元智大學資管系

學術類畢業專題頂石課程(二)

期末報告

檢索增強生成與全參數微調在校園與企業問答場景中的整合與效能分析

關宇辰、吳承翰

實習公司：ZE

指導教授：邱昭彰 教授

中華民國 114 年 11 月

Nov, 2025

Contents

Chapter1 緒論

近年來，大規模語言模型(Large Language Model, LLM)快速發展，讓以自然語言進行資訊查詢與對話式互動成為可行的方案。相較於傳統以關鍵字搜尋或人工整理 FAQ 的方式，LLM 能夠理解較口語化的問題，並整合多筆資料生成條理化的回答。因此，不論在大學校園或企業環境中，越來越多單位開始思考，是否能利用 LLM 協助處理日常的行政詢問與產品／服務相關問題。然而，這類場域普遍存在大量且持續變動的非結構化文件，例如校規辦法、公告、操作手冊與產品說明等，單靠模型本身內建的知識難以維持內容的正確性與即時性，也讓如何結合外部文件與維護資料成為實際導入時必須面對的課題。

在實際的校園環境中，學生與教職員常需要查詢各種與行政相關的資訊，例如證件補發流程、課務與成績相關規定、獎懲與申請辦法等。這些資訊通常分散於不同單位的網頁與公告之中，格式也不盡相同，往往需要花費不少時間逐一搜尋與比對。類似的需求同樣出現在企業場域，內、外部人員針對產品規格、使用方式與作業流程的提問，仍大量仰賴人工回覆或自行翻閱文件。

隨著本研究在校園問答系統與企業文件問答專案上的累積經驗，逐步產生了三個延伸的實作與研究動機：一是建置一套能協助校園行政查詢的 RAG 問答系統；二是將此類能力結合語音與虛擬角色，應用於企業情境的多模態對話助理；三是釐清在真實文件問答任務中，應優先採用檢索增強生成還是全參數微調作為主要的模型策略。

本研究以「檢索增強生成與全參數微調在校園與企業問答場景中的整合與效能分析」為主軸，聚焦在實務系統建置與模型策略比較兩個層面。首先，在校園

情境中，目標是建置一套以 RAG 為核心的問答系統，並透過 manager 與 user 端的分工，配合爬蟲機制自動蒐集與更新相關規章與公告，降低人工維護負擔，協助使用者以自然語言查詢行政資訊。其次，在企業情境中，實作一套結合 Live2D 與語音互動的對話助理，將同樣的文件問答能力延伸到多模態介面，以評估其在產品與服務說明上的應用潛力。最後，利用企業文件問答資料設計比較實驗，系統性分析僅使用基礎模型、加入 RAG、採用全參數微調以及全參數微調結合 RAG 等不同策略，在回答品質與延遲等面向上的差異，進一步討論在不同場景下選擇合適模型策略的考量。

Chapter2 相關技術與研究

2.1 大型語言模型與問答系統

近年來，大型語言模型(Large Language Models, LLM)已成為自然語言處理領域的核心技術之一，透過在大規模文本語料上進行預訓練，模型能學習複雜的語言結構與語意關係，並在機器翻譯、摘要生成與問答系統等任務上展現接近人類的語言理解與生成能力。(Hadi , 2023)系統性整理了 LLM 的發展歷程、架構設計與訓練方法，並指出在醫療、教育、金融與工程等多種情境中，LLM 具備快速部署與高度泛化的優勢，同時也面臨知識更新不易、推論成本高與幻覺(hallucination)等限制。這些觀察說明：若要在真實情境中建置問答系統(如校園與企業內部 FAQ)，單純依賴模型參數內建的知識並不足夠，仍需要結合額外的知識來源與適當的調適策略，以兼顧回答品質與系統維運成本。(Hadi , 2023)

2.2 檢索增強生成(RAG)

為了因應「知識密集型」問答任務中知識更新與可追溯性的需求，Lewis 提出檢索增強生成(Retrieval-Augmented Generation, RAG)架構，將預訓練序列到序列模型與外部向量索引結合，於生成答案前先自資料庫中檢索相關文件片段，並將其作為條件輸入交由生成模型產生回覆。(Lewis , 2020) RAG 的關鍵概念在於：一方面保留 LLM 在語言生成上的流暢度與表達能力，另一方面透過檢索機制讓模型能參考可動態更新的文件集合，例如維基百科或組織內部文件知識庫。實驗結果顯示，RAG 在多個開放領域問答任務上，相較僅依賴模型參數的微調方法能提供更具體且可驗證的回答。(Lewis , 2020) 對本專題而言，校園與企業問答情境都牽涉到大量且持續更新的文件(如公告、辦法與說明文件)，因此採用向量資料庫與檢索增強生成

架構，成為建構可維護且易於更新之問答系統的重要基礎。

2.3 校園與企業問答應用

在實際應用層面，已有研究嘗試將問答系統與聊天機器人整合，作為協助學生與教師溝通與查詢資訊的工具。SugunaSri 設計了一套「Question Answering System Application Integrated with Chatbot Using NLP」，透過自然語言處理技術與聊天介面，提供學生提問與回覆的管道，並將問題自動轉送給合適的教師或專家處理，以改善教與學過程中的互動與問題解決效率。(SugunaSri , 2024) 與傳統規則式或關鍵字比對為主的校園網站與 FAQ 相比，此類系統更強調自然語言理解與對話式互動。然而，該研究仍主要建立在傳統 NLP 與問答模組上，較少涉及 LLM 與 RAG 的結合。本專題則以校園與企業為對象，將文件型知識(各處室公告、制度文件、產品文件等)整理進向量資料庫，並透過檢索增強生成與爬蟲更新機制，嘗試進一步提升問答系統在真實情境中的可維護性與回覆品質。(SugunaSri , 2024)

2.4 全參數微調與 RAG 策略比較

在自訂化 LLM 應用時，業界常見的兩種主要策略為：以領域資料對模型進行微調(fine-tuning)，或是採用檢索增強生成架構導入外部知識庫。Oracle 在其技術文章中指出，兩者的差異不僅在於是否修改模型參數，也牽涉到資料需求、部署成本、延遲表現與知識更新方式等多個面向。(Oracle, 2024) 文章說明，RAG 透過查詢本地或內部知識庫，能在不重新訓練模型的前提下引入最新與具隱私保護的資料，適合知識變動頻繁、強調可追溯與資料安全的場景；相對地，全參數微調則需要大量整理好的標註資料與高昂訓練成本，但能讓模型在特定領域呈現更一致的語氣與專業度，且推論架構較為單純。(Oracle, 2024) 此一比較觀點與本專題的實驗設計相呼應：在校園與企業問答場景中，同時評估「僅依賴 RAG」、「僅依賴全參數微

調」與兩者結合之策略，在回答品質與延遲上的差異，作為後續系統選型與架構設計的實務參考。

Chapter3 研究方法

3.1 研究問題與整體架構

綜合前述背景與專題簡介，本研究在方法設計上主要圍繞下列三項核心問題展開：

（一）校園情境中，如何設計與維護 RAG 問答系統？

在校園場域中，如何建置一套以檢索增強生成(RAG)為核心的問答系統，並讓各處室能夠自行維護與上傳文件，同時在規章與公告更新時，透過爬蟲機制自動補齊與更新相關資料，是系統架構與流程規畫上的首要課題。

（二）企業情境中，如何將文件問答能力結合語音與 Live2D 互動？

在企業場域中，如何在既有的文件問答能力之上，整合語音輸入與 Live2D 虛擬角色的展示與回饋，使使用者能透過自然語言與視覺化角色進行互動，是多模態對話助理設計時需要面對的問題。

(三)在真實文件問答任務中，如何選擇合適的模型策略？

在企業文件問答任務下，如何在僅使用基礎模型、加入檢索增強生成、採用全參數微調，以及全參數微調結合檢索增強生成等不同策略之間取得平衡，並比較其在回答品質與延遲等面向上的差異，構成本研究實驗設計與效能分析的主要焦點。

為了回應上述三項核心問題，本研究依照實際應用場景與模型策略配置，規劃出一個具有共用基礎、但可對應不同任務需求的整體架構。

其底層以文件蒐集與處理流程為出發點，在校園情境中，由各處室透過 manager 端上傳規章與公告等文件，並搭配爬蟲定期從系所與學校網站擷取最新

內容，經由文字擷取、切分與向量化後，儲存於向量資料庫中；使用者則透過 user 端介面輸入自然語言問題，由後端根據向量相似度檢索相關段落，再交由語言模型產生回答。

在企業情境中，企業內部的產品與服務說明文件亦經過類似的整理與嵌入流程，形成可供檢索的文件庫，並在此基礎上串接語音辨識與語音合成模組，讓 Live2D 虛擬角色能以語音與視覺方式回應使用者提問。

建置在這些實作系統之上，本研究再於企業文件問答資料集上切換不同的模型策略組合，包含僅使用基礎模型、加入檢索增強生成、採用全參數微調以及全參數微調結合檢索增強生成，透過統一的評估流程比較其效能差異，作為後續分析的基礎。

3.2 校園 RAG 問答系統與爬蟲更新

在校園場域中，本研究一開始面對的問題相當單純：希望有一套系統能協助處理與行政相關的日常詢問，讓學生與教職員可以直接以自然語言提出問題，而不必自行在各個網頁與辦法文件中反覆搜尋。隨著實作逐步推進，發現若僅由開發者集中管理所有文件，不但不利於維護，也難以及時反映各處室的最新規定，因此第二階段將系統發展為區分 manager 與 user 端的檢索增強生成(RAG)問答架構：由各單位透過 manager 端上傳與管理文件，使用者則透過 user 端進行查詢。進一步在實際使用與討論過程中，又意識到校內網站與公告內容更新頻繁，若完全仰賴人工上傳容易產生落差，於是第三階段開始規畫並實作爬蟲機制，定期從相關網站自動擷取與更新文件，搭配向量資料庫的重新建置，讓整體校園 QA 系統能在減少人工負擔的前提下，持續維持回覆內容的即時性與一致性。

在系統設計上，校園問答服務被切分為負責資料維護的 manager 端與負責

對外應答的 user 端兩個角色。其整體處理流程如圖 1 所示。各處室或管理人員透過 manager 端上傳與管理相關文件，包含校規辦法、作業流程與公告等，系統在接收到檔案後，會先進行文字擷取與預處理，再將內容依照段落或頁面切分為較小的片段，為每個片段計算向量表示並寫入向量資料庫中，必要時也會記錄來源單位與檔名等中繼資料，作為之後檢索與追溯之用。另一方面，user 端提供給學生與教職員一個以自然語言提問的介面，當使用者輸入問題時，後端會先將問題轉換為向量，根據向量相似度從資料庫中擷取與之最相關的片段，接著將這些片段與原始問題一併作為語言模型的輸入，由模型生成整合後的回答並回傳至前端顯示。透過這種將資料維護與查詢流程分離的架構設計，一方面讓不同單位可以各自負責文件內容的更新與補充，另一方面也讓使用者在不需要了解文件存放位置或格式的情況下，即可直接以問答方式取得所需資訊。

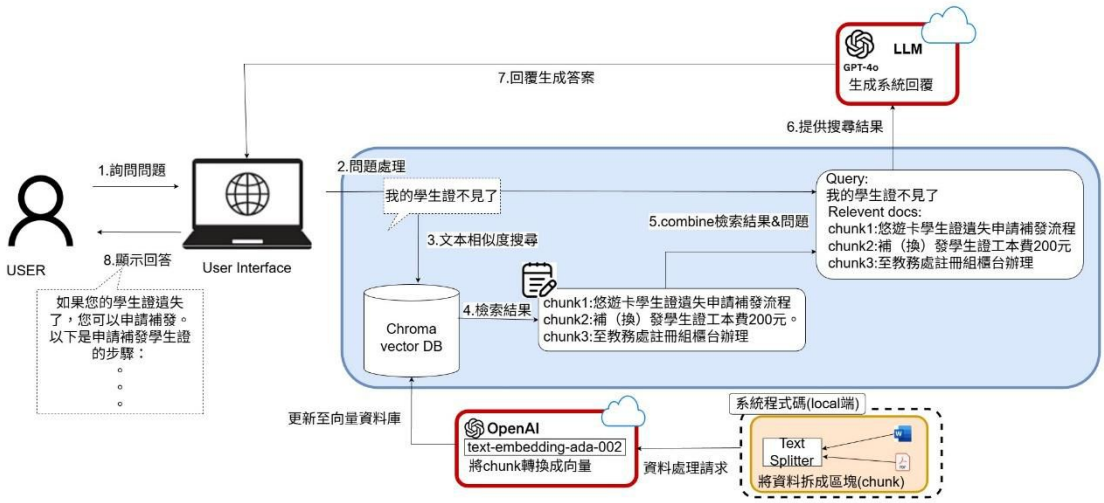

圖 1 校園 RAG 問答系統之 manager／user 架構與處理流程示意

為了減輕各處室人工維護文件的負擔並提升資料的即時性，系統在既有的 manager／user 架構之上，另外規畫並實作了爬蟲與自動更新機制(如圖 2 所示)。首先，根據實際需求挑選與行政相關的目標網站與頁面，例如系所公告、教務與學務單位的辦法說明、常見問題與下載專區等，並針對不同類型的內容設計對應的擷取方式，包含直接擷取網頁文字、下載並解析 PDF 檔案，或將表格型式的資訊轉換為可供檢索的文字描述。爬蟲程式會依預先設定的排程定期執行，將新抓取到的內容與先前版本進行比對，若發現檔案新增、刪除或內容明顯異動(例如檔案大小與最後修改時間發生變化)，便將該文件標記為需要更新，重新進行文字切分與向量化，並將對應片段寫入或覆蓋到向量資料庫中；對於已不存在的文件，則會同步移除其在資料庫中的紀錄。透過這樣的爬蟲與更新流程，校園 RAG 問答系統得以在不大幅增加管理者負擔的前提下，持續反映網站與規章的最新狀態，減少使用者查詢時遇到過期資訊的風險。

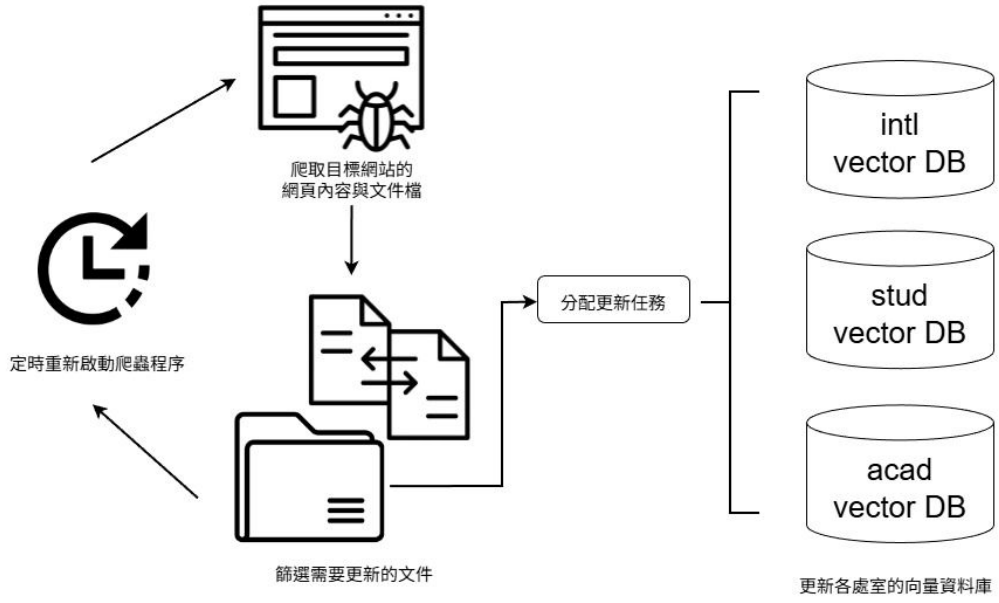

圖 2 爬蟲更新向量資料庫示意圖

3.3 企業場景：Live2D 語音 RAG 助理

在企業場域中，本研究將校園問答系統中累積的文件處理與回答能力，延伸應用至產品與服務相關的諮詢情境，並嘗試以語音與虛擬角色的方式提供互動介面。相較於僅能在瀏覽器中輸入文字的問答系統，企業在展場介紹、接待櫃台或內部教育訓練等情境，往往更需要一個能主動與人對話、具備視覺呈現的互動對象。因此，本研究在企業專案中實作了一套結合 Live2D 虛擬角色與語音輸入輸出的對話助理，讓使用者可以直接以口語提問，由系統根據企業內部的產品與文件內容產生回答，並透過角色的動作與表情進行回饋，作為檢索增強生成技術在企業問答場景中的一種多模態應用形式。

在實作層面上，Live2D 語音助理的運作流程大致可分為語音前處理、問答生成與語音輸出三個階段。其處理流程如圖 3 所示：當使用者透過麥克風進行提問時，前端會將錄製完成的語音資料傳送至後端伺服器，由語音辨識模組將音訊轉換為文字查詢。後端再根據該文字查詢，先從企業內部整理好的文件向量資料庫中檢索與問題內容最相關的片段，並將檢索結果連同原始問題一併輸入至語言模型，產生條理化的回答。接著，系統會呼叫語音合成模組，將生成的文字答案轉為語音檔，並將文字與語音播放指令一併回傳至前端，由 Live2D 虛擬角色配合語音播放同步呈現口型與動作，完成一次從語音輸入到語音與視覺回饋的對話循環。

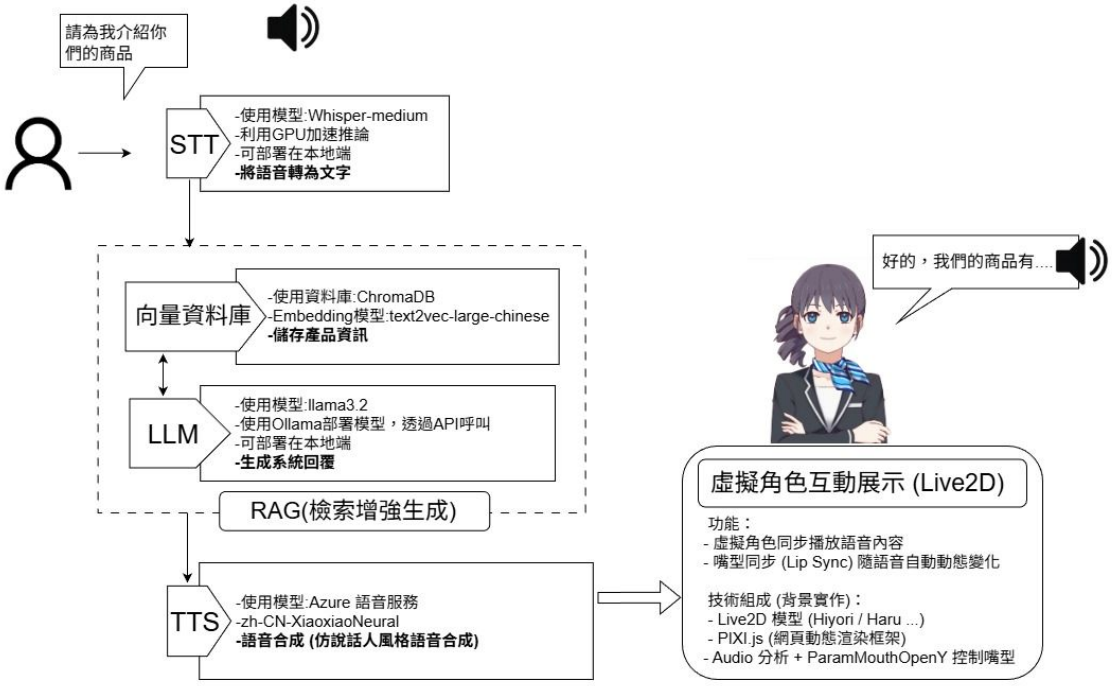

圖 2 Live2D 語音 RAG 助理 互動示意圖

在架構選擇上，企業 Live2D 語音助理同樣延續了校園 RAG 系統中「以文件為中心、透過向量檢索輔助語言模型回答」的設計思維，但在部署方式與互動介面上則做了不同的取捨。由於實際應用情境可能位於展場、接待櫃台或封閉的內部網路環境中，對網路連線穩定度與資料隱私有較高要求，因此系統在規畫時即考量以離線或局部封閉環境運行為目標，盡量將語音辨識、檢索與回答生成等核心模組部署在本地端，降低對外部雲端服務的依賴。另一方面，企業場景所使用的文件來源雖與校園不同，但在文件蒐集、文字擷取、切分與向量化等處理流程上仍可重複利用既有經驗，僅需依照企業資料的格式與內容做調整。透過這樣的設計，本研究得以在校園與企業兩種場域中共用相近的技術堆疊，一方面展示 RAG 概念在不同情境下的延伸應用，另一方面也為後續在企業文件問答資料上進行模型策略比較奠定基礎。

3.4 全參數微調 vs 檢索增強實驗設計

在模型策略的比較部分，本研究以企業文件問答為主要實驗場景，選擇實際專案中由網站爬蟲蒐集而來的中文企業產品資料作為基礎語料，約包含數百筆產品說明紀錄。經過清洗、正規化與近重複去除後，這些資料同時用於知識庫建置與評測參考答案的產生，以確保比較過程在相同資料來源與推理條件下進行。在此基礎上，實驗設計聚焦於四種模式：僅使用基礎模型作答的 Base、於基礎模型上加入檢索增強生成的 Base+RAG、透過全參數微調 LLM 內化企業知識的 FPFT，以及同時結合全參數微調與檢索增強生成的 FPFT+RAG。透過在同一組企業問答題集上比較這四種模式於語義品質、生成延遲與綜合效率等面向的差異，本研究希望釐清在真實文件問答任務中，何種策略較能在品質與時間成本之間取得合適平衡，並作為後續系統設計選擇模型路徑的參考。

本實驗所使用的資料集係來自實際企業專案中，由爬蟲從企業公開網站蒐集之中文產品資料，經彙整後約有七百筆紀錄。資料前處理時，首先僅保留與產品描述直接相關的欄位，例如產品名稱與產品說明等，並進行基本的文字清洗與正規化，包含移除多餘符號與噪音字元、統一常見格式與標點。接著，針對每筆紀錄檢查主要語言與長度區間，只保留以中文為主且介於預先設定長度範圍內的文本，並透過近重複檢測與去重方式，降低內容高度相似的紀錄對訓練與評測結果的干擾。處理完成的企業產品語料，一方面作為後續知識庫與向量索引的基礎，另一方面則用來建構問答評測集。問答集的建構採半自動流程：先以語言模型從產品說明中擷取可能的使用情境並生成初步的問題與答案，再由較大型模型或人工對每筆問答進行檢核與必要修訂，形成較為一致且可回答的最終問題與參考答案。透過這種方式，在有限人力與時程下仍可建立數量足夠且語境一致的企業問

答資料，用以支援四種模式在同一題集上的相對比較。

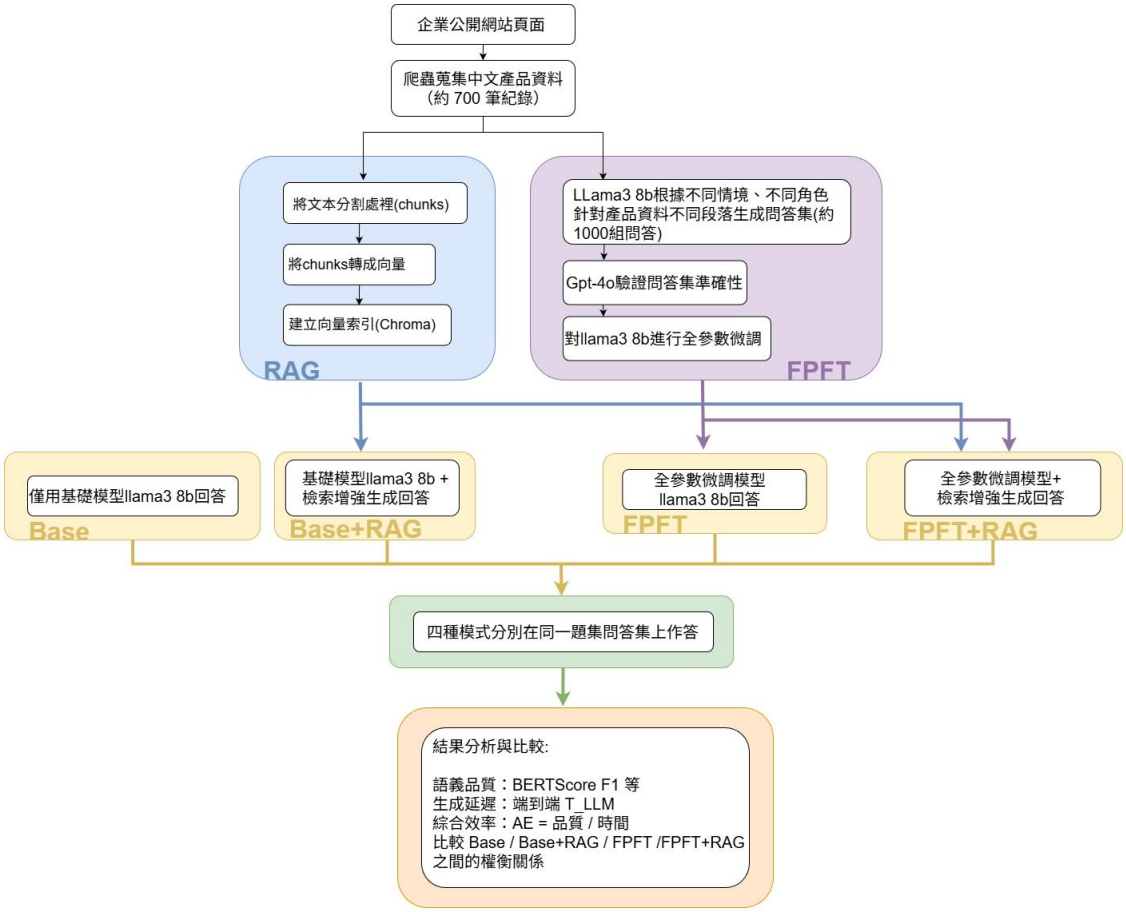

圖 3　全參數微調 vs 檢索增強實驗設計示意圖

Chapter4 初步實驗結果或初步系統展示

4.1 校園 RAG 問答系統展示

目前建置完成的校園問答系統，已整合檢索增強生成(RAG)與爬蟲更新機制，並透過前後端分離的方式，提供管理者與一般使用者兩種操作介面。在管理端，各處室可以登入系統上傳或刪除與本單位相關的規章、流程說明與公告文件，系統會自動完成文字擷取、切分與向量化，將內容寫入後端的向量資料庫中；同時，後台也會定期由爬蟲模組從校內網站與相關頁面擷取最新資料，偵測新增或變更的文件並同步更新索引。另一方面，一般使用者則可透過問答介面，以自然語言輸入問題，由系統在最新的文件內容與向量資料庫上進行檢索與生成回答，達到「管理端負責維護資料、使用端專注於查詢」的分工模式，讓整體校園行政資訊的查詢流程更為集中與友善。

在管理端介面上，系統提供給各處室一個集中管理校內文件的操作入口，其主要畫面如圖 4 所示。管理者登入後，可以在頁面中檢視目前已登錄的文件清單與基本資訊，包含檔名、上傳時間、最後更新時間以及所屬單位等，並透過按鈕執行新增、刪除或重新上傳等操作；當上傳新檔或替換既有檔案時，系統會自動觸發後端的文字擷取與切分程序，將內容轉換為多個片段並更新其向量表示，讓後續查詢可以立即使用最新版本的資料。為了協助管理者掌握資料狀態，介面同時提供簡單的搜尋與篩選功能，必要時也可以啟動孤兒掃描與修復等維護機制，檢查向量資料庫中是否存在已無對應原始檔案的紀錄並進行清理。透過這樣的圖形化操作介面，非技術背景的行政人員也能在不需了解底層實作細節的情況下，自行完成文件的上架與維護工作，減少對開發人員的依賴。

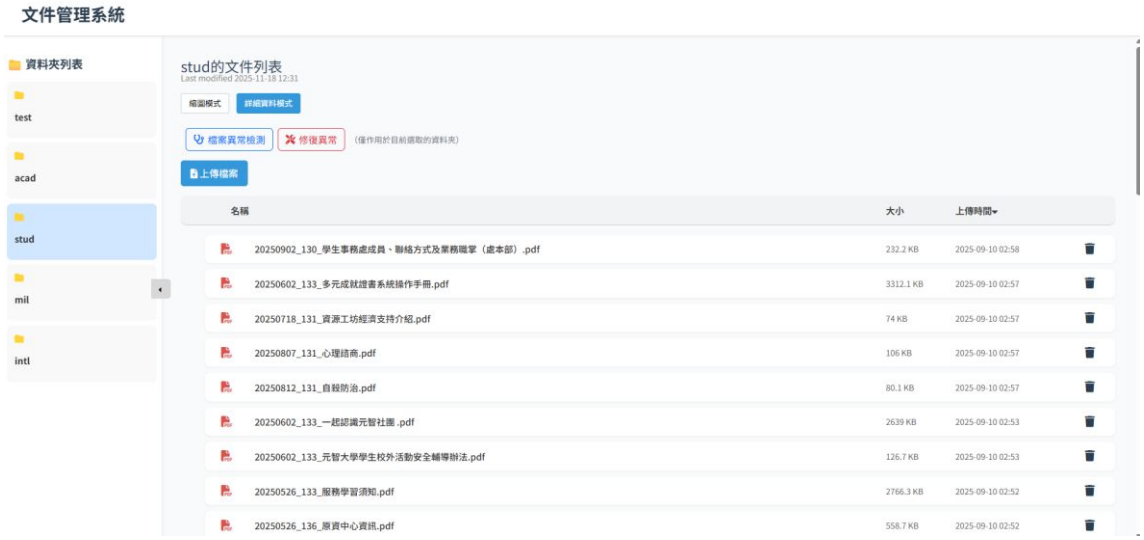

圖 4 校園 RAG 問答系統之 manager 端使用者介面示意

對一般使用者而言，校園問答系統提供了一個以自然語言提問為主的查詢介面，其主要畫面如圖 5 所示。使用者開啟系統後，只需在輸入欄位中以日常用語輸入問題，例如詢問證件補發流程、請假與成績相關規定，或某項行政作業的申請方式，即可送出查詢請求。後端接收到問題後，會在最新的向量資料庫中檢索與之最相關的文件片段，並交由語言模型整合成一段完整且易讀的回答，顯示於對話區域中；必要時，也可以在回答內容中附上來源文件名稱或簡要說明，方便使用者回溯原始規定。相較於自行在多個網站與檔案中搜尋，使用者只需透過單一介面就能得到整理過的文字說明，降低查詢門檻，也讓行政單位能以較一致的方式回覆常見問題。

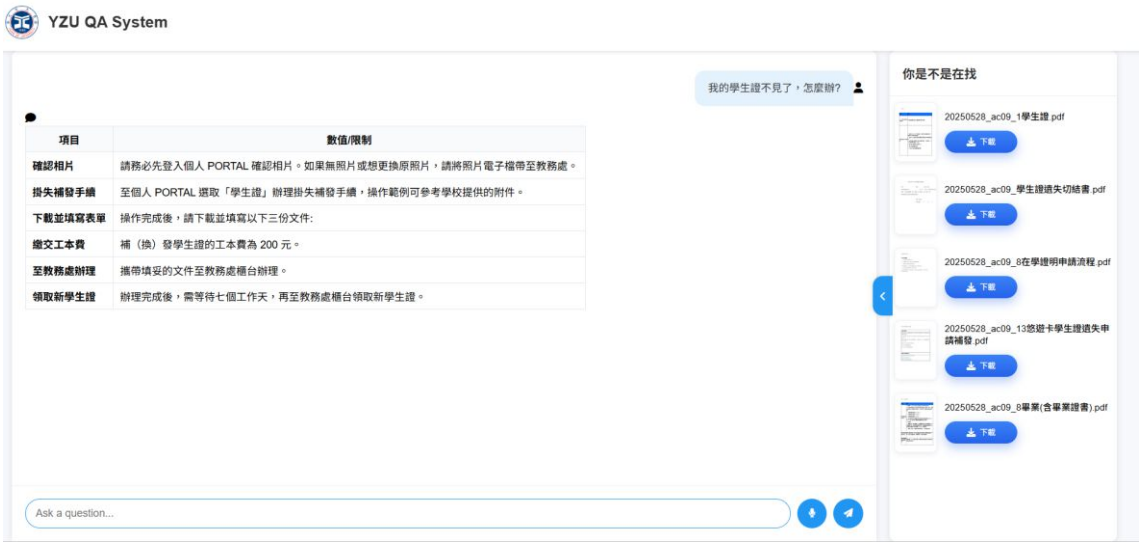

圖 5 校園 RAG 問答系統之 user 端使用者介面與問答範例

在部署方式上，校園問答系統目前已架設於雲端環境(google cloud)中，後端服務與向量資料庫透過容器化與遠端主機的方式運行，如圖 5、圖 6 所示，並搭配網頁前端對外提供存取介面。後端應用程式負責接收來自 manager 與 user 端的請求，進行文件處理、向量檢索與語言模型呼叫等工作；向量資料庫與檔案儲存則配置在 cloud storage 中。透過雲端平台所提供的監控與日誌功能，可以追蹤系統在實際運行時的資源使用狀況與錯誤紀錄，必要時也能透過調整實例規格或擴充服務數量來因應負載變化。對使用者而言，只需透過瀏覽器連線至指定網址即可使用系統，而不必關心背後的部署細節；對管理者而言，則能在既有架構下持續更新程式碼與模型設定，維持系統長期運行。

圖 6 manager 端 google cloud 容器設置示意圖

圖 7 user 端 google cloud 容器設置示意圖

4.2 企業 Live2D 語音助理展示

在本研究中，除了文字介面的問答系統之外，亦實作了一個結合 Live2D 的語音對話助理原型，將同樣的文件問答能力包裝成較具互動感的展示形式，其主要畫面如圖 8 所示。系統啟動後，螢幕上會顯示一個以 Live2D 載入的虛擬角色，使用者不需要操作按鈕或選單，只需透過鍵盤 Enter 鍵進行控制：按下 Enter 代表開始錄音，放開 Enter 則結束錄音並將語音送交後端進行處理。當後端完成語音辨識、檢索與回答生成後，會回傳一段語音檔，前端透過瀏覽器的音訊播放與音量分析機制，一方面播放系統回覆，另一方面根據音量變化驅動角色嘴型參數，使角色在回覆時呈現與語音同步的口型開合，讓整體問答過程更接近與虛擬角色對話的體驗。

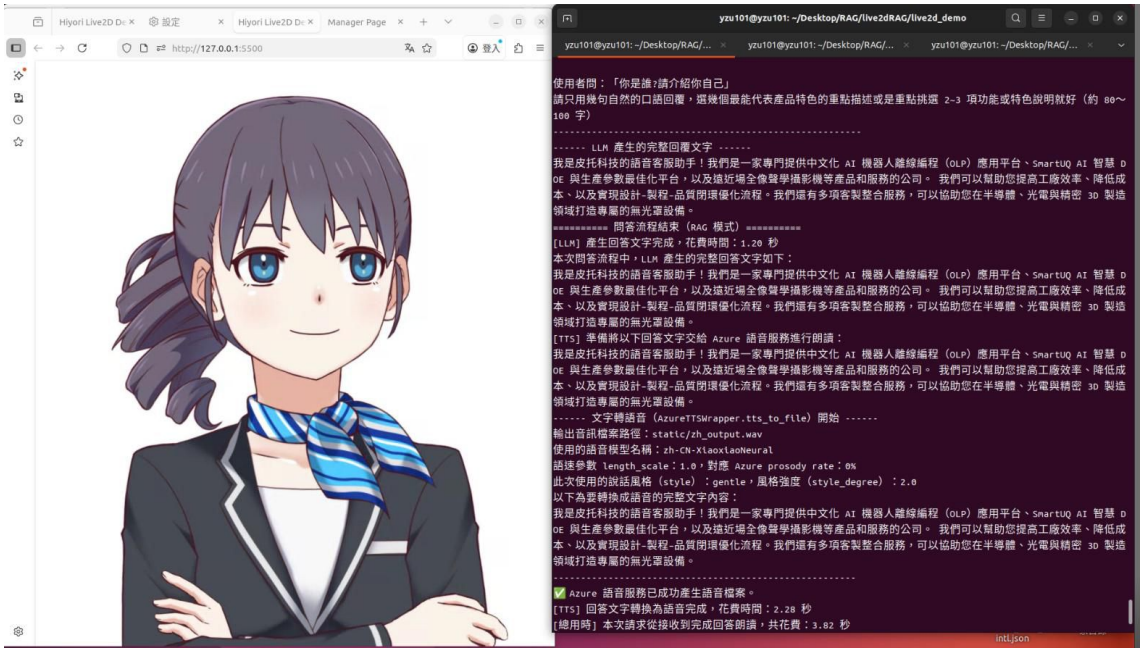

圖 8 Live2D 語音對話助理原型之操作畫面示意圖

Live2D 語音對話助理的實際運作，可分為前端錄音與後端處理兩個部分，整體流程如下所示。

(一)前端錄音與請求送出

- 使用者在畫面前按下鍵盤的 Enter 鍵時：
    - 瀏覽器透過 navigator.mediaDevices.getUserMedia 取得麥克風音訊串流(如圖 9 所示)。

```
// 初始化頁面與事件綁定
window.addEventListener('DOMContentLoaded', () => {
  window.addEventListener('keydown', onKeyDown);
  window.addEventListener('keyup', onKeyUp);
  loadModel().catch(console.error);
});

async function onKeyDown(e) {
  if (e.key !== 'Enter' || isRecording) return;

  if (isPlaying && currentAudio) {
    console.log("■ 中斷語音播放");
    currentAudio.pause();
    currentAudio.currentTime = 0;
    currentAudio.src = "";
    isPlaying = false;
    currentAudio = null;
    model.motion('Idle');
    return;
  }

  isRecording = true;
  await recordAndSendAudioStart();
}
```

圖 9　麥克風音訊串流程式碼片段

    - 建立 MediaRecorder 物件開始錄音，持續收集音訊片段。(如圖 10 所示)

```
// 錄音並傳送給 Flask 後端
async function recordAndSendAudioStart() {
  console.log("🎙 開始錄音...");
  stream = await navigator.mediaDevices.getUserMedia({ audio: true });
  mediaRecorder = new MediaRecorder(stream);
  chunks = [];

  mediaRecorder.ondataavailable = e => chunks.push(e.data);
  mediaRecorder.start();
}
```

圖 10 麥克風音訊串流程式碼片段

- 當使用者放開 Enter 鍵時：
    - MediaRecorder 停止錄音，將累積的音訊片段組合成一個 WAV 格式

的 Blob(如圖 11 所示)。

```
async function onKeyUp(e) {
  if (e.key !== 'Enter' || !isRecording) return;

  isRecording = false;
  await recordAndSendAudioStop();
}
```

圖 11 放開 Enter 鍵送出音檔程式碼片段

- 前端以 FormData 將此音訊檔命名為 audio，透過 HTTP POST 請求送至後端的 /speak API(如圖 12 所示)。

```
async function recordAndSendAudioStop() {
  return new Promise((resolve) => {
    if (!mediaRecorder || mediaRecorder.state !== "recording") {
      resolve();
      return;
    }

    mediaRecorder.onstop = async () => {
      const blob = new Blob(chunks, { type: 'audio/wav' });
      const formData = new FormData();
      formData.append("audio", blob, "recording.wav");

      console.log("📤 傳送錄音給後端...");
      const res = await fetch("/speak", { method: "POST", body: formData });
      const data = await res.json();
      console.log("🔁 文字：", data.text);
      await playWithLipSync(`${data.audio_url}?t=${Date.now()}`);
      resolve();
    };

    console.log("🔴 停止錄音...");
    mediaRecorder.stop();
    stream.getTracks().forEach(track => track.stop());
  });
}
```

圖 12 將音檔送至後端程式碼示意圖

(二)後端語音辨識與意圖判斷

- /speak 端點接收到音訊檔後：
  - 先將檔案暫存成實體 WAV 檔，方便 Whisper 模型讀取。
  - 使用部署在 GPU 上的 Whisper small 模型進行語音辨識，取得完整的文字轉寫結果(如圖 13 所示)。

```python
if 'audio' not in request.files:
    print("❌ 請求中沒有找到名為 'audio' 的檔案欄位，回傳錯誤。")
    return {"error": "音訊缺失"}, 400

total_start_time = time.perf_counter()

audio_file = request.files['audio']
with tempfile.NamedTemporaryFile(delete=False, suffix=".wav") as tmp:
    audio_path = tmp.name
    audio_file.save(audio_path)
print(f"已將使用者上傳的音訊暫存到檔案：{audio_path}")

print("STT 步驟開始，準備將語音轉成文字...")
stt_start_time = time.perf_counter()
result = stt_model.transcribe(audio_path)
stt_end_time = time.perf_counter()
print(f"[STT] 語音轉文字完成，花費時間：{stt_end_time - stt_start_time:.2f} 秒")

user_text = result['text'].strip()
print(f"[STT] 辨識出的完整文字內容為：{user_text}")
```

圖 13 /speak 端點進行語音轉文字程式碼示意圖

- 對辨識出的文字進行簡單意圖判斷(如圖 14 所示)：
    - 若內含「沒有」、「沒問題」、「不用了」、「謝謝」等關鍵詞，視為結束對話，直接產生一段結尾致意的文字回覆，並交由文字轉語音模組朗讀後回傳。
    - 若不屬於結束對話，則進入下一階段的問答處理流程。

```python
# 判斷是否為結束對話的關鍵字
if any(kw in user_text for kw in ["沒有", "沒問題", "不用了", "謝謝"]):
    print("偵測到使用者表達結束對話的意圖（包含『沒有』『沒問題』『不用了』『謝謝』其中之一）。")
    closing_text = "謝謝您的詢問，祝您有美好的一天！"
    cleaned_text = clean_text_for_tts(closing_text)
    output_path = os.path.join("static", "zh_output.wav")

    print("[TTS] 準備將以下結束訊息交給 Azure 語音服務進行朗讀：")
    print(cleaned_text)

    tts_start_time = time.perf_counter()
    tts_model.tts_to_file(cleaned_text, output_path, length_scale=1.0)
    tts_end_time = time.perf_counter()
    print(f"[TTS] 結束訊息轉換為語音完成，花費時間：{tts_end_time - tts_start_time:.2f} 秒")
    total_end_time = time.perf_counter()
    print(f"[總用時] 本次請求從接收到完成結束訊息朗讀，共花費：{total_end_time - total_start_time:.2f} 秒")
    print("=====================================")
    return {"text": closing_text, "audio_url": "/static/zh_output.wav"}


print("問答階段開始，將使用『RAG 模式』(先從向量資料庫檢索，再交給 LLM 生成回答)。")
llm_start_time = time.perf_counter()
answer_text = generate_answer(user_text)  # 用 RAG
#answer_text = generate_answer_llm_only(user_text) # 若要改用純 LLM，可改用這行
llm_end_time = time.perf_counter()
print(f"[LLM] 產生回答文字完成，花費時間：{llm_end_time - llm_start_time:.2f} 秒")
```

圖 14 簡易辨識程式碼示意圖

(三)RAG 問答與文字轉語音

- 問答階段預設採用 RAG 模式(generate_answer)，如圖 15 所示：
    - 從指定資料夾載入 Chroma 向量資料庫，如目前使用的 ./db/test。
    - 以使用者問題為查詢向量，從向量庫中取回前 10 筆最相似的文件片段與其分數。
    - 以分數門檻篩選出相關度足夠的片段(例如分數大於 0.1 的前幾筆)，將其內容整理成 context，並與原始問題一起組成提示詞後送入本機 LLM 端點(如 llama3.2:3b)。
    - LLM 產生一段適合語音播放的口語化回答文字。

```
# ========== 問答主流程 ==========
def generate_answer(question: str, folder: str = DEFAULT_FOLDER):
    print("========== 問答流程開始（使用 RAG 模式）==========")
    print(f"使用者原始問題文字：{question}")
    print(f"目前使用的向量資料庫資料夾（集合名稱）：{folder}")

    vs = get_vector_store(folder)
    scored_docs = vs.similarity_search_with_score(question, k=10)

    print("------ 向量檢索原始結果（含分數、產品、檔名與內容）------")
    if not scored_docs:
        print("沒有從向量資料庫檢索到任何文件。")
    for i, (doc, score) in enumerate(scored_docs, start=1):
        meta = doc.metadata
        product = meta.get("product", "未知產品")
        source = meta.get("filename", "未命名檔案")
        content_full = doc.page_content.strip().replace("\n", " ")
        print(f"第 {i} 筆檢索結果：")
        print(f"  相似度分數：{score:.4f}")
        print(f"  產品名稱：{product}")
        print(f"  檔案名稱：{source}")
        print(f"  文件完整內容：{content_full}")
    print("------------------------------------------------")

    filtered_pairs = [(d, s) for d, s in scored_docs if s > 0.1]
    filtered_docs = [d for d, s in filtered_pairs][:4]
```

圖 15　LLM 生成回覆程式碼示意圖

- 為了讓語音輸出更自然，系統先對文字做簡單清洗(如圖 16 所示)：
    - 移除 Markdown 標記與過長網址、電子郵件等不適合直接念出的字串。

- 將部分縮寫或技術詞彙轉換為較適合語音的念法。

```python
# ========== 工具 ==========
def clean_text_for_tts(text: str) -> str:
    text = re.sub(r"\*\*(.*?)\*\*", r"\1", text)
    text = re.sub(r"\*(.*?)\*", r"\1", text)
    text = re.sub(r'https?://\S+', '[網址略過]', text)
    text = re.sub(r'\S+@\S+', '[信箱略過]', text)
    text = text.replace("API", "A P I").replace("UI", "使用者介面")
    text = text.replace("CAD", "C A D").replace("CAE", "C A E")
    text = text.replace("HIL", "硬體在環測試").replace("MES", "生產執行系統")
    return re.sub(r'\s+', ' ', text).strip()
```

圖 16 文字篩選程式碼示意圖

- 清洗後的文字交由 AzureTTSWrapper 呼叫 Azure 語音服務(如圖 17 所示)：
    - 依據設定的語音模型與說話風格產生對應語音檔，輸出至 static/zh_output.wav。
    - 若 TTS 過程發生錯誤，則回傳預設錯誤訊息。

```python
cleaned_text = clean_text_for_tts(answer_text)
output_path = os.path.join("static", "zh_output.wav")
try:
    print("[TTS] 準備將以下回答文字交給 Azure 語音服務進行朗讀：")
    print(cleaned_text)
    tts_start_time = time.perf_counter()
    tts_model.tts_to_file(cleaned_text, output_path, length_scale=1.0)
    tts_end_time = time.perf_counter()
    print(f"[TTS] 回答文字轉換為語音完成，花費時間：{tts_end_time - tts_start_time:.2f} 秒")
except Exception as e:
    print(f"❌ TTS 發生錯誤，錯誤訊息：{e}")
    answer_text = "系統發生錯誤，請稍後再試。"

total_end_time = time.perf_counter()
print(f"[總用時] 本次請求從接收到完成回答朗讀，共花費：{total_end_time - total_start_time:.2f} 秒")
print("=====================================")
```

圖 17 呼叫 Azure 文字轉語音服務程式碼示意圖

(四)回傳與前端播放

- /speak 最終回傳包含文字與音訊路徑的 JSON 物件(如圖 18 所示)：
    - text：本次回答的完整文字內容；
    - audio_url：伺服器端產生的語音檔路徑。

```
total_end_time = time.perf_counter()
print(f"[總用時] 本次請求從接收到完成回答朗讀，共花費：{total_end_time - total_start_time:.2f} 秒")
print("========================================")

return {
    "text": answer_text,
    "audio_url": "/static/zh_output.wav"
}
```

圖 18 將音檔回傳前端示意圖

- 前端接收到回應後(如圖 19 所示)：
    - 建立 Audio 物件播放回傳的語音檔，並透過 Web Audio API 取得音量變化。
    - 根據音訊強弱動態調整 Live2D 模型的嘴型參數，讓角色在回答過程中呈現與語音同步的口型開合。
    - 若使用者在播放過程中再次按下 Enter，則中止目前語音播放並將角色動作切回靜止狀態。

```
// 播放音訊並進行嘴型同步
async function playWithLipSync(audioUrl) {
  const audio = new Audio(audioUrl);
  currentAudio = audio;  // 🔥 儲存目前播放中的 audio
  const ctx = new AudioContext();
  await ctx.resume();

  const src = ctx.createMediaElementSource(audio);
  const analyser = ctx.createAnalyser();
  analyser.fftSize = 1024;
  src.connect(analyser);
  analyser.connect(ctx.destination);

  const buf = new Uint8Array(analyser.fftSize);
  const core = model.internalModel.coreModel;

  audio.onplay = () => {
    isPlaying = true;
    const tick = () => {
      analyser.getByteTimeDomainData(buf);
      const rms = Math.sqrt(buf.reduce((s,v)=>s+(v-128)**2,0)/buf.length) / 128;
      core.setParameterValueById('ParamMouthOpenY', Math.min(rms * 5, 1.5), 1);
    };
    app.ticker.add(tick, undefined, PIXI.UPDATE_PRIORITY.HIGH);

    audio.onended = () => {
      app.ticker.remove(tick);
      core.setParameterValueById('ParamMouthOpenY', 0, 1);
      model.motion('Idle');
      ctx.close();
      isPlaying = false;
      currentAudio = null;
    };
  };

  await audio.play();
}
```

圖 19 前端播放音檔示意圖

4.3 全參數微調 vs 檢索增強實驗結果分析

(1)**實驗設定簡述**

本節針對先前所建立的中文企業產品問答資料，整理實驗結果並比較不同模型策略的差異。實驗情境以約七百筆由網站爬蟲蒐集的產品說明資料為基礎，在相同的資料來源與推理環境下，分別評估四種模式：僅使用基礎模型作答的 Base、於基礎模型上加入檢索增強生成的 Base+RAG、透過全參數微調內化企業知識的 FPFT，以及同時結合全參數微調與檢索增強生成的 FPFT+RAG。評估指標方面，以 BERTScore F1 作為語義品質的主要衡量指標，並記錄端到端生成延遲作為時間成本，同時以 AE(Accuracy per Time)作為綜合效率指標，用來觀察在品質與時間之間的權衡關係。

(2)**回答品質比較**

在回答品質方面，四種模式在 BERTScore F1 上呈現整體一致的排序關係，如圖 20 所示。整體而言，FPFT+RAG 的表現最佳，其次為單純全參數微調的 FPFT，再來是 Base+RAG，最後則是未經微調且不搭配檢索的 Base。這個結果顯示，全參數微調是品質提升的主要來源：透過在企業產品問答資料上進行微調，模型能更貼合特定領域的語彙與回答風格，產生語意上較接近參考答案的輸出；在此基礎上再加入 RAG，則能在需要補充細節與專有名詞對齊的題型上帶來小幅而穩定的額外增益。相較之下，僅在基礎模型上疊加 RAG 的 Base+RAG，雖然品質明顯優於完全不使用外部文件的 Base，但整體仍無法追上已進行全參數微調的兩種模式，顯示在此類企業文件問答場景中，單靠檢索補充上下文仍難以完全取代針對任務進行的模型調整。

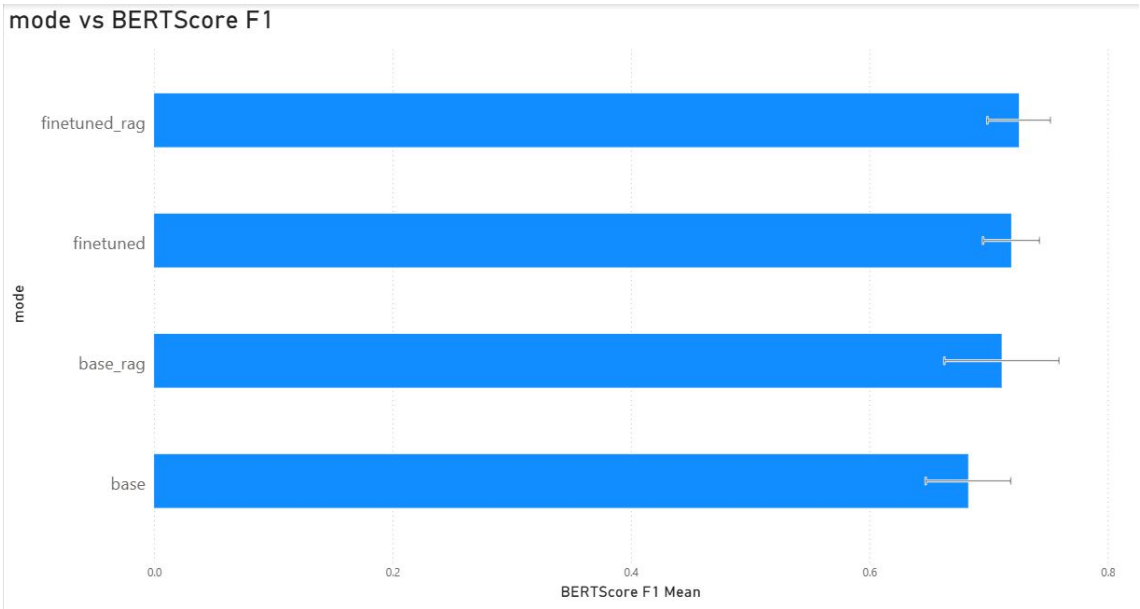

圖 20 四種模式在回答品質(BERTScore F1)上的比較圖

(3)**生成延遲比較**

在生成延遲方面，四種模式在相同硬體與推理設定下的時間表現則呈現不同的排序，如圖 21 所示。整體結果可概略描述為：FPFT 的平均延遲最低，Base 次之，FPFT+RAG 位於中間，而 Base+RAG 的延遲最高且波動也較大。這樣的差異反映出兩個層面的影響：一方面，全參數微調使模型能在不依賴長上下文的情況下直接產生較貼合的回答，推理鏈相對較短，因此在延遲與其變異度上都較為穩定；另一方面，RAG 模式在推理前多了向量檢索與片段組合的步驟，且為模型提供的提示通常更長，導致注意力運算與解碼階段的成本同步增加。特別是在 Base+RAG 模式下，模型一方面缺乏對特定領域的事先調整，另一方面又必須處理由檢索帶來的長提示，使得時間成本成為四種策略中最高的一組。

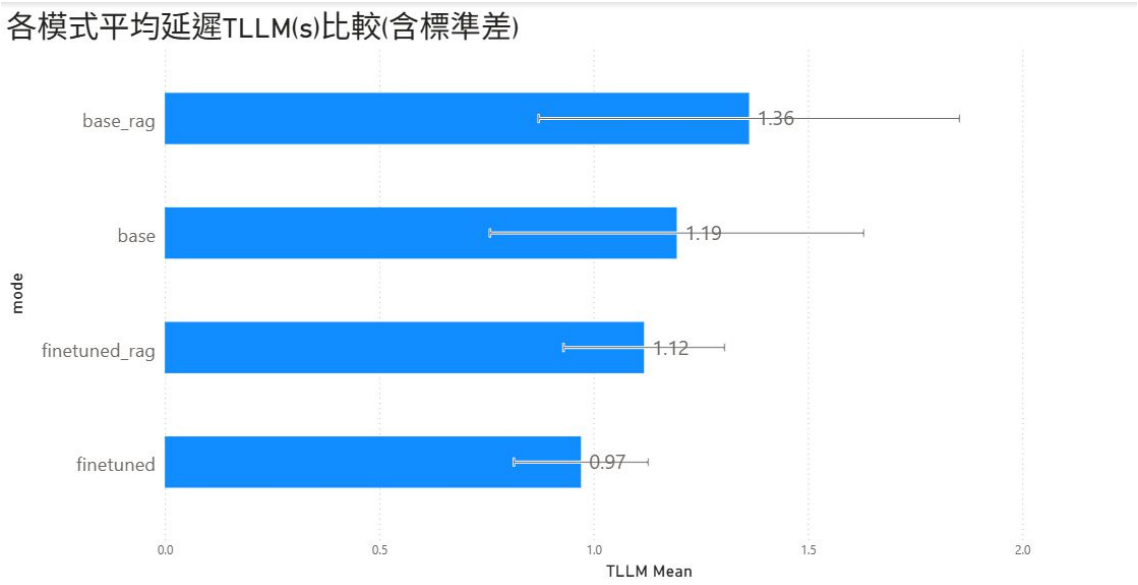

圖 21 四種模式在端到端生成延遲上的比較圖

Chapter5 結論

本研究以「檢索增強生成與全參數微調在校園與企業問答場景中的整合與效能分析」為主軸，從實務系統建置與模型策略比較兩個面向展開。首先，在校園情境中，實作了一套以 RAG 為核心的問答系統，將文件維護與查詢流程切分為 manager 與 user 兩個角色，並搭配爬蟲機制定期從校內網站與相關頁面自動擷取與更新規章與公告，讓各處室能專注於維護自身文件，而使用者則可透過自然語言介面快速取得整合後的回答。其次，本研究延伸既有的文件處理與回答能力，建置了 Live2D 語音對話助理原型，將 RAG 問答後端與 Whisper 語音辨識、Azure 語音合成以及 Live2D 虛擬角色整合，使使用者能透過鍵盤控制錄音並以口語提問，系統則回傳具備嘴型同步效果的語音回覆，展示了文件問答技術在多模態互動上的應用潛力。最後，針對企業產品問答資料，本研究設計比較實驗，系統性評估 Base、Base+RAG、FPFT 與 FPFT+RAG 四種策略在回答品質、生成延遲與綜合效率等指標上的表現，實驗結果顯示全參數微調是品質提升的主要來源，FPFT+RAG 在語義品質上最佳，而 FPFT 在時間成本與 AE 指標上具備明顯優勢，Base+RAG 則因延遲較高而在綜合效率上表現較弱，為後續在實際場域選擇模型策略提供了具體的依據。

整體而言，本研究完成了從校園行政查詢、企業多模態問答原型，到模型策略量化分析的一條完整實作與評估流程，驗證了在具體文件問答任務中，RAG 與全參數微調各自適合的情境與權衡。然而，本研究仍存在若干限制，例如校園與企業場域目前皆以特定單位與資料來源為主，尚未涵蓋更多類型的文件與多語言內容，實驗部分也聚焦於單一企業資料集與有限的模型組合。未來可在三個方向持續延伸：一

是擴充校園 RAG 系統的文件來源與使用情境，並強化監控與權限管理機制；二是將 Live2D 語音助理進一步導入實際展示或服務流程中，觀察真實使用者的互動行為與回饋；三是將本研究的比較框架套用到更多模型與硬體環境，並嘗試將 FPFT 與 RAG 的優點結合到現有系統中，持續調整模型策略與系統架構，朝向更穩定、即時且易於維護的問答服務發展。

Reference

[1] Hadi, M. U., Al-Tashi, Q., Shah, A., Qureshi, R., 2024, Large Language Models: A Comprehensive Survey of its Applications, Challenges, Limitations, and Future Prospects, https://doi.org/10.36227/techrxiv.23589741.v6

[2] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., 等人, 2020, Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks, https://arxiv.org/abs/2005.11401 arXiv+1

[3] SugunaSri, S., Leelavathy, N., Kodi, R. T., Sujatha, B., 2024, A Question Answering System Application Integrated with Chatbot Using NLP, https://indjst.org/articles/a-question-answering-system-application-integrated-with-chatbot-using-nlp

[4] Oracle, 2024, RAG vs. Fine-Tuning: How to Choose, https://www.oracle.com/tw/artificial-intelligence/generative-ai/retrieval-augmented-generation-rag/rag-fine-tuning/ oracle.com+1

附錄 A. 專題工作內容

吳承瀚：

自動循環爬蟲與向量資料庫更新

- 企業／校園網站爬蟲程式設計(URL 規劃、排程設計、例外狀況處理)
- 內容變動偵測與版本比對(checksum／hash 計算、時間戳管理)
- 爬蟲結果寫入資料夾與向量資料庫更新流程(新增／更新／刪除策略設計與實作)
- 爬蟲執行紀錄與錯誤監控(log 設計、重試機制與異常通知流程)

關宇辰：

RAG 系統前後端開發

- 校園 RAG manager 端後端 API 開發(檔案上傳、文字分塊、向量嵌入與索引建置)
- 校園 RAG manager 端與 user 端前端介面設計與實作(檔案管理介面與問答介面)
- RAG 問答流程與 user 端 API 實作(相似度檢索、context 組合與回覆生成流程)
- Live2D 語音對話助理與 RAG 串接(Whisper STT、RAG 回答、Azure TTS 與 Live2D 嘴型同步整合，含部分實驗與測試)

附錄 B. 專題心得與建議

吳承瀚:

在專題製作的過程中，我深刻地感受到解決一個技術性問題遠沒有所想的那麼簡單。原先以為只是單純修正一個小功能，在稍微研究後，發現該問題實際上牽扯到許多東西，甚至多數都是我不理解的新知識。每當我嘗試以某種方法處理，卻會發現該方法一樣有其侷限性。這迫使我只能不斷去查資料與重構程式，要學的新東西也隨之不斷增加，感覺離解決原先的問題越來越遠。然而，這正是解決問題的常態，在解決問題的過程中必然會時常發現我並不具備解決問題所需的知識。此外，真正讓我成長的往往是被那些原先不知道的問題不斷推著向前走，拓展能力邊界的過程。

關宇辰:

在這次專題的過程中，我最有感的是，從「做出一個可以動的 demo」到「做出一個真的能被別人使用的系統」，中間其實差了好幾層難度。一開始設定題目時，會覺得只要把 RAG 跑起來、Live2D 模型載進來、再加上一些微調實驗，好像就可以交差了。但實際開始實作之後才發現，每一塊都是一連串取捨：要怎麼切 manager / user 端的責任、向量資料庫要怎麼更新才不會出事、Live2D 前端的錄音與嘴型同步要怎麼跟後端的 Whisper、RAG、Azure TTS 接起來，還要同時顧效能與穩定性。很多時候，問題不是單純「某一行程式寫錯」，而是整個流程的設計本身有漏洞，只能一邊 trace log、一邊重想架構。這種不斷打掉重練、從根本設計開始調整的過程，雖然很消耗心力，但也讓我比較能把系統當成一個整體來看，而不是只盯著某個模組的實作細節。