6 的過程中，我也嘗試過不同設計模式，例如用 QThread 處理背景任務，避免主介面卡死。這讓我更清楚認識到「軟體工程」與「AI 技術」的融合點——一個專題不僅要有模型，還要有良好的使用介面。

* 1. 初期構思的總結：

回顧整個初期構思階段，從完全不懂 Python，到能熟練運用套件與框架；從單純理解 OCR 是「圖片轉文字」，到掌握其偵測與分割的細節；從對 Transformer 的敬畏，到能實際操作 NLLB-200；再到最後設計 GUI 整合系統，這一系列學習歷程不僅是技術的提升，更是心態的轉變。

一開始的我，只是想「做一個翻譯工具」。但隨著學習深入，我開始思考：這個工具如何被真正使用？如何兼顧效能與可及性？如何在有限的硬體資源下，仍提供有價值的功能？

這些問題，構成了我專題的核心方向，也讓「初期構思」不再只是技術清單，而是一段真正的學習旅程。

## 5-2 專案中期開發日誌

* 第 1 週：OCR 與 Python 環境測試：
  + 1. 目標

安裝並測試 PaddleOCR，驗證能否在 Windows 與 Linux 環境正常運作。

* + 1. 過程

我先在 Windows 上建立一個 Conda 環境，安裝了 paddlepaddle==3.0.0 與 paddleocr。第一次嘗試時，因為 CUDA 與 Paddle 版本不匹配導致錯誤。後來參考官方文件，對應到正確的 CUDA 版本才成功執行。

* + 1. 問題與 Debug

錯誤訊息：OSError: (xxx) CUBLAS not found

解決方法：重新安裝對應 CUDA 的 paddlepaddle wheel，並測試 CPU 版本確認基礎功能。

* + 1. 心得反思

第一次跑成功後看到圖片中的中文字被正確辨識出來，那一刻讓我感覺到專題是「真的有可能落地」的。我也深刻體會到環境安裝與版本管理的重要性。

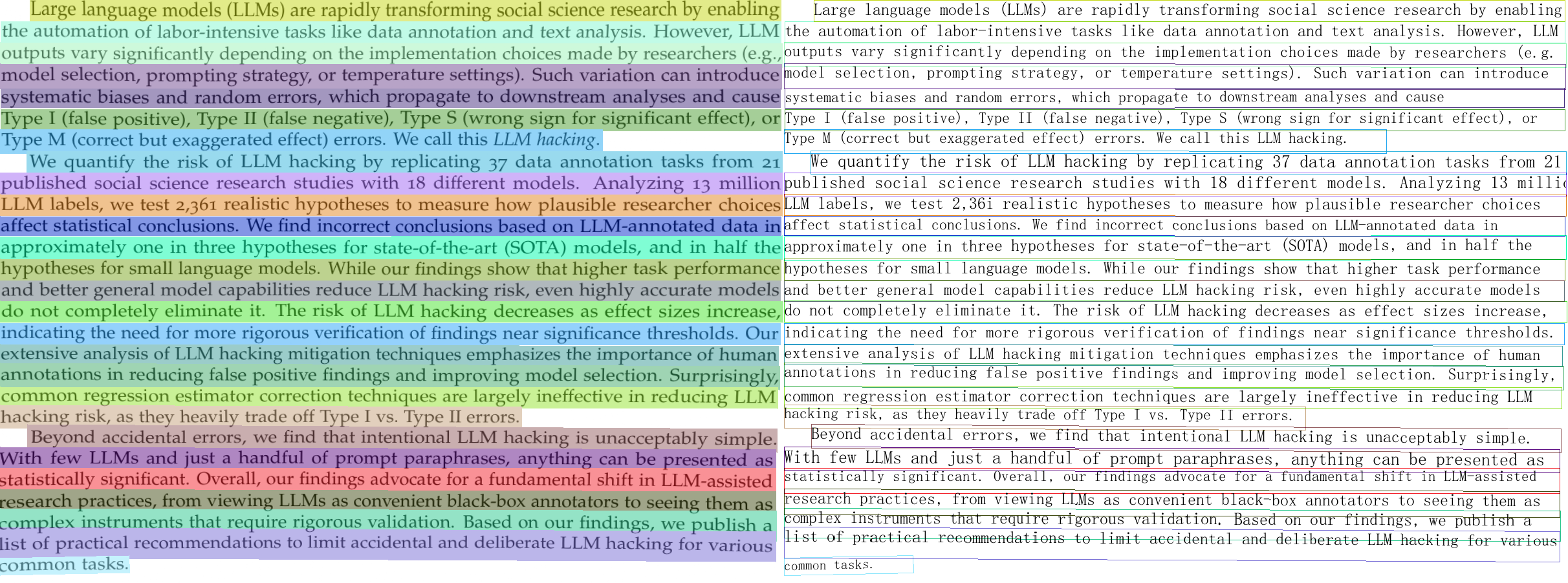
****

圖5-2-1、OCR辨識範例

* 第 2 週：OCR 精度測試與字體挑戰：
  + - 1. 目標

測試不同語言與字體的辨識效果，特別是中英文混排。

* + - 1. 過程

我找了幾張試題照片與 PDF 截圖，丟進 PaddleOCR 測試。英文部分辨識率高達 95% 以上，但中文遇到手寫字或模糊字體時錯誤率偏高。

* + - 1. 問題與 Debug
      * 遇到「1」與「l」、「0」與「O」混淆。
      * 部分圖片傾斜後，文字框偵測不精準。
      * 嘗試使用 PaddleOCR 的角度分類器（direction classifier）修正，效果改善。
      1. 心得反思

OCR 的難點在於「資料品質」遠比演算法本身更重要。這讓我開始思考後處理是否需要結合字典或語言模型。

* 第 3 週：初步翻譯模組測試：
  1. 目標

使用 Hugging Face 的 transformers 套件測試翻譯模型（MarianMT、M2M-100）。

* 1. 過程

安裝 transformers 後，載入 Helsinki-NLP/opus-mt-en-zh，嘗試將英文翻譯成中文。

* 1. 問題與 Debug
* CPU 運行速度太慢：翻譯一段 30 字的句子需 10 秒。
* 顯存不足：載入 M2M-100-1.2B 模型時直接報 CUDA out of memory。
* 嘗試換小模型（418M 參數），在 CPU 上勉強能跑。
  1. 心得反思

這一週我第一次體驗到「模型規模與硬體限制」的落差。這也讓我確立了專題一個重要原則：要有 CPU 可行的選項，否則實用性不足。

* 第 4 週：NLLB-200 初體驗：

1. 目標

下載並測試 NLLB-200 模型，驗證翻譯品質與效能。

1. 過程

從 Hugging Face 下載了 facebook/nllb-200-distilled-600M，並測試中 → 英翻譯。結果相當不錯，翻譯較自然，不像 MarianMT 那樣直譯，但對於長文還是心有餘而力不足。

1. 問題與 Debug

* 模型檔案龐大（數 GB），下載時間長。
* torch 推理時記憶體使用率偏高，在筆電上 CPU 模式速度偏慢。
* 嘗試用 batch size=1 減少資源消耗。

1. 心得反思

NLLB 的表現讓我覺得專題方向更加明確，因為它同時支援多語言且品質尚可。未來就以 NLLB 作為核心翻譯模型。

* 第 5 週：OCR 與翻譯的串接：

1. 目標

將 PaddleOCR 輸出的文字送入翻譯模型，完成「圖片 → 翻譯文字」的基本流程。

1. 過程

我寫了一個 Python 腳本：先用 paddleocr 擷取文字，再用 Hugging Face 的 pipeline 翻譯。成功把一張 PDF 截圖轉換成中文譯文。

1. 問題與Debug

* OCR 輸出的字元包含雜訊（如多餘的空格、換行）。
* 翻譯結果有時句子斷裂，因為 OCR 拆段方式不合理。
* 新增簡單的正則表達式清理文字，效果改善。

1. 心得反思

第一次跑通整個流程的成就感非常大。這也讓我更清楚看到「模組化」的重要性：OCR、翻譯、清理，必須獨立實作才能靈活組合。

* 第 6 週：PyQt6 初探：

1. 目標

開始設計 GUI，至少能實現「擷取圖片 → 顯示翻譯結果」。

1. 過程

學習 PyQt6，建立一個基本視窗。加上一個按鈕，能選取圖片並顯示 OCR+翻譯結果。

1. 問題與Debug

* GUI 與翻譯任務在同一執行緒，導致操作時視窗卡死。
* 嘗試使用 QThread，但不熟悉信號與槽（signal-slot）機制，導致程式不穩定。

1. 心得反思

這次學到 GUI 與後端邏輯需要分工。要真正做出好用的應用程式，必須學習並發揮多執行緒的特性。

* 第 7 週：多語言測試：

1. 目標

測試系統在不同語言下的表現，包括中、英、日、韓。

1. 過程

OCR 測試日文與韓文，結果辨識率還可以，但在字體模糊情況下錯誤率明顯上升。翻譯部分，NLLB 的日文 → 中文結果相當自然。

1. 問題與Debug

* PaddleOCR 對韓文字體支援不夠完美，有時會漏字。
* NLLB 翻譯時，長句容易被拆成片段。
* 嘗試將 OCR 結果合併後再翻譯，改善斷句問題。

1. 心得反思

這一週讓我更確認：專題的價值在於「多語言整合」，而不是單純的 OCR 或翻譯。

* 第 8 週：API 模式串接：

1. 目標

除了本地模型，也要能選擇使用 API（DeepL、Gemini）翻譯。

1. 過程

在 GUI 中新增選單，讓使用者選擇「API 模式」或「本地模式」。API 模式透過網路請求回傳翻譯結果。

1. 問題與Debug

* API Key 管理麻煩，初期常忘記設定環境變數。
* 網路不穩定時，GUI 會卡住，後來加上錯誤處理訊息框。

1. 心得反思

這讓系統更靈活：使用者可以依硬體資源決定翻譯方式。

* 第 9 週：歷史紀錄與查詢功能：

1. 目標

新增翻譯歷史紀錄，方便使用者回顧。

1. 過程

建立 JSON檔案，存放翻譯記錄。GUI 增加「歷史查詢」視窗。

1. 問題與Debug

* Unicode 儲存問題：日文與韓文存入檔案後顯示亂碼。
* 解決方式：確保檔案編碼為 UTF-8，問題解決。

1. 心得反思

第一次把JSON整合進 GUI，感受到「軟體工程」的完整性，而不只是單純跑模型。

* 第 10 週：整合測試與效能優化：

1. 目標

把各模組（OCR、翻譯、GUI、API、歷史紀錄）整合，並進行效能測試。

1. 過程
2. 測試不同規模的 NLLB 模型（600M vs 3.3B）。
3. 測試 CPU 與 GPU 的延遲差異。
4. 嘗試用 batch 翻譯與快取機制加速。
5. 問題與Debug

* CPU 模式下翻譯一頁文字仍需十數秒。
* 嘗試分段翻譯（每 100 字拆分一次），減少單次延遲。

1. 心得反思

專題雖然還不完美，但已經能在不同平台跑通完整流程。效能優化是長期課題，但系統的實用性已經初步建立。

* 中期開發總結：

這十週的中期開發歷程，從「跑通最小可行產品」到「整合多模組並優化效能」，不僅是技術挑戰，更是心態調整的過程。我逐漸從單純「會寫程式」轉變為「懂得規劃模組與體驗」。雖然仍有許多瓶頸，但正是這些不斷的 Debug 與學習，讓我逐漸成長為一個能獨立開發完整系統的學生。

## 5-3 專題後期學習歷程

* 第 11 週：壓力測試的開始：

1. 目標

模擬正式展示的環境，進行整體壓力測試，檢查軟體在長時間運行下的穩定性。

1. 過程

我準備了十多張不同來源的測試圖片，包括：

* 教科書拍照圖片
* PDF 截圖
* 手寫筆記掃描檔
* 中英文混雜的試卷

在測試過程中，我不僅確認 OCR 是否能正確辨識，也觀察翻譯模組在不同情境下的延遲與準確度。特別是我刻意讓程式連續運行數小時，以模擬展示時可能遇到的高強度操作。

1. 問題與Debug

* 記憶體累積：發現長時間翻譯會造成記憶體逐漸上升，懷疑是物件未正確釋放。
* GUI 卡頓：在快速連續操作時，GUI 偶爾會凍結。檢查後發現是多執行緒未正確回收造成。
* 翻譯延遲：大段文字翻譯仍需十多秒，不利於即時展示。

我嘗試增加記憶體回收機制，並將翻譯任務強制放入背景執行緒，讓 GUI 不受影響。雖然不能完全消除延遲，但至少確保介面「不會假死」。

1. 心得反思

這一週讓我體會到「展示」與「實驗」的差異。平常自己測試，只要程式能跑出結果就算成功，但在正式場合，任何延遲或凍結都可能導致現場冷場。從那一刻起，我開始強調「穩定性」比「性能極限」更重要。

* 第 12 週：模擬上台操作流程：

1. 目標

設計一個完整的展示流程，確保操作流暢且能在短時間內展現系統功能。

1. 過程

我安排了以下展示步驟：

1. 開啟 GUI，載入圖片。
2. 展示 OCR 辨識結果。
3. 切換至翻譯模式，顯示即時翻譯結果。
4. 展示歷史紀錄功能，呼叫之前翻譯過的句子。
5. 切換「API 模式」與「本地模式」，比較速度與品質。

這樣的設計能在五分鐘內清楚呈現專題的完整功能。

1. 問題與Debug

翻譯時間不穩，有時一張圖 20 秒，有時超過 100 秒。

API 模式在網路延遲高的環境下，表現甚至比本地模型更差。

1. 心得反思

我意識到「展示素材」的選擇至關重要。若展示時選到一張需要很久才能翻譯的圖片，現場就會陷入尷尬。因此，我開始整理一份「專用展示資料集」，確保能展現系統優勢。

* 第 13 週：使用者回饋（內部測試）：

1. 目標

邀請同學與朋友試用系統，收集回饋意見。

1. 過程

我將系統安裝在三台不同環境的電腦上，讓同學實際操作。觀察他們的使用行為，並收集他們的評論。

1. 問題與Debug

* 正面：介面設計直覺、OCR 較準確、翻譯結果自然。
* 負面：
* 翻譯延遲明顯，耐心較差的使用者會覺得「卡」。
* GUI 字體過小，對非資訊背景的人不太友善。
* API Key 設定麻煩，不懂技術的人不知道該放哪裡。

我針對字體大小做了自動調整，讓不同解析度下都能清楚顯示。

增加「API Key 設定頁面」，避免使用者必須修改程式碼。

1. 心得反思

開發者常常覺得「能跑就好」，但實際使用者的需求卻截然不同。他們更關心的是「是否方便」與「是否快速」。這次測試讓我從開發者心態，轉向「使用者導向」的思維。

* 第 14 週：使用者回饋（二次測試）：

1. 目標

針對上次的問題做修改後，再次邀請測試。

1. 過程

這次的測試結果明顯改善：字體問題解決後，使用者覺得介面更友好；API Key 設定頁面也降低了門檻。然而，翻譯延遲仍然是最大的批評點。

1. 問題與Debug

我嘗試將翻譯過程進行切段處理，例如將一篇 500 字文章拆成數段，每段翻譯後再拼接。雖然有時句子連貫性略差，但能顯著縮短等待時間。

1. 心得反思

這讓我理解「技術折衷」的重要性。理論上完整翻譯一整段最精準，但在實際應用中，使用者更在乎「速度」。因此，提供一個折衷方案（快速但可能稍降品質），比堅持理想更務實。

* 第 15 週：最後修正與展示準備：

1. 目標

在上台前完成最後的調整，並模擬多次展示。

1. 過程

* 整理展示資料集，確保每個例子都能快速且正確展示功能。
* 增加「翻譯進度條」，即便需要等待，使用者至少能看到系統正在處理。
* 試在不同網路環境下的 API 模式，確保不會意外斷線。

1. 問題與Debug

* 發現進度條在多執行緒下無法即時刷新，後來透過 signal-slot 機制解決。
* 檔案過大時會造成查詢延遲，臨時解法是將展示用的設定筆數上限為小型版本。

1. 心得反思

準備展示的過程，就像是在把系統「包裝」成一個成品。雖然底層還有許多不足，但只要能在台上順暢運行，就是階段性的成功。

* 最後心得與反思：

回顧整個後期歷程，我學到的不僅是技術上的知識，更是「產品化思維」的重要性：

1. 壓力測試：讓我意識到軟體必須能在「不理想環境」下仍能運作，這比最佳效能更關鍵。
2. 使用者回饋：讓我學會傾聽與改進，開發者覺得不重要的細節，往往正是使用者最關心的。
3. 最後修正：逼迫我在有限時間內做出取捨，例如「速度 vs 品質」，這是軟體工程永恆的課題。
4. 展示經驗：不只是交付一個程式，而是交付一個「能被理解與接受的成果」。

專題從初期的構思，到中期的開發，再到後期的展示與反思，我感覺自己已經不再只是「寫程式的學生」，而是逐漸理解「如何把技術轉化為產品」的工程師。

# 第六章 結論與未來展望

## 6-1 結論總結

本專題從日常翻譯需求出發，針對現有工具在隱私性、網路依賴及 API 限制上的不足，開發出一套結合 OCR 與多語言翻譯 的跨平台桌面應用程式。系統以 PaddleOCR 為基礎進行文字擷取，並整合 Meta NLLB-200 模型 以及 API 模式，提供使用者「離線翻譯」與「線上翻譯」的彈性選擇；同時以 PyQt6 實作 GUI，使操作更直覺。

經過測試，系統在中英文 OCR 上的準確率達九成以上，翻譯品質穩定，能在無網路環境下提供有效的翻譯服務，並支援多語言使用情境。相較於純 API 工具，本系統展現了 隱私保護、跨平台支援、模型可自訂 等優勢。

專題歷程中，從 Python 基礎學習、OCR 原理理解、Transformer 架構探索，到 GUI 介面整合，過程不僅提升了技術能力，更培養了我在系統設計、錯誤排查與使用者導向思維上的實踐經驗。

綜合而言，本專題完成了最初設計目標，建立出一個可實際應用、能兼顧學術與實務價值的翻譯工具。

## 6-2 未來展望

雖然系統已能運行並滿足核心需求，但仍有許多可延伸與優化的方向：

1. 跨平台支援：部署至 Mac OS

目前系統已在 Windows 與 Linux 上驗證，未來將進一步優化相依套件與打包方式，使其能順利部署於 Mac OS，讓更多使用者能在多樣化的開發環境中使用本系統。

1. 翻譯模型調用自由度

現階段系統內建 PaddleOCR 與 NLLB 模型，但未來可提供 更高的模型替換彈性。使用者可透過簡單操作（如 GUI 下拉選單或設定檔編輯），自由切換 Hugging Face 上的不同翻譯模型，甚至輸入自訂超參數（如 batch size、beam size），以符合各自的應用場景。

1. 功能延伸與多樣應用

* 加入 即時字幕翻譯 功能，應用於線上會議或影片觀看。
* 增設 快捷鍵操作，提升實際使用便利性。
* 與 雲端協作平台（如 Notion、Google Docs）整合，方便翻譯結果直接應用於工作流程。
* 透過上述優化與擴充，本系統將能進一步提升實用性與靈活性，從單純的 OCR 翻譯工具，發展成一個可持續成長、能隨使用者需求演進的智慧翻譯平台。

# 參考文獻

1. Y. Du, C. Li, R. Guo, X. Yin, W. Liu, J. Zhou, Y. Bai, Z. Yu, Y. Yang, Q. Dang, and H. Wang, “PP-OCR: A Practical Ultra Lightweight OCR System,” arXiv preprint arXiv:2009.09941, 2020.
2. NLLB Team, M. R. Costa-jussà, J. Cross, O. Çelebi, M. Elbayad, K. Heafield, K. Heffernan, E. Kalbassi, J. Lam, D. Licht, J. Maillard, A. Sun, S. Wang, G. Wenzek, A. Youngblood, B. Akula, L. Barrault, … J. Wang, “No Language Left Behind: Scaling Human-Centered Machine Translation,” arXiv preprint arXiv:2207.04672, 2022.
3. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin,“Attention is all you need,” in Advances in Neural Information Processing Systems (NeurIPS), pp. 5998–6008, 2017.