

GPT-RAG 需求解決方案討論

1-1. RAG 正確性與 GenAI 創造性的平衡

需求摘要

Agent 需同時具備兩種矛盾特質：

- **嚴謹性**：回答正確資料，不產生幻覺
- **創造性**：協助推理預測、自由發揮創意

解決方案

方案 A : System Prompting (主要方法)

透過 System Prompt 設計，讓 Agent 根據問題類型自動切換回答模式。

Prompt 設計重點：

回答模式分類：

1. 事實查詢模式 (Factual Mode)

- 當使用者詢問現況、數據、進度、規格等事實性問題時
- 嚴格基於檢索結果回答
- 必須引用來源文件
- 若資料不足，明確告知「根據現有資料無法確認」
- 不可臆測或補充未經證實的資訊

2. 創意發想模式 (Creative Mode)

- 當使用者詢問規劃建議、商業構想、預測分析、腦力激盪時
- 先陳述檢索到的事實基礎
- 明確標示「以下為創意建議/推測」
- 可自由發揮創意、提供多元觀點
- 鼓勵使用者進一步討論

判斷依據：

- 關鍵字包含「多少」、「何時」、「目前」、「進度」、「狀態」 → 事實查詢模式
- 關鍵字包含「如何」、「建議」、「可以」、「未來」、「可能」、「創意」 → 創意發想模式

Pros	Cons
<input checked="" type="checkbox"/> 實作最簡單，僅需修改 Prompt	<input checked="" type="checkbox"/> 依賴 LLM 判斷能力
<input checked="" type="checkbox"/> 不需改動程式碼架構	<input checked="" type="checkbox"/> 邊界情況可能判斷錯誤
<input checked="" type="checkbox"/> 可快速迭代調整	

方案 B : Temperature 動態調整

根據問題類型動態調整 Temperature 參數：

- 事實查詢 : Temperature = 0.1 (低)
- 創意發想 : Temperature = 0.7-0.9 (高)

實作方式：在 Orchestrator 中加入意圖偵測，根據結果設定不同 Temperature。

Pros	Cons
<input checked="" type="checkbox"/> 技術上可精確控制創造性程度	<input checked="" type="checkbox"/> 需修改 Orchestrator 程式碼
<input checked="" type="checkbox"/> 效果明顯	<input checked="" type="checkbox"/> 需先有意圖分類機制

方案 C : 回應格式強制區分

強制要求 Agent 將「事實」與「建議」分段呈現，讓使用者清楚區分。

Prompt 設計重點：

對於每個問題，依照以下結構回答：

1.  事實資訊 (基於檢索結果)
 - 嚴格引用文件內容，標註來源
2.  延伸建議 (創意發想)
 - ⚠ 以下內容為 AI 根據上述資訊的推測與建議，僅供參考
 - 在此自由發揮創意、提供商業構想、預測分析等

Pros	Cons
<input checked="" type="checkbox"/> 使用者可清楚區分事實與建議	<input checked="" type="checkbox"/> 回應長度增加
<input checked="" type="checkbox"/> 降低幻覺造成的誤解風險	<input checked="" type="checkbox"/> 部分問題可能不適用此格式
<input checked="" type="checkbox"/> 不需改程式碼	

1-1 方案比較總表

方案	實作難度	效果	建議優先序
A: System Prompting	低	中高	★ 1st
C: 回應格式區分	低	中高	★ 1st
B: Temperature 動態調整	中	高	2nd

1-2. 多模態 (Multi-modal)

需求摘要

需求	描述
檔案格式	Word (.docx)、PowerPoint (.pptx)、Excel (.xlsx)、PDF、圖片
圖文結合	處理文件中的圖片、表格與文字的關聯性

現有架構分析

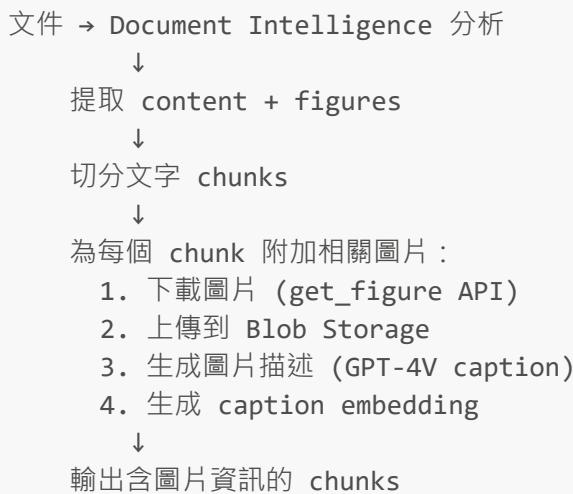
1. Azure Document Intelligence 整合 (已實作)

支援的檔案格式 (API 4.0+) :

格式	支援狀態	說明
PDF	<input checked="" type="checkbox"/>	支援 OCR 高解析度
Word (.docx)	<input checked="" type="checkbox"/>	API 4.0+
PowerPoint (.pptx)	<input checked="" type="checkbox"/>	API 4.0+
Excel (.xlsx)	<input checked="" type="checkbox"/>	API 4.0+
圖片 (jpg, png, bmp, tiff)	<input checked="" type="checkbox"/>	原生支援
HTML	<input checked="" type="checkbox"/>	API 4.0+

2. MultimodalChunker (圖文處理 - 已實作)

處理流程：



圖片處理流程：

1. 圖片提取：Document Intelligence 自動識別文件中的 figures
2. 圖片儲存：上傳至 **documents-images** container
3. 圖片描述：使用 GPT-4V 生成 caption (最多 200 字)
4. 向量化：caption 轉換為 embedding · 支援語意搜尋

3. 表格處理 (已實作)

- Document Intelligence 自動識別 HTML 表格
- 表格在 chunking 時會被保留完整結構
- Markdown 輸出格式保持表格可讀性

💡 現有功能 vs 需求對照

需求	現有支援	說明
Word 圖文結合	<input checked="" type="checkbox"/> 部分支援	文字提取完整，內嵌圖片不支援
PowerPoint 圖文表格	<input checked="" type="checkbox"/> 部分支援	文字提取完整，內嵌圖片不支援
Excel 報表	<input checked="" type="checkbox"/> 完整支援	每個 worksheet = 1 page unit
PDF 圖文結合	<input checked="" type="checkbox"/> 完整支援	支援 OCR + figures 提取
圖片檔案	<input checked="" type="checkbox"/> 完整支援	直接 OCR + caption
理解圖文關聯性	<input checked="" type="checkbox"/> 已實作	caption + 同 chunk 關聯

⚠ 重要限制

根據 Microsoft 官方文件：

Office file types (DOCX, XLSX, PPTX):

- **Embedded or linked images aren't supported**
- 僅提取文字內容，不處理內嵌圖片

檔案類型	文字提取	圖片提取	表格提取
PDF	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
圖片	<input checked="" type="checkbox"/>	N/A	<input checked="" type="checkbox"/>
Word (.docx)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
PowerPoint (.pptx)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Excel (.xlsx)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

💡 解決方案

方案 A：轉換為 PDF (建議)

作法：要求使用者將 Word/PowerPoint 轉存為 PDF 後再上傳

流程：

Word/PPT → 使用者轉存 PDF → 上傳 → Document Intelligence (完整圖文處理)

Pros	Cons
<input checked="" type="checkbox"/> 完整支援圖文提取	<input type="checkbox"/> 需要使用者額外操作
<input checked="" type="checkbox"/> 不需修改程式碼	<input type="checkbox"/> 無法完全自動化
<input checked="" type="checkbox"/> 成本效益最高	

方案 B：伺服器端自動轉換

作法：在 Ingestion 前自動將 Office 檔案轉換為 PDF

技術選項：

1. **LibreOffice Headless**：開源免費，容器化部署
2. **Microsoft Graph API**：需要 Microsoft 365 訂閱
3. **Aspose.Words/Slides**：商用授權

Pros	Cons
<input checked="" type="checkbox"/> 使用者無感	<input type="checkbox"/> 需額外部署轉換服務
<input checked="" type="checkbox"/> 完整圖文支援	<input type="checkbox"/> 增加處理時間與成本
<input type="checkbox"/> 轉換品質可能有差異	

方案 C：Azure Content Understanding (進階選項)

作法：使用 Azure AI Content Understanding 服務 (Preview)

功能	Document Intelligence	Content Understanding
跨頁表格	<input type="checkbox"/> 單頁	<input checked="" type="checkbox"/> 支援
語意分段	<input type="checkbox"/> 段落邊界	<input checked="" type="checkbox"/> 語意 chunking
Office 圖片	<input type="checkbox"/> 不支援	<input checked="" type="checkbox"/> 支援
定價	較低	較高

Pros	Cons
<input checked="" type="checkbox"/> 更完整的多模態支援	<input type="checkbox"/> Preview 階段，功能可能變更
<input checked="" type="checkbox"/> 語意 chunking	<input type="checkbox"/> 成本較高
<input checked="" type="checkbox"/> 跨頁表格處理	<input type="checkbox"/> 需重新整合架構

1-2 方案比較總表

需求	推薦方案	實作難度	效益
Office 圖文處理	方案 A: 轉 PDF	低	高

需求	推薦方案	實作難度	效益
自動化圖文處理	方案 B: LibreOffice	中	高
進階多模態	方案 C: Content Understanding	高	最高

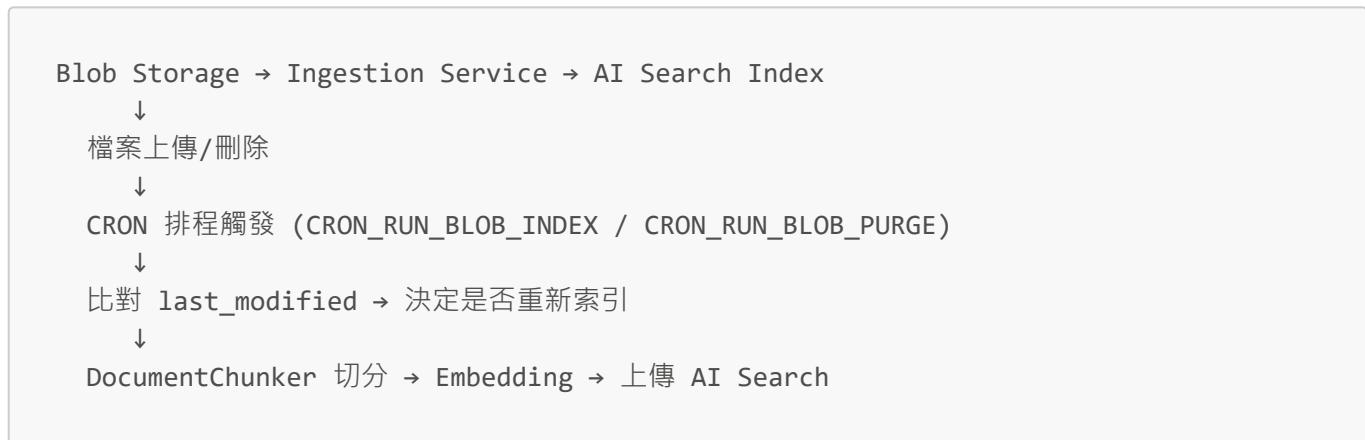
1-3. 檔案管理

需求摘要

需求	情境描述
檔案管理	如何讓使用者「下架」檔案，使其不再被 RAG 查找到？

現有架構分析

1. 目前的 Ingestion 流程



2. 現有的刪除機制

- **Blob Purger** (`CRON_RUN_BLOB_PURGE`)：定期掃描 AI Search Index，刪除對應 Blob 已不存在的 chunks
- **刪除邏輯**：比對 `parent_id` (格式: `/{container}/{blob_path}`)，若 Blob 不存在則刪除所有相關 chunks

3. 現有的 Metadata 支援

AI Search Index Schema :

欄位	用途
<code>metadata_storage_last_modified</code>	檔案最後修改時間
<code>metadata_storage_path</code>	檔案路徑 (<code>parent_id</code>)
<code>metadata_security_id</code>	權限控制 ID 列表

解決方案討論

檔案下架方案

方案 A：刪除 Blob 檔案 (現有機制)

作法：直接從 Blob Storage 刪除檔案，等待 Purger 清理 Index

流程：

使用者刪除 Blob → Purger CRON 執行 → 掃描發現 Blob 不存在 → 刪除 Index chunks

Pros	Cons
<input checked="" type="checkbox"/> 已內建，無需開發	<input type="checkbox"/> 非即時 (依賴 CRON 間隔)
<input checked="" type="checkbox"/> 完全移除資料	<input type="checkbox"/> 無法保留歷史記錄
<input checked="" type="checkbox"/> 簡單直覺	<input type="checkbox"/> 刪除後無法復原

方案 B：Soft Delete (新增 metadata 欄位)

作法：新增 `metadata_status` 欄位，標記為 `archived` 而非實際刪除

實作變更：

1. **AI Search Schema** 新增欄位：`metadata_status` (type: String, filterable: true, default: "active")
2. **查詢時加入 Filter**：`metadata_status eq 'active'`
3. **下架 API**：更新該檔案所有 chunks 的 `metadata_status` 為 `archived`

Pros	Cons
<input checked="" type="checkbox"/> 即時生效	<input type="checkbox"/> 需修改 Schema + Orchestrator
<input checked="" type="checkbox"/> 可復原 (改回 active)	<input type="checkbox"/> Index 仍佔用空間
<input checked="" type="checkbox"/> 保留歷史記錄	<input type="checkbox"/> 需新增管理 API
<input checked="" type="checkbox"/> 不影響現有 Blob	

方案 C：Security ID 權限控制 (現有機制擴展)

作法：利用現有 `metadata_security_id` 欄位，移除使用者的存取權限

流程：

使用者「下架」→ 將 `security_id` 改為 "`archived-{timestamp}`"
 → 查詢時不會匹配任何使用者權限

Pros	Cons
<input checked="" type="checkbox"/> 已有 metadata 欄位	<input type="checkbox"/> 語意不清 (權限 vs 狀態混用)
<input checked="" type="checkbox"/> 支援細粒度控制	<input type="checkbox"/> 需要前端配合傳遞 security filter
	<input type="checkbox"/> 需重新觸發 Ingestion 更新 metadata

1-4. 版本控管

需求摘要

需求	情境描述
版本控管	同檔覆蓋、版本後綴、跨檔案資訊衝突的處理

情境 1：同檔覆蓋

現有行為：當 `blob.last_modified > index.metadata_storage_last_modified` 時會重新索引，並刪除舊 chunks 後上傳新 chunks。

問題：舊版本完全被覆蓋，無歷史記錄

方案 A : Blob Versioning (Azure 原生)

作法：啟用 Azure Blob Storage 的版本控制功能

優點：

- Azure 原生支援，自動保留所有版本
- 不影響現有 Ingestion 流程
- 可透過 Azure Portal/API 瀏覽歷史

缺點：

- 舊版本不會被 RAG 索引 (僅保留 current version)
- 儲存成本增加

建議：搭配定期清理策略 (Lifecycle Management)

方案 B : Archive Folder (手動歸檔)

作法：覆蓋前將舊版移至 `archive/` 目錄

```
documents/
└── 工程進度.docx      ← 最新版 (被 RAG 索引)
    └── archive/
        └── 工程進度_20250115.docx ← 舊版 (不被索引)
```

實作：修改 Ingestion 設定，排除 `archive/` prefix

Pros	Cons
<input checked="" type="checkbox"/> 簡單易懂	<input type="checkbox"/> 需要前端/使用者配合移動檔案
<input checked="" type="checkbox"/> 舊版仍可存取	<input type="checkbox"/> 手動操作易出錯
<input checked="" type="checkbox"/> 不需改 Ingestion 程式	<input type="checkbox"/> 無自動化

方案 C：版本 Metadata + 時間排序

作法：新增 `effective_date` metadata，查詢時依時間排序

Schema 新增：`effective_date` (type: DateTimeOffset, sortable: true, filterable: true)

Ingestion 修改：從 Blob metadata 讀取 `effective_date`，預設為 `last_modified`

查詢調整：`orderby="effective_date desc"`

Pros	Cons
<input checked="" type="checkbox"/> 可自動依時間排序	<input type="checkbox"/> 需修改 Schema + Ingestion
<input checked="" type="checkbox"/> 支援跨檔案時間比較	<input type="checkbox"/> 使用者需手動設定 metadata
<input checked="" type="checkbox"/> 語意清晰	<input type="checkbox"/> 不解決同檔覆蓋問題

情境 2：版本後綴 (`_v2`, `_v3`)

問題：`工程進度_v2.pptx` 和 `工程進度_v3.pptx` 被視為獨立檔案

方案 A：Document Family ID

作法：新增 `document_family_id` 欄位，同一文件的不同版本共用相同 family ID

Metadata 設定：上傳時標記 `document_family_id = "工程進度"`

查詢邏輯：

1. 搜尋時先找相關 chunks
2. 對同一 `document_family_id` 的結果，只保留 `effective_date` 最新的

Pros	Cons
<input checked="" type="checkbox"/> 明確關聯不同版本	<input type="checkbox"/> 需使用者手動標記
<input checked="" type="checkbox"/> 可彈性處理版本關係	<input type="checkbox"/> 需修改 Orchestrator 查詢邏輯

方案 B：檔名正規化 + 自動偵測

作法：Ingestion 時自動解析檔名，提取 base name 和版本

解析邏輯：匹配 `_v1`, `_v2`, `_2025-01-15`, `_20250115` 等後綴

Pros	Cons
<input checked="" type="checkbox"/> 自動化	<input checked="" type="checkbox"/> 依賴命名規則
<input checked="" type="checkbox"/> 不需使用者額外操作	<input checked="" type="checkbox"/> 誤判風險
<input checked="" type="checkbox"/> 需修改 Ingestion 程式	

情境 3：跨檔案資訊衝突

問題：`規劃.pptx` 和 `營運.pptx` 對同一事項有不同描述

方案 A：時間戳優先排序 (建議)

作法：

- 所有 chunks 包含 `effective_date` 或 `metadata_storage_last_modified`
- Orchestrator 的 RAG Prompt 加入指引：

Prompt 設計重點：

- 當多個來源對同一事項有不同描述時：
 - 優先採用較新日期的資訊
 - 明確告知使用者資訊來源與日期
 - 若有明顯矛盾，列出所有版本供使用者判斷

Pros	Cons
<input checked="" type="checkbox"/> 不需複雜技術實作	<input checked="" type="checkbox"/> 依賴 LLM 判斷能力
<input checked="" type="checkbox"/> 保留完整資訊供使用者判斷	<input checked="" type="checkbox"/> 可能需要更長的回應

方案 B：知識圖譜 (Knowledge Graph)

作法：建立實體-屬性-時間的知識圖譜，追蹤資訊變化

範例：

```
Entity: 展演廳燈具
└─ [2025-01-10] 數量=200 (來源: 規劃.pptx)
└─ [2025-01-15] 狀態=待重新商議 (來源: 營運.pptx)
```

Pros	Cons
<input checked="" type="checkbox"/> 精確追蹤資訊演變	<input checked="" type="checkbox"/> 實作複雜度極高

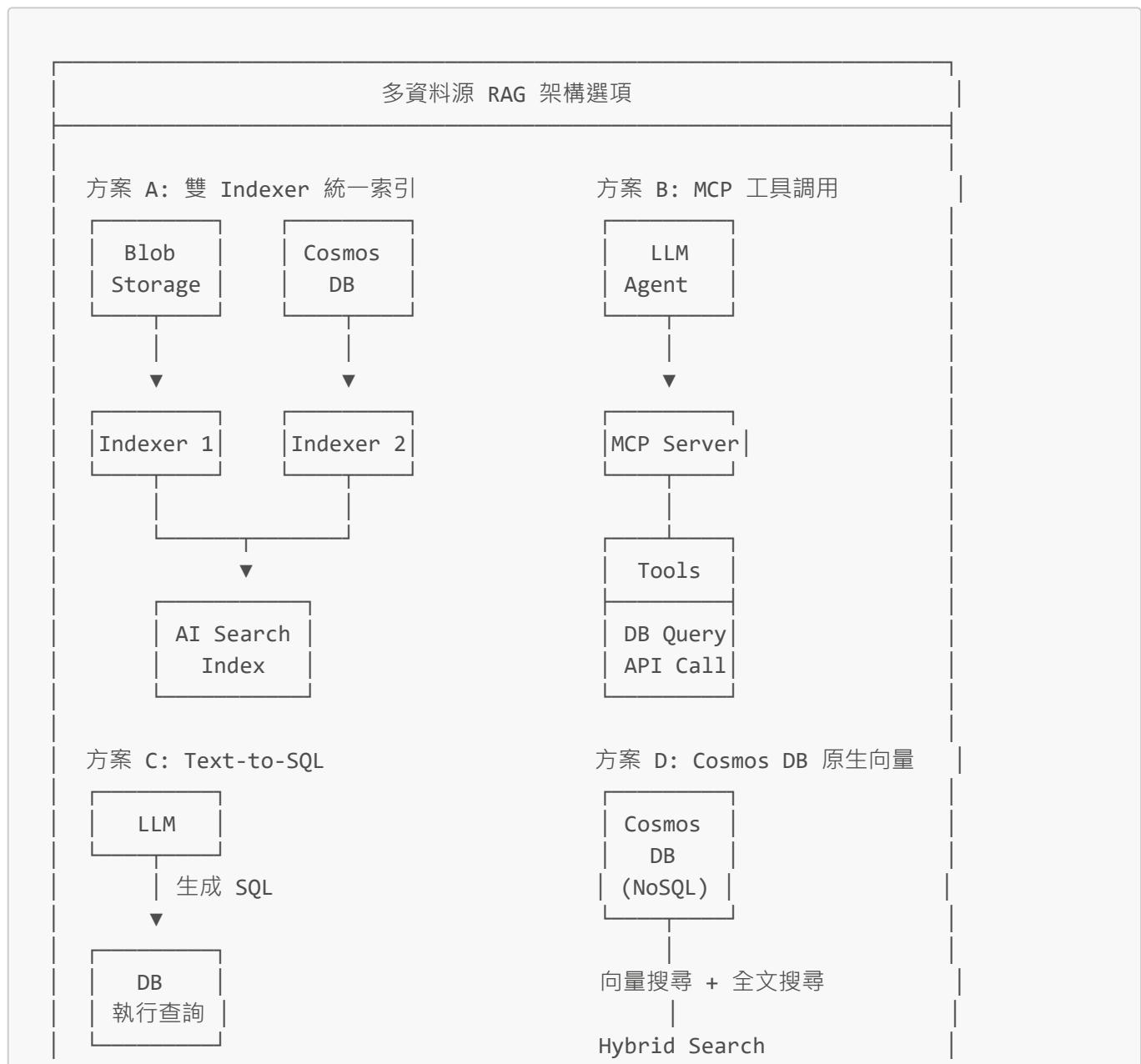
Pros	Cons
<input checked="" type="checkbox"/> 可回答「某資訊的歷史」	<input checked="" type="checkbox"/> 需大量客製化開發
	<input checked="" type="checkbox"/> 超出 GPT-RAG 現有架構

1-5. 多資料源整合策略 (Blob Storage + Database)

需求摘要

情境	描述
資料源 A	Blob Storage (非結構化文件 : Word、PDF、PPT 等)
資料源 B	Database (結構化資料 : Cosmos DB、SQL Server 等)
目標	讓 RAG 能同時查詢兩種資料源的資訊

方案概覽

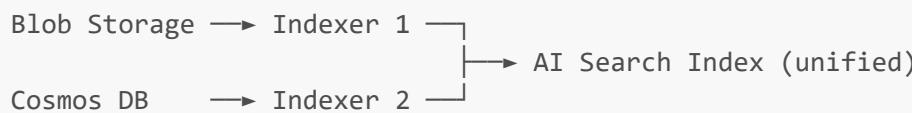


💡 解決方案詳述

方案 A：雙 Indexer 統一索引 推薦

作法：建立兩個 AI Search Indexer，分別對 Blob Storage 和 Cosmos DB 建立索引，寫入同一個 Search Index

架構：



Azure AI Search 支援的資料源：

資料源	狀態	說明
Azure Blob Storage	<input checked="" type="checkbox"/> GA	非結構化文件
Azure Cosmos DB (SQL API)	<input checked="" type="checkbox"/> GA	NoSQL 文件資料庫
Azure Cosmos DB (MongoDB API)	<input checked="" type="checkbox"/> Preview	MongoDB 相容
Azure Cosmos DB (Gremlin API)	<input checked="" type="checkbox"/> Preview	圖形資料庫
Azure SQL Database	<input checked="" type="checkbox"/> GA	關聯式資料庫
Azure SQL Managed Instance	<input checked="" type="checkbox"/> GA	托管 SQL
SQL Server on Azure VM	<input checked="" type="checkbox"/> GA	VM 上的 SQL Server
Azure MySQL	<input checked="" type="checkbox"/> Preview	MySQL 資料庫
Azure Table Storage	<input checked="" type="checkbox"/> GA	鍵值儲存
Azure Data Lake Gen2	<input checked="" type="checkbox"/> GA	大數據儲存
Azure Files	<input checked="" type="checkbox"/> Preview	檔案共享
SharePoint Online	<input checked="" type="checkbox"/> Preview	Microsoft 365
Microsoft OneLake	<input checked="" type="checkbox"/> GA	Fabric 資料湖

⚠ 不支援：Azure Cosmos DB for Cassandra、外部資料庫（PostgreSQL、Oracle）需使用 Push API

▣ Indexer 原生支援運作原理

1. 建立 Data Source 連線

定義連線字串、Table/Container、查詢條件

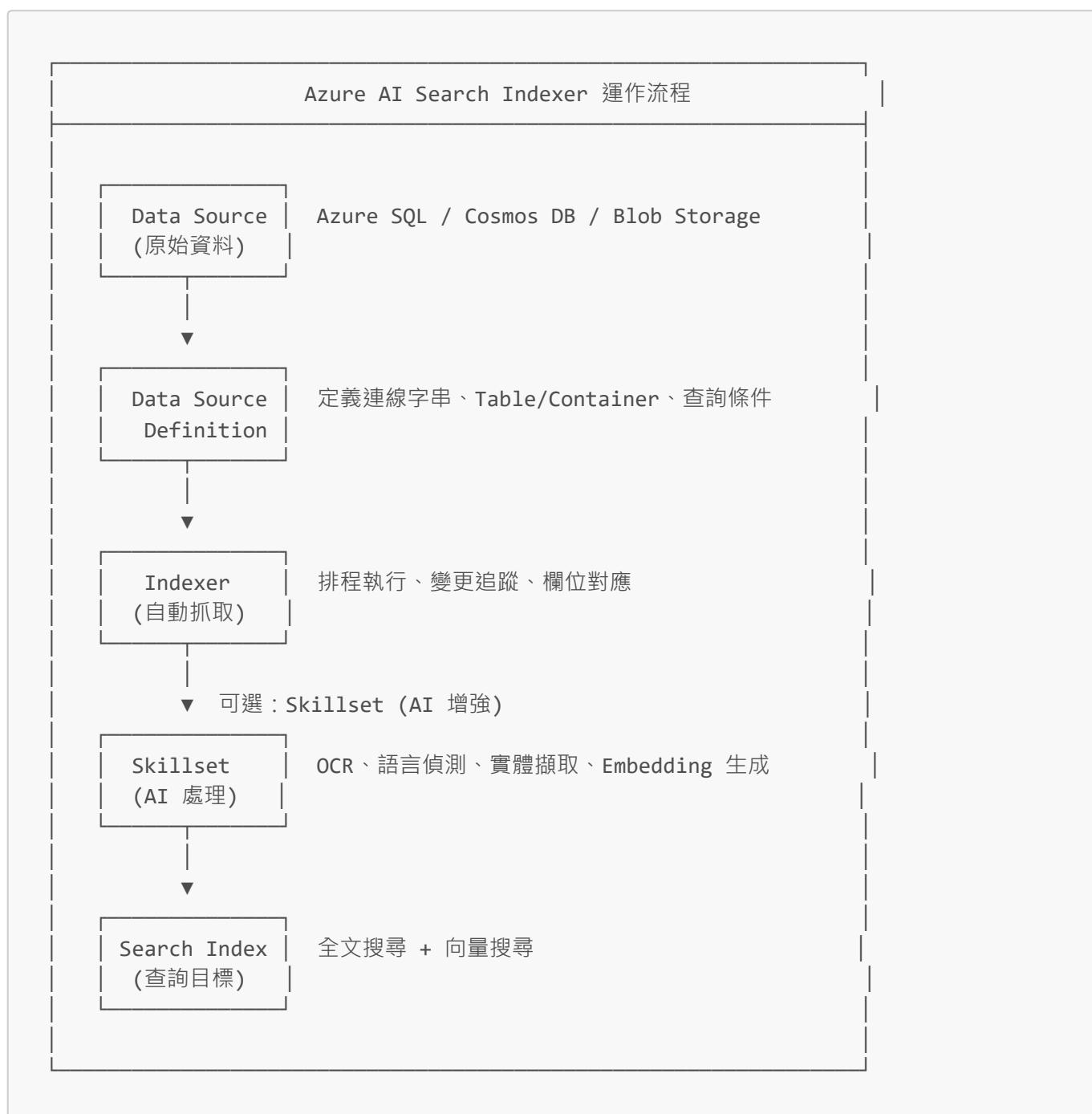
2. 建立 Indexer (自動抓取)

設定資料來源名稱、目標索引、排程間隔、欄位對應

3. 變更追蹤 (Change Tracking)

資料源	變更追蹤機制
Azure SQL	SQL Change Tracking 或 High Water Mark (timestamp 欄位)
Cosmos DB	Change Feed (內建)
Blob Storage	Last Modified timestamp

4. 完整流程圖



5. 原生支援 vs Push API 比較

方式	原生支援 (Indexer)	Push API
設定	宣告式配置	需寫程式碼
排程	Azure 自動處理	自己實作排程
變更追蹤	內建支援	自己追蹤
適用	支援的資料源	任何資料源
Portal 操作	<input checked="" type="checkbox"/> 圖形介面設定	<input type="checkbox"/> 需寫程式

💡 快速設定：Azure Portal → AI Search → **Import data** → 選擇資料源 → 設定連線 → 完成！不需要寫任何程式碼。

Pros	Cons
<input checked="" type="checkbox"/> 統一查詢介面	<input type="checkbox"/> DB 資料需轉換為文字格式
<input checked="" type="checkbox"/> Azure 原生支援	<input type="checkbox"/> 即時性受 Indexer 排程限制
<input checked="" type="checkbox"/> 向量搜尋 + 語意搜尋統一	<input type="checkbox"/> Schema 設計需考慮兩種資料源
<input checked="" type="checkbox"/> 不需改 Orchestrator	<input type="checkbox"/> Cosmos DB 變更追蹤需設定

方案 B : MCP 工具調用

作法：透過 MCP Server 將 DB 存取包裝為 Tool，讓 LLM Agent 動態調用

架構：

使用 FastMCP 建立 MCP Server，定義資料庫查詢工具：

- `query_cosmos_db`：查詢 Cosmos DB 中的結構化資料
- `get_equipment_status`：查詢特定設備的即時狀態
- `get_project_progress`：查詢專案進度摘要

Pros	Cons
<input checked="" type="checkbox"/> 即時查詢，無 Indexer 延遲	<input type="checkbox"/> LLM 需正確判斷何時調用工具
<input checked="" type="checkbox"/> 可封裝複雜業務邏輯	<input type="checkbox"/> 需維護 MCP Server
<input checked="" type="checkbox"/> 支援結構化查詢 (JOIN, 聚合)	<input type="checkbox"/> Token 消耗增加 (工具呼叫)
<input checked="" type="checkbox"/> 彈性高，可擴展	<input type="checkbox"/> 需處理權限驗證
<input checked="" type="checkbox"/> 適合 OLTP 即時資料	

方案 C : Text-to-SQL (LLM 生成查詢)

作法：提供 DB Schema，讓 LLM 生成 SQL/NoSQL 查詢語句

流程：

1. 取得 Schema
2. 請 LLM 生成查詢
3. 安全檢查 (防止 SQL Injection)
4. 執行查詢

Pros	Cons
<input checked="" type="checkbox"/> 使用者體驗自然	<input type="checkbox"/> SQL Injection 風險
<input checked="" type="checkbox"/> 支援複雜查詢	<input type="checkbox"/> LLM 生成可能有錯誤
<input checked="" type="checkbox"/> 不需預定義所有 API	<input type="checkbox"/> 需要嚴格的安全審查
	<input type="checkbox"/> Schema 變更需同步更新
	<input type="checkbox"/> 不適合敏感資料場景

方案 D : Cosmos DB 原生向量搜尋

作法：利用 Cosmos DB for NoSQL 內建的向量搜尋 + 全文搜尋 (Hybrid Search)

優點：

- 資料與向量在同一處，無需同步
- 支援 Hybrid Search (向量 + 全文)
- 即時更新，無 Indexer 延遲

限制：

- 僅適用於 Cosmos DB 內的資料
- Blob Storage 文件仍需另外處理

Pros	Cons
<input checked="" type="checkbox"/> 資料與向量統一管理	<input type="checkbox"/> 僅限 Cosmos DB 資料
<input checked="" type="checkbox"/> 即時更新	<input type="checkbox"/> Blob 文件需另外索引
<input checked="" type="checkbox"/> Hybrid Search 原生支援	<input type="checkbox"/> 需要 Cosmos DB NoSQL API
<input checked="" type="checkbox"/> 減少架構複雜度	<input type="checkbox"/> 功能較 AI Search 少

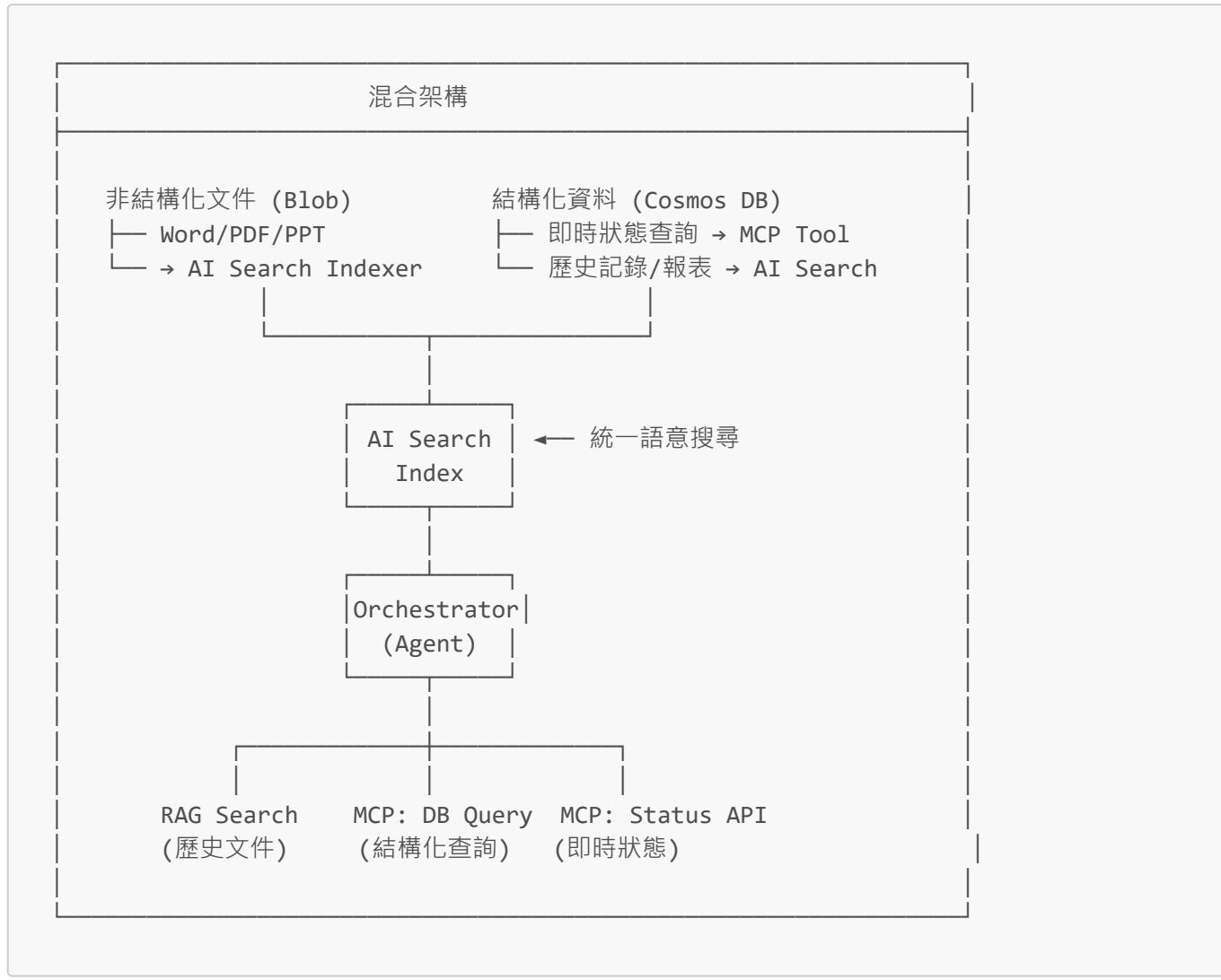
■ 方案比較總表

方案	適用場景	即時性	實作難度	維護成本
A: 雙 Indexer	主要查詢歷史/靜態資料	中 (排程)	低	低
B: MCP 工具	需即時/結構化查詢	高	中	中
C: Text-to-SQL	進階使用者，靈活查詢	高	高	高

方案	適用場景	即時性	實作難度	維護成本
D: Cosmos 原生	DB 為主要資料源	高	中	低

💡 建議組合策略

推薦：方案 A + 方案 B 混合架構



🔧 MCP Tool 設計建議

針對您的場景，建議設計以下 Tools：

1. 通用查詢工具 (結構化)

- `query_database(table, conditions)` - 依條件查詢資料庫記錄

2. 專用業務工具 (推薦)

- `get_project_status(project_id)` - 查詢專案即時狀態
- `get_floor_usage(floor_number)` - 查詢樓層用途資訊
- `get_equipment_list(location)` - 查詢特定區域的設備清單

3. 聚合查詢工具

- `get_project_summary()` - 取得所有專案進度摘要

Tool 設計原則：

- 封裝業務邏輯，降低 LLM 錯誤風險
- 限制查詢範圍，確保安全
- 回傳結構化結果，易於 LLM 理解
- 避免開放式 SQL 執行

整體方案比較總表

需求	推薦方案	實作難度	效益
RAG 正確性 vs 創造性	System Prompting + 回應格式區分	低	高
多模態 Office 圖文	轉 PDF 或 LibreOffice 自動轉換	低~中	高
檔案下架	方案 B: Soft Delete	中	高
同檔覆蓋版本保留	方案 A: Blob Versioning	低	中
版本後綴處理	方案 A: Document Family ID	中	高
跨檔案衝突	方案 A: 時間戳優先 + Prompt	低	中
多資料源整合	雙 Indexer + MCP 工具混合	中	高