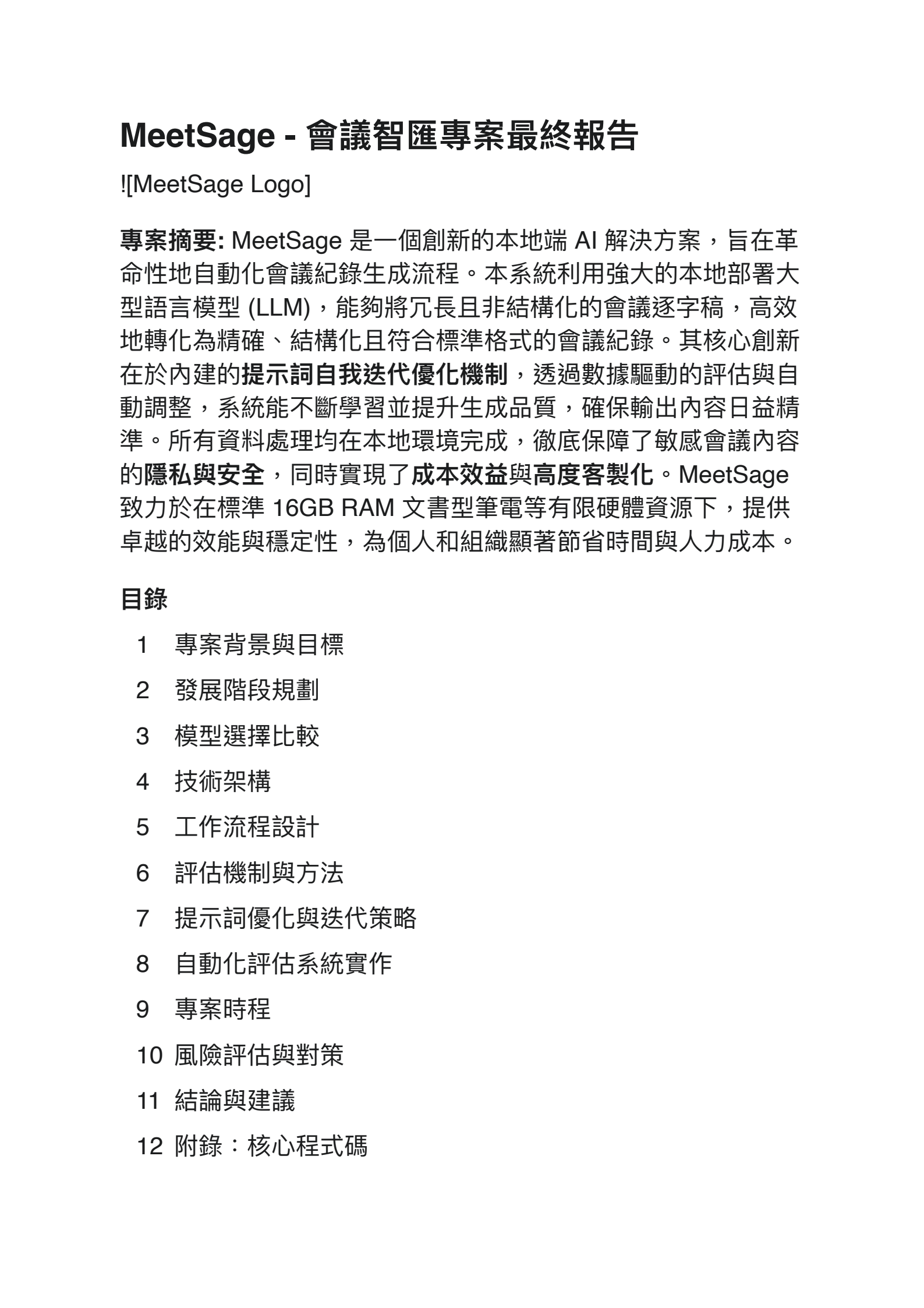
# MeetSage - 會議智匯專案最終報告




**專案摘要:** MeetSage 是一個創新的本地端 AI 解決方案，旨在革命性地自動化會議紀錄生成流程。本系統利用強大的本地部署大型語言模型 (LLM)，能夠將冗長且非結構化的會議逐字稿，高效地轉化為精確、結構化且符合標準格式的會議紀錄。其核心創新在於內建的**提示詞自我迭代優化機制**，透過數據驅動的評估與自動調整，系統能不斷學習並提升生成品質，確保輸出內容日益精準。所有資料處理均在本地環境完成，徹底保障了敏感會議內容的**隱私與安全**，同時實現了**成本效益**與**高度客製化**。MeetSage 致力於在標準 16GB RAM 文書型筆電等有限硬體資源下，提供卓越的效能與穩定性，為個人和組織顯著節省時間與人力成本。

## 目錄

# 1. 專案背景與目標

## 背景

在現代工作環境中，會議是溝通與決策的核心環節。然而，會議結束後整理詳盡且準確的會議紀錄，往往是一項耗時費力且容易出錯的任務。傳統的人工整理方式效率低下，而市面上雖有眾多基於雲端 AI 的解決方案，但對於涉及敏感商業資訊、客戶數據或專有技術的會議內容，將逐字稿上傳至外部雲端服務可能引發嚴重的**資料隱私與安全疑慮**。此外，長期依賴雲端服務的 API 費用也可能成為一筆不小的營運開支。基於這些考量，開發一個能在本地端安全、高效運行的 LLM 系統，成為了迫切且具戰略意義的需求。

## 目標

1. **自動化會議紀錄生成:**
    - 將冗長的會議逐字稿，精煉並轉化為清晰、結構化且易於閱讀的會議紀錄。
    - 確保紀錄內容涵蓋所有關鍵決策、行動項目、重要討論要點及參與者發言。
    - 使生成的紀錄符合預設的標準格式，提高內容的一致性與專業度。
2. **提示詞自我迭代優化:**
    - 建立一個**數據驅動**的評估與改進機制，能夠客觀衡量生成會議紀錄的品質。
    - 透過分析生成結果與標準參考紀錄之間的差異，自動或半自動地調整和優化 LLM 的提示詞。
    - 形成一個持續學習的閉環，使系統能從每次評估中汲取經驗，不斷提升生成品質，減少人工干預。

3 **本地處理保障隱私與安全:**

- 確保所有會議逐字稿的處理、LLM 推理、數據存儲和結果生成均在本地電腦環境中完成。
- 徹底消除資料外洩的風險，滿足嚴格的隱私保護要求。
- 擺脫對外部網路服務的依賴，實現離線工作能力。

4 **硬體適配與效能優化:**

- 針對標準 16GB RAM 文書型筆電等主流硬體配置進行優化，確保系統能夠順暢、穩定地運行。
- 在有限的記憶體和計算資源下，實現生成速度與輸出品質的最佳平衡。
- 探索並實施輕量化模型、高效量化技術和優化處理流程，以最大化硬體效能。

## 2. 發展階段規劃

本專案將分為五個主要階段，每個階段都有明確的目標、關鍵考量和主要活動，以確保專案的穩健推進和成果的逐步完善。

### 一、初始概念驗證階段 (2週)

- **核心目標:** 驗證本地 LLM 處理中文會議逐字稿並生成基本紀錄的可行性。
- **關鍵考量:**
  - Ollama 環境的快速搭建與穩定性。
  - 最小可行產品 (MVP) 的功能範圍界定，避免過度設計。
  - 初步評估小型 LLM 模型在繁體中文任務上的基本表現。

- **主要活動:**
    - **Ollama 環境搭建:** 完成 Ollama 的安裝與配置，確保其能正常運行。
    - **基礎模型下載與測試:** 下載並載入 Phi-3 Mini 或 Gemma 2 等小型 LLM 模型，進行基本問答和文本生成測試。
    - **基礎提示詞模板開發:** 設計首個用於會議紀錄生成的提示詞，包含基本指令和結構要求。
    - **初步生成測試:** 使用少量逐字稿進行會議紀錄生成，觀察初步輸出品質。
    - **建立基本的比較評估機制:** 實現手動或簡易腳本化的方式，對生成結果與參考紀錄進行初步比較。

**二、模型選擇與優化階段 (3週)**

- **核心目標:** 根據實際測試結果，確定最適合專案的 LLM 模型組合（開發用小模型與生產用大模型）及其最佳配置。
- **關鍵考量:**
    - 不同模型參數大小對記憶體消耗和生成速度的實際影響。
    - 模型在繁體中文語境下，特別是會議內容摘要和關鍵信息提取方面的表現。
    - 提示詞的微小變動對模型輸出的敏感程度。
- **主要活動:**
    - **測試多種模型性能:** 廣泛測試列表中的多個模型（如 Llama 3.2, Taiwan-LLM, TAIDE LX-7B），記錄其在統一測試集上的表現。
    - **對比不同量化版本效果:** 針對選定模型，測試不同量化

級別 (如 Q4_K_M) 對性能和資源佔用的影響。

- **優化上下文窗口設置:** 實驗不同上下文長度對生成品質和記憶體使用的影響，找到最佳平衡點。
- **設計更完善的提示詞結構:** 根據初步測試結果，迭代改進提示詞的指令清晰度、結構化要求和範例引入。
- **建立模型性能基準測試:** 開發自動化腳本，對不同模型進行標準化、可重複的性能測試，生成客觀數據。

**三、提示詞迭代系統開發階段 (4週)**

- **核心目標:** 實現自動化提示詞評估、分析和優化循環，使系統具備自我改進能力。
- **關鍵考量:**
  - 評估指標的選擇與權重分配，確保其能有效反映會議紀錄的品質。
  - 差異檢測算法的精確性，能否準確找出生成內容與標準答案的差距。
  - 提示詞版本控制的實施，確保每次優化都有據可查。
  - 自動化提示詞調整邏輯的有效性，能否根據問題自動生成合理的改進建議。
- **主要活動:**
  - **開發自動化差異分析工具:** 編寫 Python 腳本，利用 ROUGE、BERTScore 和自定義結構檢查等指標，自動比較生成紀錄與標準紀錄的差異。
  - **實現提示詞調整建議生成:** 設計規則或基於 LLM 的邏輯，根據差異分析結果，自動生成針對提示詞的改進建議。
  - **建立提示詞版本管理系統:** 整合 Git 或自定義版本控制邏輯，追蹤提示詞的每一次修改及其對應的性能數

據。

- **設計評分機制與排名策略:** 建立綜合評分模型，對不同提示詞版本進行排名，便於選擇最佳提示詞。
- **測試自動調整效果:** 運行完整的迭代循環，驗證自動生成的提示詞是否能有效提升紀錄品質。

**四、整合與優化階段 (3週)**

- **核心目標:** 將所有獨立開發的組件整合為一個連貫、高效的系統，並進行全面性能優化。
- **關鍵考量:**
  - 各模塊之間的數據流和接口設計是否流暢。
  - 系統在長時間運行和處理大量數據時的穩定性和資源消耗。
  - 使用者介面是否直觀易用，方便非技術人員操作。
  - 批次處理功能能否有效提升處理效率。
- **主要活動:**
  - **優化記憶體使用:** 針對 LLM 載入、推理和數據處理過程中的記憶體瓶頸進行優化，例如採用更高效的數據結構或分塊處理策略。
  - **改進錯誤處理機制:** 增加更健壯的錯誤捕獲和處理邏輯，確保系統在遇到問題時能給出明確的反饋或自動恢復。
  - **增強系統恢復能力:** 設計斷點續傳或狀態保存機制，防止意外中斷導致工作進度丟失。
  - **開發簡易使用者介面 (CLI/GUI):** 建立命令行介面 (CLI) 方便自動化運行，並可選開發一個輕量級的圖形使用者介面 (GUI) 方便手動操作和結果查看。

    - **整合批次處理功能:** 實現一次性處理多個會議逐字稿的功能，提高工作效率。

**五、維護與持續改進階段 (長期)**

- **核心目標:** 確保 MeetSage 系統的長期有效運作，並根據新技術和用戶需求持續進化。
- **關鍵考量:**
    - LLM 技術的快速發展，需要定期評估和引入新模型。
    - 用戶反饋是系統改進的重要來源。
    - 知識庫的擴充對於提升特定領域的紀錄品質至關重要。
- **主要活動:**
    - **建立模型更新流程:** 定期關注 Ollama 和開源 LLM 社群的最新發展，評估並整合性能更優的模型版本。
    - **開發自動化測試套件:** 擴展現有評估機制，建立全面的回歸測試套件，確保新功能或模型更新不會引入新的問題。
    - **設計使用者反饋機制:** 建立簡單的渠道收集用戶對生成紀錄品質的意見，作為進一步優化的依據。
    - **定期評估系統效能:** 持續監控系統在實際使用中的記憶體、速度和準確性，及時發現並解決潛在問題。
    - **探索新模型與新技術:** 研究如 RAG (檢索增強生成)、多模態輸入 (如直接處理音頻) 等前沿技術，為系統引入更多可能性。

## 3. 模型選擇比較

下表總結了適合16GB記憶體筆電的不同模型特性，並增加了具體應用場景和強項任務的描述，以幫助更精確地選擇模型：

| 模型名稱 | 參數大小 | 記憶體需求(量化後) | 上下文長度 | 優點 | 缺點 | 適用場景/強項任務 | 適用階段 |
|---|---|---|---|---|---|---|---|
| Phi-3 Mini | 2B | ~2-3GB | 4K | **Microsoft 開發**，極小記憶體佔用、快速響應、適合快速迭代測試、MMLU基準~70% | 摘要能力有限、繁體中文支援不佳(訓練語料中繁中佔比較少)、複雜指令理解較弱 | **資源有限的終端裝置上的簡單問答**、短文本分類、英文 QA、輕量級文本生成 | 初始開發階段 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Gemma 2 | 2B | ~3-4GB | 8K | **Google 開發**，適度的效能表現、快速響應、Google模型品質保證、多語言支援 | 繁體中文表現一般(對特定台灣用語理解有限)、專業術語處理較弱、結構化輸出能力有限 | 多語言簡單對話、英文內容生成、**快速原型開發與提示詞測試**、輕量級任務 | 初始開發階段 |
| Llama 3.2 | 3B | ~4-5GB | 8K | **Meta 最新小型模型**，較好的效能、平衡的尺寸與能力、良好的指令跟隨能力、Meta官方2025年模型 | 記憶體需求稍高、專業領域知識有限、中文能力一般 | 指令跟隨、基礎摘要、簡單對話、**提示詞精細優化** | 測試階段、提示詞優化 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Taiwan-LLM-7B-v2.1 | 7B | ~5-6GB | 8K | **台大MiuLab開發**，台灣本地化、優秀的繁體中文處理 (預訓練使用300億詞彙)、TC-Eval表現接近GPT3.5 | 較專注於學術應用、較少產業領域內容、處理速度相對較慢 | **台灣文化語境理解與生成**、學術文本生成、繁中知識問答、**本地化內容摘要測試** | 測試階段、學術導向應用 |
| TAIDE LX-7B | 7B | ~5-6GB | 8K | **台灣政府支持專案**，專注於產業應用、受過資安評估、針對台灣文化優化、官方支持開放使用、2024年最新模型 | 參數數量小於商業模型、發展時間較短、較少學術基準評測 | **政府文件處理**、台灣產業應用、**需資安合規場景**、**本地化內容摘要測試** | 測試階段、政府與產業應用 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Llama 3.1-TAIDE-LX-8B | 8B | ~6-7GB | 131K | **Meta與台灣團隊合作**，台灣本地化、強大的繁體中文處理、**超長上下文支援(131K)**、Meta模型基礎能力強、政府與學界合作開發 | 記憶體需求較高、生成速度較慢 (處理長上下文時)、全長上下文難在16GB筆電使用 (需分塊處理) | **長文會議處理與摘要**、複雜內容摘要、專業領域任務、**需大上下文視窗的分析場景** | 生產階段、最終部署 |
| Mistral 7B | 7B | ~5-6GB | 8K | **法國團隊開發**，表現接近更大模型、優秀的推理能力、開源社群支援良好 | 繁體中文支援較弱 (訓練語料中繁中佔比較少)、台灣語境理解有限、需額外本地化調整 | 一般文本生成、語法推理任務、**需開源授權且對中文要求不高的場景**、作為對照組 | 替代選項、作為對照組 |

**階段適配建議**

1. **開發階段:** 使用 Phi-3 Mini 或 Gemma 2 等小型模型進行快速原型開發和初步提示詞測試，以其低資源消耗和快速響應能力，加速早期迭代。
2. **優化階段:** 升級至 Llama 3.2 進行提示詞的精細優化。此模型在較小體積下提供了良好的指令跟隨能力，有助於驗證

複雜提示詞的效果。

3. **測試階段:** 同時使用 Taiwan-LLM-7B-v2.1 和 TAIDE LX-7B 進行本地化測試比較。這兩個模型分別代表台灣學術界和政府/產業界在繁體中文 LLM 方面的努力，通過對比能找到最適合會議紀錄場景的本地化模型。
4. **生產階段:** 部署 Llama3.1-TAIDE-LX-8B 作為最終生產模型。其結合了 Meta 的強大基礎能力和台灣本地化優化，特別是超長上下文支援，使其成為處理真實會議逐字稿的最佳選擇。

## 4. 技術架構

MeetSage 系統採用模組化分層設計，確保各組件職責清晰、易於維護和擴展。整個架構旨在最大化本地端運行的效率和安全性。

### 核心組件

1. **模型管理層 (Model Management Layer)**
    - **Ollama 運行環境:** 作為本地 LLM 的統一運行時和 API 接口，負責模型的載入、卸載和推理請求。Ollama 簡化了不同開源模型的部署和管理。
    - **模型下載與切換機制:** 允許用戶方便地下載、更新和切換不同的 LLM 模型，以適應不同階段的需求或實驗新的模型。
    - **量化設定管理:** 管理模型的量化版本（如 Q4_K_M），以在有限記憶體下運行更大參數的模型，達到效能與資源的最佳平衡。
    - **關鍵技術:** Ollama CLI, Python `subprocess` 模組調用 Ollama API。
2. **文本處理層 (Text Processing Layer)**

- **逐字稿預處理:** 對原始會議逐字稿進行初步清洗，移除時間戳、發言人標籤、重複詞、口頭禪等噪音，提高文本質量。
- **中文分詞與清理:** 使用專為繁體中文優化的分詞工具 (如 `jieba-tw`) 對文本進行精確分詞，為後續的關鍵信息提取和評估做準備。
- **段落與句子切分:** 將長篇逐字稿合理地切分為更小的處理單元，以適應 LLM 的上下文窗口限制，並保持語義連貫性。
- **關鍵技術:** Python 字符串操作、正則表達式 (`re`)、`jieba-tw` 庫。

### 3 提示詞管理層 (Prompt Management Layer)

- **提示詞模板庫:** 集中管理多個版本的提示詞模板，每個模板針對不同的生成目標或優化方向。
- **版本控制系統:** 整合 Git 或自定義的文件版本控制，追蹤提示詞的每一次修改，確保可回溯性和實驗可重複性。
- **評估記錄管理:** 儲存每個提示詞版本在不同模型和測試集上的評估結果，形成數據驅動的優化基礎。
- **關鍵技術:** GitPython 庫、文件系統操作、YAML/JSON 格式儲存提示詞元數據。

### 4 評估與優化層 (Evaluation & Optimization Layer)

- **結果比較引擎:** 自動比較 LLM 生成的會議紀錄與人工標準參考紀錄之間的差異。
- **差異分析工具:** 利用 NLP 指標 (如 ROUGE, BERTScore) 和自定義規則，量化生成內容的完整性、準確性和結構符合度，並識別具體問題類型 (如信息遺漏、格式錯誤)。

- **建議生成機制:** 根據差異分析結果，自動生成針對提示詞的改進建議，指導下一輪優化。這可以是基於規則的簡單邏輯，或未來可引入 LLM 自我反思生成。
- **關鍵技術:** `rouge-chinese`, `bert-score`, `pandas` 用於數據分析，自定義 Python 邏輯。

5 **使用者介面層 (User Interface Layer)**

- **命令行介面 (CLI):** 提供簡單的命令行接口，方便用戶啟動評估、優化或完整流程，尤其適合自動化腳本調用。
- **批次處理介面:** 允許一次性處理多個會議逐字稿，提高效率。
- **可選的簡易 GUI:** 未來可考慮開發一個輕量級的圖形使用者介面，提供更直觀的配置、運行監控和結果可視化功能，降低非技術用戶的使用門檻。
- **關鍵技術:** `argparse` 用於 CLI，未來可考慮 `tkinter` 或 `PyQt` 實現 GUI。

**資料流**

系統的數據流呈現為一個閉環，強調持續優化和迭代：

1 **逐字稿輸入:** 原始會議逐字稿作為系統的起始輸入。

2 **預處理:** 逐字稿經過文本處理層進行清洗、分詞、切分，準備好輸入 LLM。

3 **分塊:** 長文本會被智能分塊，以適應 LLM 的上下文窗口限制。

4 **提示詞套用:** 預處理後的文本與提示詞管理層選定的提示詞模板結合，形成完整的 LLM 輸入。

5 **LLM 處理:** 結合了提示詞和文本的輸入，通過模型管理層調用 Ollama 運行本地 LLM 進行推理，生成初步的會議紀

錄。

6. **後處理:** LLM 生成的原始輸出可能需要進一步的格式化、校對或整合，以符合最終的標準紀錄要求。
7. **會議紀錄輸出:** 最終生成的會議紀錄。
8. **結果比較:** 生成的會議紀錄與預先準備的「標準紀錄」進行比較，這是評估與優化層的關鍵環節。
9. **差異分析:** 評估與優化層分析比較結果，識別生成品質的問題點。
10. **提示詞優化:** 根據差異分析的洞察，自動或半自動地生成改進的提示詞建議。
11. **提示詞更新:** 新的提示詞版本被儲存並納入版本控制，形成新的「提示詞套用」循環，驅動系統的持續迭代和自我改進。

## 5. 工作流程設計

本專案的工作流程設計旨在提供一個清晰、可重複且高效的開發與運行路徑，確保在單人作業和本地端環境下的最佳實踐。

1. 初始設置階段
   這是專案啟動前的必要準備工作，確保所有基礎環境和資源到位。
   - **安裝 Ollama:** 根據 Ollama 官方指南，在本地系統上安裝 Ollama 運行時。
   - **下載模型:** 使用 Ollama CLI 下載專案所需的所有 LLM 模型，包括開發、測試和生產階段推薦的模型。
   - **設置環境變數:** 配置 Ollama 相關的環境變數，例如 `OLLAMA_KEEP_ALIVE`、`OLLAMA_KV_CACHE_TYPE`、`OLLAMA_FLASH_ATTENTION`，以優化模型載入和推理效能。

- **配置參數:** 根據硬體條件和需求，調整 Ollama 或 LLM 的運行參數（如 `num_gpu`、`num_thread` 等，如果適用）。
- **準備測試資料:** 收集或創建至少 10-20 對「會議逐字稿 - 標準會議紀錄」對照組，作為金標準數據集，並按照指定目錄結構組織。
- **關鍵設置詳解:**
  - `OLLAMA_KEEP_ALIVE=1d`: 將模型在記憶體中保留 1 天，避免頻繁載入和卸載帶來的延遲，顯著加快連續處理速度。
  - `OLLAMA_KV_CACHE_TYPE=q8_0`: 使用 8 位元量化的 KV Cache，減少記憶體佔用，尤其對於長上下文處理有益。
  - `OLLAMA_FLASH_ATTENTION=1`: 啟用 Flash Attention 技術，進一步優化注意力機制計算，提高推理速度並降低記憶體消耗。

2 開發階段工作流
此階段主要聚焦於快速原型開發和基礎功能的驗證。

- **小模型快速原型:** 使用 Phi-3 Mini 或 Gemma 2 等小型模型進行快速迭代。這些模型體積小、啟動快，非常適合在開發初期快速驗證提示詞設計和功能邏輯。
- **提示詞設計與初測:** 根據會議紀錄的需求，設計初始提示詞模板，並使用少量測試案例進行手動或簡易腳本化的生成測試。
- **快速測試評估:** 針對每次提示詞修改，進行快速、小規模的評估，主要關注生成內容的相關性和基本結構。
- **版本控制:** 即使在開發階段，也要保持提示詞和程式碼的版本控制，方便回溯和追蹤改動。

- **最佳實踐:**
    - **從簡單案例開始:** 優先處理結構清晰、內容簡單的會議逐字稿，逐步增加複雜度。
    - **保持提示詞版本控制:** 每次對提示詞進行重大修改時，都應提交到版本控制系統，並附上清晰的修改說明。
    - **對比不同提示詞的效果差異:** 即使是手動評估，也要記錄不同提示詞版本在相同輸入下的輸出差異，作為優化的依據。

3 提示詞優化工作流
這是 MeetSage 專案的核心，實現提示詞的自我迭代優化。

- **設計提示詞:** 根據上一階段的經驗，設計或選擇一個基礎提示詞版本作為優化的起點。
- **應用於測試集:** 將當前提示詞應用於預先準備的金標準測試集中的所有逐字稿，生成會議紀錄。
- **比較與標準紀錄差異:** 運行自動化評估引擎，將生成的紀錄與對應的標準參考紀錄進行客觀比較，計算各項指標。
- **生成改進建議:** 根據評估指標和差異分析結果，由優化引擎自動識別提示詞存在的問題（如結構不完整、關鍵信息遺漏等），並生成針對性的改進建議。
- **更新提示詞:** 根據建議，自動或人工調整提示詞模板，形成新的提示詞版本。
- **重複評估:** 將新的提示詞版本重新應用於測試集，再次進入評估循環，直到達到預設的品質標準。
- **優化策略:**

- **從關鍵差異點著手:** 優先解決對生成品質影響最大的問題（如決策遺漏、格式混亂）。
- **對提示詞進行模塊化設計:** 將提示詞分解為不同的功能模塊（如角色設定、任務指令、格式要求、約束條件、範例等），便於獨立優化和組合。
- **使用 A/B 測試評估不同版本:** 在每次迭代中，可以同時測試多個提示詞變體，通過客觀數據比較其效果，選擇表現最佳的進入下一輪。
- **建立提示詞效能排行榜:** 定期生成評估報告，對不同提示詞版本的綜合評分進行排名，清晰展示優化進度。

4 生產階段工作流
當系統經過充分優化並達到預期品質後，即可部署到實際生產環境中。

- **標準化輸入:** 確保輸入系統的逐字稿格式一致，並經過必要的預處理。
- **預處理:** 完整的文本預處理流程，包括噪音清除、分詞、切分等，為 LLM 提供最優質的輸入。
- **Llama3.1-TAIDE-LX-8B 處理:** 使用經過驗證的 Llama3.1-TAIDE-LX-8B 模型進行會議紀錄生成，利用其強大的繁體中文能力和長上下文處理優勢。
- **後處理格式化:** 對 LLM 的原始輸出進行最終的格式調整和校對，確保其完全符合最終的標準會議紀錄要求。
- **人工確認:** 儘管系統高度自動化，但在生產環境中，建議對最終輸出的會議紀錄進行人工審核，確保關鍵信息的絕對準確性。

- **最終輸出:** 將完成的會議紀錄導出為所需格式（如 Markdown、PDF、Word）。
- **效能考量:**
  - **限制上下文窗口長度:** 儘管 Llama3.1-TAIDE-LX-8B 支援 131K 長上下文，但在實際運行中，根據硬體和生成速度需求，可將實際使用的上下文窗口限制在 8K-16K 之間，以平衡效能和品質。
  - **關閉不必要的應用程式:** 在運行生成任務時，關閉其他佔用大量記憶體的應用程式，為 LLM 釋放更多資源。
  - **分批處理長文檔:** 對於極長的會議逐字稿，實施分塊處理策略，將其分解為多個小塊，分別生成摘要後再進行整合，避免單次處理過載。
  - **使用高度量化版本 (Q4_K_M):** 確保生產環境中使用的模型是經過高效量化的版本，以最大化在 16GB RAM 筆電上的運行效率。

## 6. 評估機制與方法

建立一套全面且客觀的評估機制是 MeetSage 專案成功的基石，它將指導開發進程並確保最終產品的品質。

### 階段進展評估標準

每個開發階段的結束都應基於明確的定量和定性指標來判斷是否可以進入下一階段。

1. **定量指標:**
   - **完成率:** 每個階段所規劃的必要任務（如功能實現、測試完成）的完成百分比。例如，若一個階段有 10 個核心任務，完成 8 個則為 80%。

- **效能閾值:** 設定 LLM 生成品質的最低可接受標準。例如，**ROUGE-L F1 分數 > 0.65** 可作為摘要品質的初步門檻。
- **錯誤指標:** 衡量生成內容中關鍵信息的遺漏或錯誤率。例如，**關鍵信息遺漏率 < 10%**。
- **穩定性指標:** 評估系統在重複運行或長時間運行下的一致性。例如，**連續 10 次測試的關鍵指標標準差 < 0.05**。

2 **定性指標:**

- **用戶故事完成與驗收:** 確認所有核心用戶故事（或需求）已實現，並通過人工審核，符合預期功能。
- **關鍵功能穩定性測試:** 確保系統的核心功能（如模型載入、生成、評估）在各種情況下都能穩定工作，不會頻繁崩潰。
- **團隊一致性評估:** 專案成員（即使是單人作業，也指自我審核）對當前階段目標達成情況的共識。

3 **階段過渡標準:**

- **初期概念驗證 → 模型選擇與優化:** 基本生成功能完成，且初步準確率達到 **>70%**。
- **模型選擇與優化 → 提示詞迭代系統開發:** 已確定開發和生產模型，提示詞架構設計完成，並已建立初步評估基準。
- **提示詞迭代系統開發 → 整合與優化:** 提示詞至少迭代 **≥5 輪**，且品質呈現持續提升趨勢，差異分析引擎和自動評估機制已初步工作。
- **整合與優化 → 維護與持續改進:** 所有核心組件已整合，完整工作流測試通過，且系統在本地環境能穩定運行 **30 分鐘以上不崩潰**。

**模型效能評估方法**

對 LLM 模型的效能評估應採用多維度衡量，以全面反映其在會議紀錄生成任務上的表現。

1. **自動化指標測量:**
    - **ROUGE 系列 (ROUGE-1、ROUGE-2、ROUGE-L):** 衡量生成文本與參考文本之間的詞彙重疊度。
        - **ROUGE-1:** 衡量單詞重疊。
        - **ROUGE-2:** 衡量雙詞組重疊。
        - **ROUGE-L:** 衡量最長公共子序列，更關注句子結構和流暢度。
    - **BLEU (Bilingual Evaluation Understudy):** 評估生成內容與標準答案的 n-gram 相似度，常用於機器翻譯，也可輔助評估摘要。
    - **BERTScore:** 使用基於 BERT 的上下文嵌入來計算生成文本與參考文本之間的**語義相似度**，能捕捉到詞彙層面之上的意義匹配。
    - **自定義 F1 指標評估關鍵信息提取能力:** 針對會議紀錄中最重要的元素（如決策、行動項目、關鍵數據），開發基於關鍵詞匹配或正則表達式的 F1 分數，量化其提取的準確性和召回率。
2. **結構完整性評估:**
    - **正則表達式檢測會議紀錄結構完整性:** 編寫正則表達式規則，自動檢查生成的會議紀錄是否包含所有預設的標題、子標題和段落（如會議主題、時間、參與者、討論內容、決議事項、後續追蹤）。目標是達到 **檢測率 > 95%**。
    - **關鍵部分識別:** 驗證 LLM 是否能準確識別並標記出會

議紀錄中的關鍵部分，例如將決策事項以粗體或特定符號突出顯示。

- **格式一致性評分:** 設計一個 1-10 分的評分標準，評估生成紀錄的整體格式是否符合預設的樣式指南（如縮進、項目符號使用、標點符號統一）。

3. **資源利用監控:**
    - **處理速度 (tokens/second):** 衡量模型每秒能生成的 token 數量，直接反映生成效率。
    - **峰值與平均記憶體使用率:** 使用系統監控工具 (如 `htop`, `任務管理器`) 記錄 LLM 運行時的記憶體佔用情況，識別潛在的記憶體洩漏或過度消耗。
    - **長時間運行穩定性測試:** 讓系統連續處理 **10 個以上**的會議逐字稿，觀察其穩定性，包括有無崩潰、錯誤率是否增加、效能是否隨時間下降。

**文本處理評估**

有效的文本處理是 LLM 輸入質量的基礎，其評估同樣重要。

1. **預處理效能指標:**
    - **雜訊清除準確率:** 衡量移除時間戳、口頭禪、重複詞等非內容元素的準確性。
    - **分詞正確率 (繁體中文專用):** 針對中文分詞的特殊性，評估分詞結果與人工標註的黃金標準的一致性。
    - **標點符號與格式標準化率:** 檢查預處理後文本的標點符號和基本格式是否統一。
2. **特殊處理效能:**
    - **專業術語識別率:** 衡量系統能否準確識別並保留會議中出現的特定領域專業術語。

- **中英混合文本處理準確度:** 評估系統在處理中英文混雜的逐字稿時，能否正確區分和保留英文專有名詞或句子。
- **會議特定標記識別率:** 衡量系統能否識別和處理會議中特有的標記，如「[主席]」、「[決議]」、「[行動項目]」等。

**比較評估基準框架**

為實現數據驅動的優化，需要建立一個堅實的比較評估基準框架。

1. **金標準資料集構建:**
    - **數量與多樣性:** 收集至少 **20 對**「逐字稿-標準會議紀錄」對照組。這些對照組應涵蓋不同類型、不同複雜度的會議（例如：決策會議、討論會議、資訊分享會議、客戶會議、技術會議等）。
    - **標註質量:** 對每對標準會議紀錄進行高質量的**人工標註**，明確標註出關鍵信息點、決策、行動項目和所有必要的結構元素，作為 LLM 輸出的「黃金標準」。
2. **多維度評分系統:**
    - **內容完整性 (0-10分):** 衡量生成紀錄是否包含了逐字稿中的所有重要信息，沒有遺漏關鍵點。
    - **組織結構性 (0-10分):** 評估紀錄的邏輯結構、段落劃分、標題使用是否清晰合理。
    - **準確性 (0-10分):** 衡量紀錄中信息的真實性，有無事實性錯誤或幻覺內容。
    - **簡潔性 (0-10分):** 評估紀錄是否去除了冗餘信息和口頭禪，語言是否精煉。
    - **可讀性 (0-10分):** 衡量紀錄的語句是否流暢、語法是否

正確、整體閱讀體驗是否良好。

3 **視覺化比較工具:**

- **差異高亮顯示:** 開發工具，能夠並排顯示生成紀錄和標準紀錄，並高亮顯示兩者之間的差異（如新增、刪除、修改的內容）。
- **遺漏與額外內容標記:** 明確標記生成紀錄中遺漏的標準信息和額外添加的冗餘信息。
- **結構匹配度熱力圖:** 可視化工具，以熱力圖形式展示生成紀錄各部分與標準紀錄對應部分的匹配程度。

# 7. 提示詞優化與迭代策略

提示詞優化是 MeetSage 專案的核心競爭力，通過系統化的策略，實現提示詞的持續改進。

**提示詞敏感度測試**

理解提示詞的敏感度對於高效優化至關重要。

1 **控制變量測試方法:**

- **單一變量修改實驗:** 每次只改變提示詞中的一個元素（例如：僅改變角色設定、僅調整格式要求、僅增減範例數量、僅修改指令強度），觀察其對輸出結果的影響。
- **標準化輸入交叉測試:** 使用相同的標準化測試案例，在不同提示詞變體下進行生成，確保結果的可比性。
- **變化記錄與分析框架:** 詳細記錄每次修改的內容、時間、以及對應的評估指標變化，形成一套可追溯的實驗日誌。

2 **敏感度量化指標:**

- **變化影響度量:** 計算提示詞某個元素變化後，各評估指

標（ROUGE、BERTScore、結構完整性等）的變動百分比，量化其敏感程度。

- **關鍵元素識別矩陣:** 建立一個矩陣，標示出哪些提示詞元素對哪些輸出品質指標影響最大。
- **敏感度熱力圖生成:** 可視化地展示提示詞不同區域或元素對模型輸出的影響程度。

3 **穩定性測試協議:**

- **噪音容忍度測試:** 在逐字稿中加入輕微的噪音（如錯別字、語氣詞、非標準標點），測試提示詞能否維持穩定的生成品質。
- **指令順序變異實驗:** 測試提示詞中指令的先後順序變化是否會顯著影響生成結果。
- **模糊指令處理一致性評估:** 評估模型在面對略微模糊或開放性指令時，生成結果的一致性和合理性。

**模型性能基準測試**

定期對模型進行性能基準測試，以確保其在不同提示詞和任務下的穩定表現。

1 **自動化測試套件:**

- **批次測試腳本:** 開發可自動化運行所有測試用例的腳本，減少人工干預。
- **標準化測試集:** 建立一個包含多種會議類型、不同長度和複雜度的標準化測試集（至少 50 個案例），確保測試的全面性。
- **性能報告模板:** 設計標準化的性能報告模板，清晰呈現各項指標數據和趨勢。

2 **多維度性能評估:**

- **處理時間:** 精確測量每 1000 字逐字稿的處理時間，評估系統的效率。
- **記憶體使用曲線:** 監控模型推理過程中的記憶體使用峰值和平均值，識別性能瓶頸。
- **輸出一致性係數:** 衡量在相同輸入和提示詞下，多次生成結果的一致性。

3 **基準對比系統:**

- **與人工紀錄比較框架:** 將系統生成的紀錄與人工整理的紀錄進行直接比較，作為最終的品質評判標準。
- **模型間橫向比較矩陣:** 對比不同 LLM 模型在相同提示詞和測試集下的表現，選擇最佳模型。
- **版本間縱向比較追蹤:** 追蹤同一模型或提示詞在不同優化版本間的性能變化，量化改進效果。

**自動化提示詞優化系統**

這是實現提示詞自我迭代的核心組件，將評估與優化緊密結合。

1 **差異分析引擎設計:**

- **文本對比算法 (基於 Diff):** 使用如 `diff-match-patch` 等庫，精確找出生成文本與參考文本之間的字符級或詞級差異。
- **差異分類框架:** 將識別出的差異自動分類為不同類型（例如：信息遺漏、信息錯誤、格式不符、冗餘內容、語法問題等），為後續的提示詞調整提供精確的依據。
- **自動報告生成器:** 根據差異分析結果，自動生成詳細的差異報告，指出問題所在。

2 **提示詞生成器架構:**

- **模板化組合系統:** 將提示詞分解為可配置的模塊（如：角色設定、任務指令、格式要求、約束條件、範例），通過自動化邏輯重新組合這些模塊，生成新的提示詞。
- **基於問題類型的強化調整:** 根據差異分析中識別出的問題類型，智能地調整提示詞中對應的指令強度或內容。例如，若結構問題突出，則強化「請嚴格按照以下結構輸出」的指令。
- **自動化權重優化 (未來擴展):** 探索使用強化學習或其他優化算法，自動調整提示詞中各指令的「權重」或強調程度，以達到最佳效果。

3 **評估-優化循環系統:**

- **全自動「生成→評估→調整」循環:** 實現一個無人值守的自動化循環，系統自動生成、自動評估、自動調整提示詞，並重複此過程。
- **表現追蹤數據庫:** 建立一個數據庫（可以是簡單的 CSV 或 JSON 文件），記錄每次迭代的提示詞版本、評估指標、改動內容和時間戳。
- **優勝策略選擇算法:** 根據綜合評分，自動選擇表現最佳的提示詞版本作為下一輪迭代的基礎，或作為最終推薦的生產提示詞。

**迭代開發方法**

有效的迭代開發方法確保專案在有限資源下實現持續改進。

1 **小批量迭代計劃:**

- **短週期迭代:** 建立短週期（例如：2-3 天/迭代）的開發節奏，快速實現小功能或解決小問題。
- **單一焦點問題解決:** 每次迭代集中解決一個具體問題或改進一個特定功能，避免同時處理過多任務導致複雜

性增加。

- **成果驗收標準:** 每次迭代結束時，都有明確的成果可以檢視和驗收。

2 **A/B 測試框架:**

- **多版本並行測試設計:** 在每次迭代中，可以同時測試多個提示詞變體 (A, B, C...)，並在同一批測試數據上運行。
- **數據收集與分析流程:** 自動收集每個變體的評估數據，並進行統計分析，比較其性能差異。
- **決策樹選擇機制:** 根據 A/B 測試結果，通過預設的決策規則（例如：綜合評分提升超過 X%）自動選擇優勝版本。

3 **版本控制與回溯機制:**

- **Git 工作流整合:** 嚴格使用 Git 進行提示詞和程式碼的版本控制，每次迭代結束或重要修改後進行提交。
- **性能數據與版本關聯:** 將每次提交的 Git Hash 與對應的評估性能數據關聯起來，方便追溯是哪次代碼或提示詞修改導致了性能變化。
- **回溯與分支策略:** 允許輕鬆回溯到歷史版本（如果新版本效果不佳），並利用 Git 分支進行並行實驗。

4 **增量改進策略:**

- **先解決「致命問題」:** 優先處理對系統功能或品質有嚴重影響的「致命問題」（如決策遺漏、模型崩潰）。
- **再處理「改進點」:** 在核心問題解決後，再逐步優化「改進點」（如語句流暢度、簡潔性）。
- **優先處理高頻出現的問題:** 根據差異分析報告，優先解決在多個測試案例中普遍存在的問題。

- **保留成功元素:** 在修改提示詞時，保留那些已被證明有效的指令或結構，只修改問題部分。

## 8. 自動化評估系統實作

本章將詳細說明 MeetSage 自動化評估系統的具體實作細節，包括系統架構、各模塊功能、目錄結構以及運行指令。整個系統設計為完全在本地運行，不依賴任何雲端服務，確保資料隱私與安全。

### 系統架構

本系統使用 Python 開發，包含三個核心模塊，協同工作以實現提示詞的自動評估與優化：

1. **評估引擎 (evaluate.py):** 負責模型與提示詞的實際評估，包括呼叫 LLM 生成、計算各項評估指標並生成報告。
2. **優化引擎 (prompt_optimizer.py):** 負責分析評估結果，識別提示詞存在的問題，並根據預設規則或智能邏輯生成改進的提示詞版本。
3. **批次處理 (run_evaluations.py):** 作為系統的入口點，提供統一的命令行介面，方便用戶啟動評估、優化或完整的迭代流程，並管理日誌和結果輸出。

### 目錄結構

為確保專案的組織性和可維護性，建議採用以下標準目錄結構：

```
meetsage/
├── data/
│   ├── transcripts/        # 存放會議逐字稿文件 (e.g., meeting_001.txt)
│   └── references/         # 存放標準會議紀錄文件 (e.g., meeting_001.txt)
├── prompts/                # 存放提示詞模板文件 (e.g., prompt_001.txt)
│   ├── prompt_001.txt
```

```
|   ├── prompt_002.txt
|   └── ...
├── scripts/
|   ├── evaluate.py          # 評估引擎的核心程式碼
|   ├── prompt_optimizer.py  # 優化引擎的核心程式碼
|   └── run_evaluations.py   # 批次運行腳本 (系統入口)
├── results/                 # 存放所有運行結果的輸出目錄 (會按時間戳生成子目錄)
└── README.md                # 專案說明文件
```

### 核心評估引擎 (evaluate.py)

評估引擎是系統的「眼睛」，負責客觀地衡量 LLM 生成內容的品質。

- **功能詳述:**
  1. **模型管理:** 內置預定義的模型列表，並通過 `subprocess` 模組調用 Ollama CLI，實現對不同本地 LLM 模型的無縫呼叫。
  2. **數據處理:** 負責從 `data/transcripts/` 和 `data/references/` 目錄載入會議逐字稿和對應的標準參考會議紀錄。
  3. **指標計算:**
     - **ROUGE (rouge_chinese):** 計算生成文本與參考文本之間的詞彙重疊度（ROUGE-1, ROUGE-2, ROUGE-L 的 F1 分數）。
     - **BERTScore (bert_score):** 基於預訓練的 BERT 模型計算生成文本與參考文本之間的語義相似度。
     - **關鍵信息保留率:** 通過 `jieba` 分詞後，比較參考文本中的關鍵詞（名詞、數字等）在生成文本中的出現比例。

- **結構完整性:** 簡化地檢查生成文本是否包含預設的會議紀錄關鍵部分（如會議主題、決議事項等）。

4. **報告生成:** 將所有評估結果匯總為 `pandas.DataFrame`，保存為 CSV 文件，並生成易於閱讀的 Markdown 格式報告，包含提示詞和模型的性能排名，以及可視化圖表（PNG 格式）。

```
import os
import json
import time
import subprocess
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import jieba # 繁體中文分詞
from rouge_chinese import Rouge # 中文ROUGE評估
from bert_score import score as bert_score # BERTScore評估
from diff_match_patch import diff_match_patch # 用於文本差異比較

class PromptEvaluator:
    """
    提示詞評估引擎，負責呼叫LLM生成、計算評估指標並生成報告。
    """
    def __init__(self, data_dir, prompts_dir, output_dir):
        """
        初始化評估器。
        Args:
            data_dir (str): 存放逐字稿和參考會議紀錄的目錄。
            prompts_dir (str): 存放提示詞模板的目錄。
            output_dir (str): 存放評估結果和報告的輸出目錄。
        """
        self.data_dir = data_dir
        self.prompts_dir = prompts_dir
        self.output_dir = output_dir
        self.results = [] # 儲存評估結果

        # 定義要評估的模型列表
```

```
        self.models = [
            {"name": "phi-3-mini", "params": "2B"},
            {"name": "gemma-2b", "params": "2B"},
            {"name": "llama3-8b", "params": "8B"}, # Llama 3 8B作為通用大
型模型對比
            {"name": "taiwan-llm-7b-v2.1", "params": "7B"},
            {"name": "taide-7b", "params": "7B"},
            {"name": "llama3.1-taide-8b", "params": "8B"}, # 最終生產模型
        ]

        # 確保輸出目錄存在
        if not os.path.exists(output_dir):
            os.makedirs(output_dir)

    def load_transcript(self, file_id):
        """
        載入指定ID的逐字稿文件。
        Args:
            file_id (int): 逐字稿文件的數字ID。
        Returns:
            str: 逐字稿內容。
        """
        file_path = os.path.join(self.data_dir, "transcripts",
f"meeting_{file_id:03d}.txt")
        if not os.path.exists(file_path):
             raise FileNotFoundError(f"找不到逐字稿文件: {file_path}")
        with open(file_path, "r", encoding="utf-8") as f:
            return f.read()

    def load_reference(self, file_id):
        """
        載入指定ID的參考會議紀錄文件。
        Args:
            file_id (int): 參考會議紀錄文件的數字ID。
        Returns:
            str: 參考會議紀錄內容。
        """
        file_path = os.path.join(self.data_dir, "references",
f"meeting_{file_id:03d}.txt")
        if not os.path.exists(file_path):
```

```
            raise FileNotFoundError(f"找不到參考會議紀錄文件: {file_path}")
        with open(file_path, "r", encoding="utf-8") as f:
            return f.read()

    def load_prompt(self, prompt_id):
        """
        載入指定ID的提示詞模板文件。
        Args:
            prompt_id (int): 提示詞模板文件的數字ID。
        Returns:
            str: 提示詞模板內容。
        """
        file_path = os.path.join(self.prompts_dir,
f"prompt_{prompt_id:03d}.txt")
        if not os.path.exists(file_path):
            raise FileNotFoundError(f"找不到提示詞文件: {file_path}")
        with open(file_path, "r", encoding="utf-8") as f:
            return f.read()

    def generate_with_ollama(self, model, prompt, max_tokens=2048):
        """
        使用Ollama呼叫指定模型生成會議紀錄。
        Args:
            model (str): Ollama中的模型名稱。
            prompt (str): 完整的提示詞內容。
            max_tokens (int): 生成的最大token數。
        Returns:
            dict: 包含生成文本、耗時和成功狀態的字典。
        """
        start_time = time.time()
        try:
            # 使用subprocess呼叫ollama命令
            result = subprocess.run(
                ["ollama", "run", model, prompt, "--format", "json"],
                capture_output=True,
                text=True,
                check=True # 如果命令返回非零退出碼，則拋出
CalledProcessError
            )
```

```
            end_time = time.time()
            # Ollama --format json 會返回一個包含 "response" 字段的JSON
            response = json.loads(result.stdout)
            return {
                "text": response.get("response", ""), # 確保response字段存在
                "time_taken": end_time - start_time,
                "success": True
            }
        except FileNotFoundError:
             end_time = time.time()
             return {
                 "text": "錯誤: Ollama命令找不到。請確保Ollama已安裝並在
PATH中。",
                 "time_taken": end_time - start_time,
                 "success": False
             }
        except subprocess.CalledProcessError as e:
            end_time = time.time()
            return {
                "text": f"Ollama命令執行失敗: {e.stderr}",
                "time_taken": end_time - start_time,
                "success": False
            }
        except json.JSONDecodeError:
             end_time = time.time()
             return {
                 "text": f"Ollama返回的不是有效的JSON: {result.stdout}",
                 "time_taken": end_time - start_time,
                 "success": False
             }
        except Exception as e:
            end_time = time.time()
            return {
                "text": f"生成過程中發生未知錯誤: {str(e)}",
                "time_taken": end_time - start_time,
                "success": False
            }


    def calculate_metrics(self, generated, reference):
```

```
        """
        計算生成文本與參考文本之間的評估指標。
        Args:
            generated (str): LLM生成的會議紀錄文本。
            reference (str): 標準參考會議紀錄文本。
        Returns:
            dict: 包含各種評估指標的字典。
        """
        metrics = {}

        # 基本指標
        metrics["length_ratio"] = len(generated) / len(reference) if
len(reference) > 0 else 0
        metrics["char_count"] = len(generated)

        # ROUGE指標 (ROUGE-1, ROUGE-2, ROUGE-L 的 F1 分數)
        try:
            rouge = Rouge()
            # ROUGE需要輸入是列表格式
            scores = rouge.get_scores([generated], [reference])
            metrics["rouge-1"] = scores[0]["rouge-1"]["f"]
            metrics["rouge-2"] = scores[0]["rouge-2"]["f"]
            metrics["rouge-l"] = scores[0]["rouge-l"]["f"]
        except Exception as e:
            # 處理ROUGE計算失敗的情況
            metrics["rouge-1"] = 0
            metrics["rouge-2"] = 0
            metrics["rouge-l"] = 0
            print(f"ROUGE計算失敗: {str(e)}")

        # BERTScore (使用中文模型)
        try:
            # BERTScore也需要輸入是列表格式
            P, R, F1 = bert_score([generated], [reference], lang="zh")
            metrics["bert_score"] = F1.mean().item() # 取平均F1分數並轉為
Python數值
        except Exception as e:
            # 處理BERTScore計算失敗的情況
            metrics["bert_score"] = 0
```

```
            print(f"BERTScore計算失敗:{str(e)}")

        # 關鍵信息保留率 (簡化版: 比較參考文檔中的名詞和數字在生成文檔中的出現率)
        # 更複雜的實現可能需要NER或關鍵詞抽取
        try:
            # 使用jieba進行分詞
            reference_words = set(jieba.cut(reference))
            generated_words = set(jieba.cut(generated))
            # 計算交集
            intersection = reference_words.intersection(generated_words)
            metrics["key_info_retention"] = len(intersection) / len(reference_words) if len(reference_words) > 0 else 0
        except Exception as e:
             metrics["key_info_retention"] = 0
             print(f"關鍵信息保留率計算失敗: {str(e)}")



        # 結構完整性檢查 (檢查是否包含必須的會議紀錄部分)
        # 這是一個基於關鍵詞的簡化檢查，更完善的需要NLP解析
        required_sections = ["會議主題", "時間", "參與者", "討論內容", "決議事項", "後續追蹤"]
        section_presence = sum(1 for section in required_sections if section in generated)
        metrics["structure_completeness"] = section_presence / len(required_sections) if len(required_sections) > 0 else 0

        return metrics

    def evaluate_prompt(self, prompt_id, meeting_ids=None):
        """
        評估單個提示詞在所有模型和指定會議上的表現。
        Args:
            prompt_id (int): 要評估的提示詞ID。
            meeting_ids (list, optional): 要評估的會議ID列表。如果為None，預設評估前5個會議。
        """
        if meeting_ids is None:
            # 預設評估前5個會議，可以根據實際情況調整
```

```
            meeting_ids = range(1, 6)

        try:
            prompt_template = self.load_prompt(prompt_id)
        except FileNotFoundError as e:
            print(f"跳過提示詞 {prompt_id}: {e}")
            return # 如果提示詞文件不存在，則跳過

        for meeting_id in meeting_ids:
            try:
                transcript = self.load_transcript(meeting_id)
                reference = self.load_reference(meeting_id)
            except FileNotFoundError as e:
                 print(f"跳過會議 {meeting_id}: {e}")
                 continue # 如果會議文件不存在，則跳過當前會議

            for model_info in self.models:
                model_name = model_info["name"]
                model_params = model_info["params"]

                # 填充提示詞模板
                # 這裡只替換了 {transcript}，如果提示詞需要其他變數，需要在此處添加替換邏輯
                filled_prompt = prompt_template.replace("{transcript}", transcript)

                print(f"正在評估: 提示詞 #{prompt_id}, 會議 #{meeting_id}, 模型 {model_name}")

                # 生成會議紀錄
                generation_result = self.generate_with_ollama(model_name, filled_prompt)

                if generation_result["success"]:
                    generated_text = generation_result["text"]
                    time_taken = generation_result["time_taken"]

                    # 計算評估指標
                    metrics = self.calculate_metrics(generated_text, reference)
```

```
                # 儲存生成的會議紀錄到輸出目錄
                output_filename = f"prompt{prompt_id:03d}
_meeting{meeting_id:03d}_{model_name.replace(':', '_')}.txt"
                output_path = os.path.join(self.output_dir, output_filename)
                with open(output_path, "w", encoding="utf-8") as f:
                    f.write(generated_text)

                # 記錄結果
                result_entry = {
                    "prompt_id": prompt_id,
                    "meeting_id": meeting_id,
                    "model": model_name,
                    "model_params": model_params,
                    "time_taken": time_taken,
                    **metrics # 將計算出的指標添加到結果字典中
                }
                self.results.append(result_entry)
            else:
                # 記錄失敗結果
                result_entry = {
                    "prompt_id": prompt_id,
                    "meeting_id": meeting_id,
                    "model": model_name,
                    "model_params": model_params,
                    "time_taken": generation_result["time_taken"],
                    "success": False,
                    "error": generation_result["text"] # 記錄錯誤信息
                }
                self.results.append(result_entry)
                print(f"生成失敗: {generation_result['text']}")


    def evaluate_all_prompts(self, prompt_ids=None, meeting_ids=None):
        """
        評估多個提示詞。
        Args:
            prompt_ids (list, optional): 要評估的提示詞ID列表。如果為
None，自動檢測prompts目錄下的所有提示詞。
            meeting_ids (list, optional): 要評估的會議ID列表。
        Returns:
```

```
        pandas.DataFrame: 包含所有評估結果的DataFrame。
    """
    if prompt_ids is None:
        # 自動檢測prompts目錄下符合命名格式的提示詞文件
        prompt_ids = []
        for file in os.listdir(self.prompts_dir):
            if file.startswith("prompt_") and file.endswith(".txt"):
                try:
                    # 從文件名中提取數字ID
                    prompt_id = int(file.replace("prompt_", "").replace(".txt",
"",))
                    prompt_ids.append(prompt_id)
                except ValueError:
                    # 如果文件名格式不正確，跳過
                    continue
        prompt_ids.sort() # 按ID排序

    if not prompt_ids:
        print("沒有找到任何提示詞文件進行評估。請確保prompts目錄下
有prompt_XXX.txt文件。")
        return pd.DataFrame()


    self.results = [] # 清空之前的結果
    for prompt_id in prompt_ids:
        self.evaluate_prompt(prompt_id, meeting_ids)

    # 將結果轉換為pandas DataFrame並保存為CSV
    results_df = pd.DataFrame(self.results)

    # 確保輸出目錄存在
    if not os.path.exists(self.output_dir):
        os.makedirs(self.output_dir)

    csv_path = os.path.join(self.output_dir, "evaluation_results.csv")
    results_df.to_csv(csv_path, index=False)
    print(f"評估結果已保存到: {csv_path}")

    return results_df
```

```
    def generate_report(self):
        """
        根據評估結果生成Markdown格式的報告。
        Returns:
            str: Markdown格式的報告內容。
        """
        # 從保存的CSV文件載入結果，確保報告是基於最新的評估數據
        csv_path = os.path.join(self.output_dir, "evaluation_results.csv")
        if not os.path.exists(csv_path):
            print("找不到評估結果文件，無法生成報告。請先運行評估。")
            return "找不到評估結果文件，無法生成報告。"

        df = pd.read_csv(csv_path)

        if df.empty:
            print("評估結果為空，無法生成報告。")
            return "評估結果為空，無法生成報告。"

        # 篩選成功的結果進行指標計算
        successful_df = df[df.get("success", True) != False].copy() # 使用.copy()避免SettingWithCopyWarning

        if successful_df.empty:
            report = "# 提示詞評估報告\n\n## 評估失敗\n\n所有評估嘗試均失敗，請檢查Ollama設置和模型是否可用。"
            report_path = os.path.join(self.output_dir, "evaluation_report.md")
            with open(report_path, "w", encoding="utf-8") as f:
                f.write(report)
            print(f"報告已生成 (所有評估失敗): {report_path}")
            return report


        # 計算每個提示詞的平均表現
        # 這裡使用一個簡單的綜合分數，權重可以根據需要調整
        prompt_performance = successful_df.groupby("prompt_id").agg({
            "rouge-l": "mean",
            "bert_score": "mean",
            "key_info_retention": "mean",
```

```
            "structure_completeness": "mean",
            "time_taken": "mean"
        }).reset_index()

        # 計算綜合分數 (權重可調整)
        # 假設各指標同等重要，時間越短越好（取倒數或負值）
        # 注意：這裡簡單取負值，實際中可能需要更複雜的歸一化
        prompt_performance["composite_score"] = (
            prompt_performance["rouge-l"] * 0.25 +
            prompt_performance["bert_score"] * 0.25 +
            prompt_performance["key_info_retention"] * 0.25 +
            prompt_performance["structure_completeness"] * 0.25
            # 暫時不納入time_taken到綜合分數，因為它受硬體影響大，主要用於參考
            # - prompt_performance["time_taken"] * 0.05 # 假設時間權重為5%
        )

        # 根據綜合分數排序
        prompt_performance = prompt_performance.sort_values("composite_score", ascending=False)

        # 生成報告內容
        report = "# 提示詞評估報告\n\n"
        report += "本報告基於使用Ollama呼叫不同模型，對各提示詞在指定會議逐字稿上生成的會議紀錄進行自動化評估。\n\n"

        report += "## 提示詞排名 (基於綜合評分)\n\n"
        report += "綜合評分是根據 ROUGE-L, BERTScore, 關鍵信息保留率, 結構完整性 的平均值計算得出 (權重均為0.25)。\n\n"
        report += prompt_performance.to_markdown(index=False) + "\n\n"

        report += "## 各模型表現對比 (平均值)\n\n"
        # 計算各模型的平均表現
        model_performance = successful_df.groupby(["model", "model_params"]).agg({
            "rouge-l": "mean",
            "bert_score": "mean",
            "key_info_retention": "mean",
```

```
            "structure_completeness": "mean",
            "time_taken": "mean"
        }).reset_index()

        # 計算模型綜合分數 (與提示詞綜合分數邏輯一致)
        model_performance["composite_score"] = (
            model_performance["rouge-l"] * 0.25 +
            model_performance["bert_score"] * 0.25 +
            model_performance["key_info_retention"] * 0.25 +
            model_performance["structure_completeness"] * 0.25
        )
        model_performance =
model_performance.sort_values("composite_score", ascending=False)

        report += model_performance.to_markdown(index=False) + "\n\n"

        report += "## 評估指標說明\n\n"
        report += "- **ROUGE-L (F1):** 衡量生成文本與參考文本之間的最
長公共子序列相似度。\n"
        report += "- **BERTScore (F1):** 基於BERT模型計算生成文本與參
考文本之間的語義相似度。\n"
        report += "- **關鍵信息保留率:** 參考文本中的關鍵詞 (名詞、數字
等) 在生成文本中出現的比例 (簡化計算)。\n"
        report += "- **結構完整性:** 會議紀錄是否包含必須的結構部分 (如
會議主題、決議事項等) 的比例 (簡化檢查)。\n"
        report += "- **Time Taken (秒):** 生成會議紀錄所花費的時間。\n\n"

        # 視覺化 (生成圖表文件)
        try:
            plt.figure(figsize=(12, 6))
            # 確保提示詞ID是字符串類型，避免繪圖問題
            prompt_ids_str = prompt_performance["prompt_id"].astype(str)
            plt.bar(prompt_ids_str, prompt_performance["composite_score"])
            plt.xlabel("提示詞ID")
            plt.ylabel("綜合評分")
            plt.title("提示詞效能比較")
            plt.tight_layout()
            prompt_plot_path = os.path.join(self.output_dir,
"prompt_performance.png")
```

```
            plt.savefig(prompt_plot_path)
            plt.close() # 關閉圖形，釋放記憶體
            report += f"## 提示詞效能圖表\n\n![提示詞效能比較圖表]
({os.path.basename(prompt_plot_path)})\n\n"
        except Exception as e:
            print(f"生成提示詞效能圖表失敗: {str(e)}")
            report += "## 提示詞效能圖表\n\n生成圖表失敗。\n\n"


        try:
            plt.figure(figsize=(12, 6))
            # 確保模型名稱是字符串類型
            models_str = model_performance["model"].astype(str)
            plt.bar(models_str, model_performance["composite_score"])
            plt.xlabel("模型")
            plt.ylabel("綜合評分")
            plt.title("模型效能比較")
            plt.xticks(rotation=45, ha="right") # 旋轉X軸標籤，避免重疊
            plt.tight_layout()
            model_plot_path = os.path.join(self.output_dir,
"model_performance.png")
            plt.savefig(model_plot_path)
            plt.close() # 關閉圖形
            report += f"## 模型效能圖表\n\n![模型效能比較圖表]
({os.path.basename(model_plot_path)})\n\n"
        except Exception as e:
            print(f"生成模型效能圖表失敗: {str(e)}")
            report += "## 模型效能圖表\n\n生成圖表失敗。\n\n"


        # 保存報告到輸出目錄
        report_path = os.path.join(self.output_dir, "evaluation_report.md")
        with open(report_path, "w", encoding="utf-8") as f:
            f.write(report)

        print(f"評估報告已生成到: {report_path}")
        return report

# 如果腳本作為主程序運行，執行評估流程
```

```
if __name__ == "__main__":
    # 這裡提供一個簡單的運行範例
    # 實際運行時建議使用 run_evaluations.py 腳本
    evaluator = PromptEvaluator(
        data_dir="../data", # 假設data目錄在meetsage/scripts的上一層
        prompts_dir="../prompts", # 假設prompts目錄在meetsage/scripts的上一層
        output_dir="../results/manual_run" # 結果輸出到results/manual_run目錄
    )

    print("開始評估所有提示詞在預設會議上的表現...")
    evaluator.evaluate_all_prompts() # 評估所有提示詞在預設會議上

    print("生成評估報告...")
    report = evaluator.generate_report()
    print("評估完成。")
```

—

# 前期準備工作（需連網下載）

## 1. 模型與工具下載

```
bash
# 下載所有需要的模型
ollama pull phi-3-mini
ollama pull gemma-2b
ollama pull llama3-8b
ollama pull taiwan-llm-7b-v2.1
ollama pull taide-7b
ollama pull llama3.1-taide-8b

# Python套件一次性安裝
pip install pandas numpy matplotlib seaborn jieba-tw rouge-chinese bert-score nltk spacy diff-match-patch gitpython pyyaml
python -m spacy download zh_core_web_md
```

## 2. 評估資料集準備

- 收集或準備10-20對「逐字稿-標準會議紀錄」對照組

- 組織為標準目錄結構：

```
/data
  /transcripts   # 逐字稿
    meeting_001.txt
    meeting_002.txt
    ...
  /references    # 標準會議紀錄
    meeting_001.txt
    meeting_002.txt
    ...
```