# 從雜亂到有序：構建一個完美的自動化會議記錄生成系統

## 第一章：解構會議記錄自動化挑戰

在當代企業環境中，高效的時間管理與資訊流通是維持競爭力的關鍵。然而，會議記錄的整理與格式化，這項看似基礎的行政工作，卻往往消耗大量人力資源，且容易因人為疏忽產生錯誤或不一致。將此流程自動化，不僅能釋放寶貴的人力，更能確保企業內部資訊記錄的標準化與準確性，這已成為學術界與業界共同關注的課題 1。本報告旨在提供一個全面、深入且可實踐的技術指南，以應對「如何完美地依據指定的 Word 格式，將雜亂的文本整理成固定的會議記錄，並提供 Word 檔案下載」這一核心需求。

### 1.1 自動化的商業必要性

手動整理會議記錄是一項典型的低附加價值、高重複性的工作。記錄者需要在會議中或會後聆聽錄音，將口語化的討論轉化為書面文字，再 painstakingly 地調整字體、段落、表格，使其符合公司內部的標準範本。此過程不僅耗時，且極易出錯，例如遺漏關鍵決策、誤記負責人，或格式設定不一致。一個自動化的系統能夠從根本上解決這些問題，將非結構化的原始文本（無論是速記草稿還是語音轉文字的初步結果）轉化為一份專業、標準化的官方文件，從而提升整體營運效率。

### 1.2 雙重挑戰：資訊萃取與文件生成

使用者提出的需求可以被精確地分解為兩個獨立但環環相扣的技術挑戰：

1. **資訊萃取 (Information Extraction)**：此階段的核心任務是讓機器「讀懂」非結構化的文本。這意味著需要運用自然語言處理 (NLP) 技術，從一段連續的文字中辨識並抽取出具有特定意義的結構化資訊，例如會議主題、出席者名單、討論要點、決議事項以及最重要的——行動項目與其負責人 3。
2. **文件生成 (Document Generation)**：在成功萃取出結構化數據後，下一個挑戰是將這些數據以程式化的方式填入一個預先定義好的 Microsoft Word 範本中。這不僅僅是簡單的文字替換，還需要能夠動態生成表格、處理列表，並完整保留範本原有的複雜格式，如頁首、頁尾、公司標誌和樣式設定 6。

### 1.3 定義「完美」：從原始文本到無瑕報告

使用者需求中的「完美」一詞，為本解決方案設定了極高的技術標準。這意味著系統不能僅僅是一個脆弱的、基於固定規則的腳本。真實世界的會議記錄草稿充滿了語言的「雜亂性」——句式多變、用詞不一、甚至可能存在錯別字或口語化的表達。例如，指派一個任務，可能被表述為「請王總負責」、「這件事交給王總跟進」、「王總，這個你看一下」，或是更簡潔的「執行：王總」。

一個真正「完美」的系統必須具備足夠的智慧與彈性，以應對這種語言上的不確定性。傳統的規則式方法，如正則表達式 (Regular Expressions)，在面對這種多樣性時會顯得力不從心，任何微小的語言變體都可能導致其失效 8。因此，要達到使用者所期望的健壯性與高準確率，解決方案的核心必須採用更先進的 NLP 模型。這些模型，特別是大型語言模型 (LLM)，能夠超越關鍵字匹配，理解文本的深層語義和上下文，從而準確地識別出不同表述方式背後的相同意圖 10。這正是本報告將重點闡述的技術路徑，也是實現從「雜亂」到「完美」轉化的關鍵所在。

## 第二章：建立數據綱要：萃取的藍圖

在撰寫任何程式碼之前，首要任務是精確定義我們的目標。本章節將深入分析使用者提供的範本文件 (20250901經營會議記錄.pdf) 13，以建立一個明確、完整的數據綱要 (Data Schema)。這個綱要將作為資訊萃取階段的最終產出，以及文件生成階段的唯一輸入，是整個自動化流程中至關重要的「數據合約」。

### 2.1 目標文件的結構剖析

透過對範例會議記錄的詳細檢視，我們可以將其資訊結構進行歸類，以便於後續的數據模型設計。

* **靜態元數據 (Static Metadata)**：這些是單一值的欄位，在每份會議記錄中只會出現一次。
  + 開會事由 (Meeting Subject)
  + 地點 (Location)
  + 時間 (Time)
  + 主席 (Chairperson)
  + 記錄 (Recorder)
* **列表型元數據 (List-based Metadata)**：這些是包含多個值的欄位，通常是人員名單。
  + 出席人員 (Attendees)
  + 請假人員 (Absentees)
  + 列席人員 (Non-voting Attendees)
* **層級式內容 (Hierarchical Content)**：這是文件的主體部分，即「議程」。其結構具有巢狀特性，一個議程可能包含多個大項，每個大項下又包含若干個討論子點，並附帶一個或多個「執行/跟催」項目。這種層級關係必須在數據綱要中得到體現，否則將失去重要的上下文聯繫。例如，「ERP 系統」這個主題下的執行項目，必須與該主題綁定，而不是成為一個孤立的待辦事項。

### 2.2 設計數據合約：結構化中間表示

基於上述分析，我們可以設計一個能夠精確描述會議記錄所有資訊的數據結構。在 Python 中，使用 dataclasses 是一個極佳的選擇，因其能提供類型提示和清晰的結構定義。一個可能的數據綱要定義如下：

Python

from dataclasses import dataclass, field  
from typing import List, Optional  
  
@dataclass  
class ActionItem:  
 """代表一個行動項目及其負責人"""  
 owner: str  
 task\_description: Optional[str] = None # 任務描述可能隱含在主題中  
  
@dataclass  
class AgendaItem:  
 """代表一個議程項目，包含討論點和行動項目"""  
 topic: str  
 discussion\_points: List[str]  
 action\_items: List[ActionItem] = field(default\_factory=list)  
  
@dataclass  
class MeetingMinutes:  
 """代表整份會議記錄的結構化數據"""  
 meeting\_subject: str  
 location: str  
 time: str  
 chairperson: str  
 recorder: str  
 attendees: List[str]  
 absentees: List[str]  
 non\_voting\_attendees: Optional[List[str]] = None  
 agenda: List[AgendaItem] = field(default\_factory=list)  
 approvers: Optional[List[str]] = None

這個綱要的設計直接反映了原始文件的巢狀結構。一個頂層的 MeetingMinutes 物件包含了所有的元數據和一個 AgendaItem 的列表。每個 AgendaItem 又包含了自身的討論要點和一個 ActionItem 列表。這種設計確保了數據的完整性和上下文的保留，是實現精確文件生成的基礎。

### 2.3 結構化中間表示的力量

定義這樣一個嚴謹的數據綱要，其價值遠不止於數據儲存。它在系統架構中扮演著「解耦合」的關鍵角色。

* **對於資訊萃取模組**：該綱要成為一個清晰的萃取目標。無論是使用傳統的 NLP 技術還是先進的 LLM，最終的任務都是填充這個結構化物件。這使得萃取邏輯的開發有了明確的指引。
* **對於文件生成模組**：該綱要成為一個穩定的數據來源。文件生成程式碼的開發者可以確信他們將會收到一個符合此結構的物件，從而可以編寫健壯、可預測的程式碼來遍歷數據並填充 Word 範本。

這種分離使得兩個模組可以獨立開發、測試和升級。例如，未來若決定從 spaCy 模型更換為更先進的 LLM 來進行萃取，只要新的萃取模組仍然能夠產出符合此綱要的數據物件，文件生成模組就無需進行任何修改。同樣地，如果公司決定變更 Word 範本的視覺設計，只要所需數據欄位不變，萃取模組也無需改動。這種架構上的靈活性和可維護性，是構建企業級自動化系統的核心原則。

為了更直觀地展示此數據綱要，下表詳細列出了每個欄位及其對應的描述和範例。

**Table 2.1: 會議記錄數據綱要**

| 欄位名稱 | 數據類型 | 描述 | 範例 13 |
| --- | --- | --- | --- |
| meeting\_subject | string | 會議的主要事由或主題。 | "經營會議" |
| location | string | 會議進行的地點。 | "內湖 3B01會議室" |
| time | string | 會議的日期與時間範圍。 | "2025年09月01日10:00~10:55" |
| chairperson | string | 會議主席的姓名與職稱。 | "王超群董事長" |
| recorder | string | 會議記錄者的姓名。 | "彭美慧" |
| attendees | List[string] | 出席人員的姓名列表。 | ["溫健良", "邵惠周",...] |
| absentees | List[string] | 請假人員的姓名列表。 | ["高永結", "史明龍"] |
| non\_voting\_attendees | List[string] | 列席人員的姓名列表。 | ["湯惠剛"] |
| agenda | List[AgendaItem] | 包含所有議程項目的列表。 | [...] |
| AgendaItem.topic | string | 議程大項的標題。 | "1. ERP系統,優化項目及待完成報表進度:" |
| AgendaItem.discussion\_points | List[string] | 該議程下的具體討論內容或決議。 | ["(1)8/26~27 已召開會議...", "(2)目標於今年底前完成..."] |
| AgendaItem.action\_items | List[ActionItem] | 該議程下的行動項目列表。 | [...] |
| ActionItem.owner | string | 行動項目的執行或跟催負責人/部門。 | "李祥睿助理副總 各部門主管" |

## 第三章：核心引擎：從非結構化文本中萃取結構化數據

本章節是整個解決方案的技術核心，將深入探討如何將原始的中文會議記錄文本，轉換為前一章節定義的結構化數據綱要。我們將介紹一種結合了傳統 NLP 工具與最先進大型語言模型 (LLM) 的混合策略，以實現最高的準確性與健壯性。

### 3.1 準備工作：中文文本的 NLP 環境

為了處理中文文本，我們需要一個功能強大且高效的 NLP 函式庫。spaCy 是一個業界公認的、為生產環境設計的開源函式庫，以其卓越的性能和現代化的 API 設計而著稱 14。

中文語言模型選擇

spaCy 提供了多個預訓練的中文管線 (pipeline)，包括 zh\_core\_web\_sm (小), zh\_core\_web\_md (中), zh\_core\_web\_lg (大), 和 zh\_core\_web\_trf (基於 Transformer) 16。這些模型內建了斷詞、詞性標註 (Part-of-Speech Tagging) 和命名實體辨識 (Named Entity Recognition, NER) 等功能 16。

* **sm/md/lg 模型**：這些是基於卷積神經網絡 (CNN) 的模型，執行速度快，對計算資源要求較低。模型越大，通常包含的詞向量越豐富，準確率也越高。
* **trf 模型**：這個模型基於更先進的 Transformer 架構 (如 BERT)，在理解上下文和處理複雜語義方面表現更佳，因此在 NER 等任務上通常能達到最高的準確度。

對於本任務，由於準確識別人員姓名 (PERSON) 和部門 (ORG) 對於指派行動項目至關重要，因此建議使用 zh\_core\_web\_lg 或 zh\_core\_web\_trf 模型，以犧牲些許處理速度來換取更高的辨識準確率。

### 3.2 針對元數據的混合萃取策略

對於文件中結構相對固定的元數據，可以採用高效的混合策略。

* **規則式方法**：對於像「時間」、「地點」這類具有明顯模式的欄位，可以使用正則表達式進行快速匹配。這是一種計算成本極低且非常可靠的方法 3。
* **命名實體辨識 (NER)**：對於「主席」、「記錄」、「出席人員」等欄位，可以利用 spaCy 的 NER 功能。首先定位到包含這些關鍵詞的行，然後對該行的文本運行 NER 模型，萃取出所有被標記為 PERSON 的實體 18。

Python

import spacy  
  
# 載入高效能的中文模型  
nlp = spacy.load("zh\_core\_web\_lg")  
  
text = "主席: 王超群董事長, 出席人員: 溫健良、邵惠周、廖振翔"  
doc = nlp(text)  
  
# 萃取人名實體  
persons =  
# 輸出可能為: ['王超群', '溫健良', '邵惠周', '廖振翔']

這種混合方法結合了規則的效率和 NER 的靈活性，能夠快速準確地處理文件頭部的元數據。

### 3.3 最先進技術：運用 LLM 進行結構化萃取

對於文件主體中結構複雜、語義多變的議程內容，傳統的 NLP 方法（如依存句法分析）雖然可行，但開發複雜度高且健壯性差。此時，大型語言模型 (LLM) 提供了一種革命性的解決方案 9。

「LLM 即解析器」模式

這個模式的核心思想是，不再編寫複雜的程式碼來解析文本，而是將 LLM 作為一個通用的、可指令的「解析引擎」。其成功的關鍵在於將 LLM 的強大語義理解能力與確定性的數據驗證工具相結合。

1. **使用 Pydantic 強制定義綱要**：Pydantic 是一個 Python 的數據驗證函式庫，它允許我們用簡單的類別來定義數據的結構和類型 20。我們將第二章的數據綱要用 Pydantic 模型來實現。這不僅是為了數據儲存，更重要的是，它將成為驗證 LLM 輸出的「守門員」。
2. **精心設計提示 (Prompt Engineering)**：我們需要建構一個清晰的提示，指示 LLM 如何執行萃取任務。一個高效的提示通常包含以下幾個部分（這種方法被稱為「少樣本提示」Few-Shot Prompting）：
   * **角色設定**：告知 LLM 它扮演的角色，例如「你是一個專業的會議記錄分析助手」。
   * **任務描述**：清晰地說明任務目標，例如「請從以下會議記錄文本中，萃取所有議程項目、討論要點和行動項目，並以指定的 JSON 格式輸出」。
   * **輸出格式定義**：將 Pydantic 模型的 JSON Schema 提供給 LLM，讓它確切地知道需要輸出的欄位名稱、類型和巢狀結構。
   * **高品質範例**：提供一到兩個完整的「輸入文本 -> 輸出 JSON」的範例。這能極大地幫助 LLM 理解任務的細節和邊界情況，顯著提升輸出結果的穩定性和準確性 11。
3. **整合與驗證**：透過 API 呼叫 LLM（例如使用 openai 函式庫），獲取其生成的 JSON 字串。然後，立即嘗試用這個字串來實例化我們的 Pydantic 模型。
   * 如果 JSON 格式正確且所有欄位都符合 Pydantic 模型定義的類型，則實例化成功，我們得到一個乾淨、可靠、類型安全的 Python 物件。
   * 如果 LLM 輸出的 JSON 格式有誤，或者欄位不匹配，Pydantic 會立刻拋出一個驗證錯誤 (ValidationError)。我們可以捕捉這個錯誤，並觸發重試機制，例如將錯誤訊息反饋給 LLM，要求它修正後重新生成 12。

這種「LLM 理解 + Pydantic 驗證」的閉環工作流，巧妙地結合了 LLM 的靈活性和 Pydantic 的嚴謹性。它將 LLM 從一個不穩定的文本生成器，轉變為一個可靠的結構化數據提供者，是解決複雜非結構化資訊萃取問題的當前最佳實踐。

### 3.4 方法論比較

為了讓技術選型更加清晰，下表對比了傳統 NLP 方法與推薦的 LLM 方法在各個維度的表現。

**Table 3.1: 資訊萃取方法論比較**

| 方法論 | 準確率與健壯性 | 實作複雜度 | 營運成本 | 可維護性 | 最適用場景 |
| --- | --- | --- | --- | --- | --- |
| **傳統 NLP (規則+模型)** | 中等。對語言變體較脆弱，規則需窮舉所有可能性。 | 高。需要深厚的語言學知識和複雜的規則編寫，開發週期長。 | 低。模型可在本地端執行，無持續性 API 費用。 | 差。每當出現新的表達方式時，都需要修改程式碼並重新部署。 | 結構高度固定、語義單一的欄位，如日期、時間。 |
| **LLM + Pydantic** | 高至極高。能理解上下文和語義，對語言變體具有很強的適應性。 | 中等。主要工作在於提示工程和 API 整合，而非複雜的演算法編寫。 | 中至高。依賴第三方 API 服務，會產生持續性費用。 | 優。調整提示 (Prompt) 通常比修改和測試複雜的程式碼規則更容易、更快速。 | 結構複雜、語義豐富、表達方式多樣的內容萃取。 |

顯然，對於本報告旨在解決的核心問題——從「雜亂」的議程文本中萃取層級式內容——LLM + Pydantic 的方法在準確率、健壯性和長期可維護性方面具有壓倒性優勢。儘管存在 API 營運成本，但其所節省的開發時間和維護成本，以及帶來的高品質結果，使其成為構建「完美」自動化系統的不二之選。

## 第四章：自動化文件組裝與 python-docx

當我們成功地將非結構化文本轉換為乾淨、結構化的數據物件後，接下來的任務便是將這些數據精確地填入 Microsoft Word 文件範本中。本章節將詳細介紹如何使用 python-docx 函式庫，透過一種高效的「範本化」策略，來完成此一任務。

### 4.1 python-docx 函式庫簡介

python-docx 是 Python 社群中用於創建和修改 .docx 檔案的首選工具 7。它提供了一套完整的 API，讓我們可以程式化地操作 Word 文件的各種元素，包括段落、文字、表格、樣式等 24。

### 4.2 範本化的強大威力

從零開始用程式碼構建一份包含複雜格式（如頁首、頁尾、表格樣式、字體設定）的 Word 文件，是一件極其繁瑣且難以維護的工作。一個更為優雅且高效的策略是「範本化」。

這種策略的核心思想是**將內容與表現分離**。我們首先在 Microsoft Word 中手動創建一個 .docx 檔案作為「範本」。這個範本檔案 (template.docx) 包含了所有靜態的、與格式相關的元素：

* **頁首與頁尾**：例如公司名稱、文件標題、頁碼以及「Confidential」等標示。
* **樣式定義**：預設的字體、字號、顏色、段落間距等。
* **表格結構與樣式**：預先設計好的表格框架，包含正確的欄寬、框線和底色。

python-docx 的一個關鍵特性是，當它開啟一個現有的文件時，它會完整保留所有它無法直接編輯或理解的元素，例如頁首、頁尾的詳細設定 6。這使得範本化策略極其實用。我們可以在範本中為需要動態填入的數據設定簡單的佔位符（例如

{{chairperson}}），然後讓 Python 腳本專注於尋找並替換這些佔位符，以及動態填充表格內容。

這種做法帶來了巨大的維護優勢。如果未來公司的品牌識別系統更新，需要修改文件的頁首 Logo 或標準字體，非技術人員（如行政或市場部門的同事）可以直接在 Word 中修改 template.docx 檔案，而無需開發人員介入修改任何一行 Python 程式碼。這大大降低了系統的長期維護成本，並提高了業務的靈活性。

### 4.3 程式化填充：從數據到文件

以下是將結構化數據填充到 Word 範本的具體步驟與程式碼範例。

1. 載入範本  
   使用 docx.Document 即可輕鬆載入我們預先準備好的範本檔案。  
   Python  
   from docx import Document  
     
   # 載入範本文件  
   document = Document('template.docx')
2. 替換簡單佔位符  
   對於像主席、時間等單一值的元數據，我們可以遍歷文件中的所有段落和文字塊 (run)，尋找並替換佔位符。  
   Python  
   def replace\_placeholder(doc, placeholder, value):  
    """在文件中尋找並替換指定的佔位符"""  
    for paragraph in doc.paragraphs:  
    if placeholder in paragraph.text:  
    # 替換過程需要處理跨 run 的情況，簡化版如下  
    # 完整的實現需要更細緻的 run 遍歷和替換  
    inline = paragraph.runs  
    for i in range(len(inline)):  
    if placeholder in inline[i].text:  
    text = inline[i].text.replace(placeholder, value)  
    inline[i].text = text  
     
   # 從我們的數據物件中獲取數據  
   # meeting\_data =... (從第三章獲得的 MeetingMinutes 物件)  
   replace\_placeholder(document, '{{chairperson}}', meeting\_data.chairperson)  
   replace\_placeholder(document, '{{meeting\_date}}', meeting\_data.time)  
   #... 對其他元數據進行替換
3. 動態建立議程表格  
   這是最核心的部分。我們需要在範本中預留一個表格（可以只有一個標題行），或者在程式碼中找到一個特定的標記位置來插入新表格。假設我們的範本中已經有一個ID為 agenda\_table 的表格。  
   Python  
   # 假設我們能透過某種方式定位到議程表格  
   # 實際應用中可能需要更穩健的定位方式，例如尋找特定標題後的表格  
   table = document.tables # 假設它是文件中第一個表格  
     
   # 遍歷從數據物件中獲取的議程列表  
   for item in meeting\_data.agenda:  
    # 為每個議程項目新增一行  
    row\_cells = table.add\_row().cells  
     
    # 格式化討論要點和行動項目  
    discussion\_text = "\n".join(item.discussion\_points)  
     
    action\_items\_text\_parts =  
    for action in item.action\_items:  
    action\_items\_text\_parts.append(f"{action.owner}")  
    action\_items\_text = "\n".join(action\_items\_text\_parts)  
     
    # 填充儲存格內容  
    # 假設表格有三欄：議程、報告/決議/執行事項、執行/跟催  
    row\_cells.text = item.topic  
    row\_cells.text = discussion\_text  
    row\_cells.text = action\_items\_text  
     
   這段程式碼會根據結構化數據中的 agenda 列表長度，動態地向表格中添加相應數量的行，並將每個議程的詳細資訊填入正確的儲存格中。

### 4.4 儲存最終文件

完成所有內容的填充後，最後一步是將修改後的 Document 物件儲存為一個新的 .docx 檔案。

Python

# 將填充好內容的文件儲存到新檔案  
document.save('output\_meeting\_minutes.docx')

更進一步，為了實現網頁下載功能，python-docx 支援將文件儲存到一個「類檔案物件」(file-like object) 中，例如記憶體中的二進位流，這將在下一章節中詳細介紹 6。

## 第五章：完整工作流程：從原始文本到可下載的報告

本章節將前述所有概念融會貫通，構建一個完整、連貫的端到端應用程式。這個應用程式將作為一個 Web 服務，接收非結構化的會議記錄文本，經過自動化處理後，直接向使用者提供一份格式完美的 Word 文件供下載，從而完整地回應使用者的所有需求。

### 5.1 系統架構概覽

整個自動化系統的數據流可以概括為以下幾個步驟，這構成了一個典型的微服務架構模式：

1. **接收請求**：使用者透過 HTTP POST 請求，將原始的會議記錄文本發送到 Web 服務的指定 API 端點。
2. **啟動服務**：Web 伺服器（我們將使用 Flask）接收請求並調用核心處理邏輯。
3. **資訊萃取**：呼叫第三章中設計的 NLP 萃取服務。該服務接收原始文本，利用 LLM 和 Pydantic 模型，將其轉換為結構化的 MeetingMinutes 數據物件。
4. **文件生成**：將上一步產生的數據物件傳遞給第四章中設計的文件生成服務。該服務載入 Word 範本，並將數據填充進去。
5. **記憶體內儲存**：文件生成後，並非儲存到伺服器的硬碟上，而是直接寫入一個記憶體內的二進位流 (in-memory binary stream)。
6. **回傳檔案**：Web 伺服器構建一個 HTTP 回應，將記憶體中的 Word 文件流作為附件 (attachment) 發送給使用者，觸發瀏覽器的下載行為。

這種架構的優點在於其無狀態 (stateless) 和高效性。由於過程中不涉及對伺服器本地檔案系統的讀寫，系統可以更輕鬆地進行水平擴展，並且避免了管理臨時檔案所帶來的複雜性和潛在的權限問題。

### 5.2 使用 Flask 搭建 Web 服務

Flask 是一個輕量級的 Python Web 框架，非常適合快速搭建 API 服務。以下是一個基本的 Flask 應用程式骨架，用於接收我們的請求。

Python

# app.py  
from flask import Flask, request, send\_file, jsonify  
import io  
  
# 引入我們之前定義的萃取和生成模組  
# from extraction\_service import extract\_minutes\_data  
# from generation\_service import create\_minutes\_document  
  
app = Flask(\_\_name\_\_)  
  
@app.route('/generate\_minutes', methods=)  
def generate\_minutes\_endpoint():  
 #... 完整的處理邏輯將在此實現...  
 pass  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(debug=True)

### 5.3 端到端整合腳本

現在，我們將所有部分整合到 Flask 的 API 端點中。這段程式碼是整個系統的核心協調器。

Python

# app.py (完整版)  
from flask import Flask, request, send\_file, jsonify  
import io  
from docx import Document  
  
# 假設這是我們在第三章和第四章中定義的函數和類別  
# 為了演示，這裡會使用簡化的虛擬實現  
from schema import MeetingMinutes # 假設 schema.py 包含 Pydantic/dataclass 定義  
from extraction\_service import extract\_minutes\_data # 假設此函數返回 MeetingMinutes 物件  
from generation\_service import create\_minutes\_document # 假設此函數接收數據並返回 Document 物件  
  
app = Flask(\_\_name\_\_)  
  
@app.route('/generate\_minutes', methods=)  
def generate\_minutes\_endpoint():  
 """  
 接收原始文本，生成並回傳會議記錄 Word 檔案。  
 """  
 # 1. 獲取原始文本輸入  
 raw\_text = request.data.decode('utf-8')  
 if not raw\_text:  
 return jsonify({"error": "Request body is empty"}), 400  
  
 try:  
 # 2. 呼叫資訊萃取服務  
 # structured\_data: MeetingMinutes = extract\_minutes\_data(raw\_text)  
 # --- 為了演示，使用虛擬數據 ---  
 from schema import ActionItem, AgendaItem  
 structured\_data = MeetingMinutes(  
 meeting\_subject="經營會議",  
 location="內湖 3B01會議室",  
 time="2025年09月01日10:00~10:55",  
 chairperson="王超群董事長",  
 recorder="彭美慧",  
 attendees=["溫健良", "邵惠周", "廖振翔"],  
 absentees=["高永結", "史明龍"],  
 non\_voting\_attendees=["湯惠剛"],  
 agenda=,  
 action\_items=[ActionItem(owner="李祥睿助理副總 各部門主管")]  
 ),  
 AgendaItem(  
 topic="2. AI應用事項:",  
 discussion\_points=["(1) 外部一系列講座,截至目前已舉辦七場", "(3) AI策略規劃師認證:8月有2名同仁參加並通過考試"],  
 action\_items=[ActionItem(owner="湯惠剛資深副總"), ActionItem(owner="陳文彬資深協理")]  
 )  
 ]  
 )  
 # --- 虛擬數據結束 ---  
  
 # 3. 呼叫文件生成服務  
 # document: Document = create\_minutes\_document(structured\_data, 'template.docx')  
 # --- 為了演示，使用虛擬生成邏輯 ---  
 document = Document('template.docx') # 假設有一個範本  
 # 這裡應包含第四章的佔位符替換和表格填充邏輯  
 #...  
 document.add\_paragraph(f"主題：{structured\_data.meeting\_subject}")  
 document.add\_paragraph(f"主席：{structured\_data.chairperson}")  
 # --- 虛擬生成邏輯結束 ---  
  
  
 # 4. 啟用下載功能  
 return serve\_document(document)  
  
 except Exception as e:  
 # 實際應用中應加入更詳細的錯誤日誌  
 return jsonify({"error": f"An error occurred: {str(e)}"}), 500  
  
def serve\_document(document: Document):  
 """  
 將 docx Document 物件轉換為可供下載的 HTTP 回應。  
 """  
 # 5. 將文件儲存到記憶體流  
 file\_stream = io.BytesIO()  
 document.save(file\_stream)  
 file\_stream.seek(0) # 將指標移至流的開頭，準備讀取  
  
 # 6. 構建 Flask 回應以觸發下載  
 return send\_file(  
 file\_stream,  
 as\_attachment=True,  
 download\_name='meeting\_minutes.docx',  
 mimetype='application/vnd.openxmlformats-officedocument.wordprocessingml.document'  
 )  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # 在生產環境中，應使用 Gunicorn 或 uWSGI 等 WSGI 伺服器  
 app.run(host='0.0.0.0', port=5000)

### 5.4 實現文件下載的關鍵

實現「下載」功能而非僅僅在伺服器端生成檔案的關鍵，在於上述程式碼的 serve\_document 函數。

1. **使用 io.BytesIO**：python-docx 的 document.save() 方法不僅可以接受檔案路徑，還可以接受任何「類檔案物件」 6。  
   io.BytesIO 正是這樣一個物件，它在記憶體中模擬了一個二進位檔案。我們將生成的 Word 文件內容直接寫入這個記憶體緩衝區。
2. **設置 send\_file 參數**：Flask 的 send\_file 函數是專門用來發送檔案的。
   * as\_attachment=True：這個參數是核心。它會在 HTTP 回應的標頭 (Header) 中加入 Content-Disposition: attachment，這會告訴瀏覽器不要嘗試在頁面內顯示這個檔案，而是將其作為附件處理，從而彈出下載對話框。
   * download\_name='meeting\_minutes.docx'：這指定了使用者下載時預設的檔案名稱。
   * mimetype：這指定了檔案的媒體類型 (MIME Type)。對於 .docx 檔案，正確的 MIME 類型是 application/vnd.openxmlformats-officedocument.wordprocessingml.document。這有助於瀏覽器和作業系統正確識別檔案類型。

透過這種方式，我們構建了一個專業且可擴展的解決方案，它將複雜的 NLP 處理和文件操作封裝在一個簡單的 API 背後，完美地實現了使用者從輸入原始文本到下載格式化 Word 報告的全部需求。

## 第六章：進階議題與系統的未來展望

成功構建一個可運作的原型系統只是第一步。為了將其部署到生產環境並確保長期穩定運行，還需要考慮錯誤處理、持續優化、效能擴展等進階議題。本章節將探討如何加固我們的解決方案，並展望其未來的發展潛力。

### 6.1 系統的健壯性與錯誤處理

一個生產級的系統必須能夠優雅地處理各種預期之外的狀況。

* **萃取失敗的應對策略**：LLM 的輸出並非 100% 穩定。有時它可能生成格式錯誤的 JSON，或者其內容無法通過 Pydantic 的驗證。
  + **重試機制**：當 Pydantic 驗證失敗時，不應直接向使用者回報錯誤。可以實作一個帶有重試邏輯的機制。例如，捕捉 ValidationError，將錯誤訊息（例如「缺少 chairperson 欄位」）作為新的上下文資訊，再次請求 LLM 進行修正。
  + **回退機制 (Fallback)**：如果多次重試後 LLM 仍然無法提供有效的輸出，系統可以回退到一個更簡單的模式，例如僅萃取議程標題，或者直接通知人工介入，而不是完全崩潰。
* **輸入驗證**：在將文本發送給成本較高的 LLM API 之前，應進行初步的輸入驗證。檢查輸入文本是否為空、長度是否過短，或者是否包含基本的會議記錄關鍵詞（如「會議」、「主席」、「議程」等），以過濾掉明顯無效的請求。

### 6.2 準確率的持續改善與維護

模型的準確率並非一成不變，需要持續的維護和優化。

* **提示工程 (Prompt Engineering)**：LLM 的表現與提示的品質高度相關。隨著系統在實際應用中遇到越來越多的邊界案例（edge cases），應持續迭代和優化我們的提示。例如，如果發現模型經常將「建議」誤判為「決議」，可以在提示的範例中加入一個明確區分這兩者的例子。
* **少樣本學習 (Few-Shot Learning) 的威力**：在提示中提供高品質、多樣化的範例，是引導模型產生正確輸出的最有效方法之一。應建立一個小型的「黃金標準」範例集（例如 5-10 個），涵蓋各種不同的會議類型和語言風格。當模型表現不佳時，首先應檢查和擴充這個範例集 11。

### 6.3 擴展性與效能考量

當系統需要處理大量請求時，效能和擴展性成為主要考量。

* **非同步處理**：LLM API 的呼叫通常是 I/O 密集型操作，可能需要數秒鐘才能返回結果。將 Flask 應用程式部署在支援非同步工作模式的 WSGI 伺服器（如 Gunicorn 搭配 gevent 或 uvicorn）上，可以顯著提高系統的吞吐量，使其能同時處理多個請求。
* **結果快取 (Caching)**：如果存在重複處理相同會議記錄文本的可能性，可以引入快取機制。將原始文本的雜湊值 (hash) 作為鍵，將萃取出的結構化數據作為值，存儲在 Redis 等記憶體資料庫中。在處理新請求時，首先檢查快取，可以避免不必要的 API 呼叫，降低成本和延遲。

### 6.4 學術前沿與未來功能

本報告所構建的系統是一個堅實的基礎，而會議記錄自動化領域仍在不斷發展。未來的增強功能可以從以下幾個方向探索：

* **抽象式摘要 (Abstractive Summarization)**：目前的系統是「萃取式」的，即直接從原文中提取句子。未來的系統可以利用更先進的摘要模型，生成更為流暢、連貫的「抽象式」摘要，即用模型自己的語言來總結會議要點 1。
* **情緒分析 (Sentiment Analysis)**：可以整合情緒分析模型，分析會議中不同發言者對特定議題的態度（正面、負面、中立），為管理者提供超越文字內容的深層洞察 26。
* **發言人日誌 (Speaker Diarization)**：若輸入源是會議錄音，可以結合語音辨識和發言人辨識技術，自動標記每一段發言的歸屬，生成帶有發言人標籤的逐字稿，從而可以分析特定人員的觀點和承諾 27。

## 結論

本報告詳細闡述了一個從零開始構建全自動化會議記錄生成系統的完整技術藍圖。該系統的核心在於採用了一種結合大型語言模型 (LLM) 與確定性數據驗證的先進架構，以應對處理非結構化中文文本的複雜挑戰。

**核心建議與實踐路徑如下：**

1. **架構先行，數據為本**：在動手編寫程式碼之前，必須先透過深入分析目標文件，定義一個嚴謹、層級化的數據綱要。這個綱要不僅是系統的藍圖，更是解耦資訊萃取與文件生成這兩大模組的關鍵。
2. **擁抱 LLM，兼顧傳統**：對於語義豐富、結構多變的議程內容，推薦採用「LLM 即解析器」的模式。透過精心設計的提示工程和 Pydantic 的強制驗證，可以將 LLM 轉變為一個穩定、可靠的結構化數據來源。同時，對於格式固定的元數據，可輔以正則表達式和傳統 NER 技術以提高效率。
3. **內容與表現分離**：在文件生成階段，應採用「範本化」策略。利用 python-docx 函式庫讀取一個預先設計好樣式的 Word 範本，僅以程式碼填充內容。這種做法極大地提升了系統的長期可維護性，使得視覺格式的變更無需開發人員介入。
4. **服務化交付**：為了滿足使用者對「檔案下載」的需求，整個解決方案應被封裝在一個輕量級的 Web 服務中。透過在記憶體中生成文件並直接以 HTTP 回應流的形式發送，可以構建一個高效、無狀態且易於擴展的應用程式。

總而言之，將雜亂的會議記錄文本自動轉換為格式完美的 Word 文件，是一個典型的、融合了多種 NLP 技術和軟體工程實踐的複雜問題。遵循本報告提出的架構和方法，開發者不僅能夠解決當前的具體需求，更能構建一個健壯、可擴展、可維護的企業級自動化解決方案，為組織帶來切實的效率提升。

#### 引用的著作

1. (PDF) Automatic Generation of Minutes of Meetings - ResearchGate, 檢索日期：9月 12, 2025， <https://www.researchgate.net/publication/360456552_Automatic_Generation_of_Minutes_of_Meetings>
2. Voice Meeting Summarization - RJPN, 檢索日期：9月 12, 2025， <https://rjpn.org/ijcspub/papers/IJCSP23B1312.pdf>
3. NLP Libraries in Python - GeeksforGeeks, 檢索日期：9月 12, 2025， <https://www.geeksforgeeks.org/nlp/nlp-libraries-in-python/>
4. Navigating a Python NLP Library: What You Need to Know - Coursera, 檢索日期：9月 12, 2025， <https://www.coursera.org/articles/python-nlp-library>
5. Text Analysis in Python: Techniques and Libraries Explained - Airbyte, 檢索日期：9月 12, 2025， <https://airbyte.com/data-engineering-resources/text-analysis-in-python>
6. Working with Documents - python-docx - Read the Docs, 檢索日期：9月 12, 2025， <https://python-docx.readthedocs.io/en/latest/user/documents.html>
7. python-docx — python-docx 1.2.0 documentation - Read the Docs, 檢索日期：9月 12, 2025， <https://python-docx.readthedocs.io/>
8. Scrape unstructured data and transform to structured : r/Python - Reddit, 檢索日期：9月 12, 2025， <https://www.reddit.com/r/Python/comments/175w2n7/scrape_unstructured_data_and_transform_to/>
9. Which methods use to extract specific info on unstructured text? : r/learnmachinelearning, 檢索日期：9月 12, 2025， <https://www.reddit.com/r/learnmachinelearning/comments/17jey8s/which_methods_use_to_extract_specific_info_on/>
10. Introducing LangExtract: A Gemini powered information extraction library, 檢索日期：9月 12, 2025， <https://developers.googleblog.com/en/introducing-langextract-a-gemini-powered-information-extraction-library/>
11. google/langextract: A Python library for extracting structured ... - GitHub, 檢索日期：9月 12, 2025， <https://github.com/google/langextract>
12. Building an AI Agent for Parsing: A Practical Guide with Python Code | by Jay Kim | Medium, 檢索日期：9月 12, 2025， <https://medium.com/@bravekjh/building-an-ai-agent-for-parsing-a-practical-guide-with-python-code-1aeb2b66e3e5>
13. 20250901經營會議記錄.pdf
14. spaCy · Industrial-strength Natural Language Processing in Python, 檢索日期：9月 12, 2025， <https://spacy.io/>
15. Natural Language Processing With spaCy in Python, 檢索日期：9月 12, 2025， <https://realpython.com/natural-language-processing-spacy-python/>
16. Models & Languages · spaCy Usage Documentation, 檢索日期：9月 12, 2025， <https://spacy.io/usage/models>
17. Chinese · spaCy Models Documentation, 檢索日期：9月 12, 2025， <https://spacy.io/models/zh>
18. www.analyticsvidhya.com, 檢索日期：9月 12, 2025， <https://www.analyticsvidhya.com/blog/2021/06/nlp-application-named-entity-recognition-ner-in-python-with-spacy/#:~:text=Named%20Entity%20Recognition%20(NER)%20is,spaCy%2C%20NLTK%2C%20and%20StanfordNLP.>
19. Named Entity Recognition - GeeksforGeeks, 檢索日期：9月 12, 2025， <https://www.geeksforgeeks.org/nlp/named-entity-recognition/>
20. Extracting Structured Data from Unstructured Text Using Dynamic Techniques - Medium, 檢索日期：9月 12, 2025， <https://medium.com/@lucasdacunh/extracting-structured-data-from-unstructured-text-using-dynamic-techniques-f8a4d9e4b9fa>
21. AI-Powered Information Extraction and Matchmaking - Towards Data Science, 檢索日期：9月 12, 2025， <https://towardsdatascience.com/ai-powered-information-extraction-and-matchmaking-0408c93ec1b9/>
22. python-openxml/python-docx: Create and modify Word documents with Python - GitHub, 檢索日期：9月 12, 2025， <https://github.com/python-openxml/python-docx>
23. python-docx - PyPI, 檢索日期：9月 12, 2025， <https://pypi.org/project/python-docx/>
24. Python-docx: A Comprehensive Guide to Creating and Manipulating Word Documents in Python | by Manoj Das | Medium, 檢索日期：9月 12, 2025， <https://medium.com/@HeCanThink/python-docx-a-comprehensive-guide-to-creating-and-manipulating-word-documents-in-python-a765cf4b4cb9>
25. Automated Generation of Meeting Minutes using Deep Learning Techniques, 檢索日期：9月 12, 2025， <https://www.semanticscholar.org/paper/Automated-Generation-of-Meeting-Minutes-using-Deep-Manuel-Menon/0bb3fa5f26961663d8acff09e9323c6dc478d6d4>
26. Natural Language Processing in Meetings: Understanding Stakeholder Sentiment | adam.ai, 檢索日期：9月 12, 2025， <https://adam.ai/blog/nlp-meetings-stakeholder-sentiment>
27. Minutes of Meeting Generation for Online Meetings Using NLP & ML Techniques - SciSpace, 檢索日期：9月 12, 2025， <https://scispace.com/papers/minutes-of-meeting-generation-for-online-meetings-using-nlp-5a3o76lo2d>