RAG 資料前處理實作流程與現有工具推薦深度研究報告

1. 執行摘要與戰略背景

在當前生成式 AI（Generative AI）的技術浪潮中，檢索增強生成（Retrieval-Augmented Generation, RAG）架構已確立為企業解決大型語言模型（LLM）幻覺（Hallucination）、知識時效性滯後及領域專業知識匱乏的首選方案。然而，隨著 RAG 系統從實驗室原型走向生產環境，開發者與架構師普遍面臨一個殘酷的現實：模型的推論能力並非效能的決定性上限，資料品質（**Data Quality**）與檢索上下文的精確度（**Retrieval Precision**）才是真正的瓶頸所在。

本報告旨在提供一份詳盡、具備實操性的 RAG 資料前處理技術指南。不同於坊間僅關注「文本切割」的淺層討論，本研究將深入剖析從非結構化資料攝取（Ingestion）、高階清洗（Advanced Cleaning）、語意結構化分塊（Semantic Chunking）、元數據本體工程（Metadata Ontology Engineering）到多模態處理（Multimodal Processing）的完整技術鏈路。

根據 2024 年至 2025 年初的技術趨勢分析，資料前處理正經歷從「規則導向」向「模型導向」的典範轉移。傳統的正則表達式（Regex）與光學字元辨識（OCR）已不足以應對複雜的企業文檔，新一代的 **GenAI-Native** 解析器（如 LlamaParse）與多向量檢索架構（Multi-Vector Retrieval）正在重塑資料處理的標準。本報告將透過量化基準測試（Benchmark），深度比較 LlamaParse、Unstructured.io、Docling 與 LangChain 等工具在處理 PDF 表格、多欄排版及混合媒體內容時的效能差異，並針對不同規模的企業場景提出具體的架構建議。

---

2. RAG 資料前處理的理論框架與核心挑戰

2.1「垃圾進，垃圾出」在向量空間的幾何效應

在傳統機器學習中，「垃圾進，垃圾出」（Garbage In, Garbage Out）通常指訓練數據的雜訊會導致模型權重偏差。而在 RAG 架構中，這一效應主要體現在向量空間的語意稀釋（**Semantic Dilution**）。

當文檔包含無意義的頁眉、頁尾、亂碼或斷裂的句子時，嵌入模型（Embedding Model）會將這些雜訊一併編碼進向量中 [1]。這會導致向量在高維空間中的位置發生偏移，使得原本語意相關的查詢（Query）無法與文檔向量產生足夠高的餘弦相似度（Cosine Similarity），進而導致檢索失敗（Recall Failure）。更嚴重的是，若分塊策略切斷了邏輯關聯（例如將問題與答案切分到不同塊），LLM 將因缺乏完整上下文而產生錯誤推論 [3]。

2.2 從非結構化到語意結構化（**Semantic Structuring**）

現代 RAG 前處理的核心目標，是將非結構化的二進制文件（如 PDF, DOCX）轉換為具備語意結構

的中介格式。這包含三個層次的還原：

1. 物理佈局還原（**Layout Restoration**）：正確識別多欄排版、閱讀順序、邊欄註釋與主文的關係。
2. 邏輯結構提取（**Logical Extraction**）：識別標題層級（H1-H3）、列表關係、表格的行列對應。
3. 語意關聯增強（**Semantic Enrichment**）：提取隱含的元數據（如發布年份、作者意圖、實體關係），並將其顯性化為可檢索的標籤 [5]。

2.3 標準化前處理流水線（Preprocessing Pipeline）

一個生產級的 RAG 資料處理流水線應包含以下五個嚴格定義的階段，各階段間應具備可觀測性與模組化設計：

1. 資料攝取與解析（**Ingestion & Parsing**）：處理多源異構資料，解決格式相容性與OCR難題。
2. 清洗與正規化（**Cleaning & Normalization**）：去除雜訊，標準化編碼與格式。
3. 結構化分塊（**Structural & Semantic Chunking**）：將長文檔切分為適合嵌入的語意單元。
4. 元數據提取（**Metadata Extraction**）：利用 NLP 或 LLM 提取關鍵資訊以支援混合檢索。
5. 嵌入與索引策略（**Embedding & Indexing Strategy**）：選擇合適的嵌入模型與索引結構（如父子索引）。

---

3. 第一階段：資料攝取與解析技術深度評測

此階段是 RAG 系統的第一道關卡，也是技術門檻最高的環節。PDF 作為企業最常見的文檔格式，其「視覺導向」的特性使其成為資料解析的重災區。

3.1 PDF 解析的本質難題與技術分類

PDF（Portable Document Format）本質上是一系列繪圖指令的集合，而非結構化資料的容器。它記錄的是「在座標 (x, y) 繪製字符 'A'」，而非「這裡是標題」。因此，解析器必須進行逆向工程以重建文本結構。目前的解析技術主要分為三類：

1. 基於規則與流的解析（**Rule/Stream-based**）：如 PyMuPDF, pdfplumber。直接讀取 PDF 內容流，速度極快，但無法處理掃描件（圖片型 PDF）且對複雜排版（如跨頁表格）識別率低。
2. 基於計算機視覺的解析（**Vision-based / Layout Analysis**）：如 Unstructured (依賴 Detectron2 或 YOLOX), Microsoft Azure AI Document Intelligence。利用視覺模型識別版面區塊（表格、圖片、標題），再對區塊進行 OCR。準確率較高，但速度慢且計算成本高。
3. 基於生成式 **AI** 的解析（**GenAI-Native**）：如 LlamaParse。利用多模態大模型（Multimodal LLMs）直接「看」文件並生成結構化輸出（如 Markdown）。這是目前的 SOTA（State-of-the-Art）方案，特別擅長處理表格與圖表 [7]。

3.2 主流工具深度評測：LlamaParse vs. Unstructured vs. Docling

根據 2025 年的最新基準測試數據，以下是三款核心工具的詳細對比分析：

3.2.1 LlamaParse (LlamaIndex 生態)

LlamaParse 是 LlamaIndex 團隊推出的雲端解析服務，專為 LLM 應用設計。

- 核心優勢：
  - 表格重建能力：這是 LlamaParse 的殺手級功能。它能極高精度地將 PDF 中的複雜表格轉換為 Markdown 格式（| Header | Value |），保留了行與列的語意對應，這是傳統 OCR 工具難以企及的 [8]。
  - 多模態理解：能理解並描述圖片內容（Image Captioning），將圖表轉化為文字描述整合進文檔流。
  - 速度與擴展性：基準測試顯示，LlamaParse 的處理速度極快，平均每份文檔約 6 秒，且受頁數增加的影響極小，展現了強大的後端並行處理能力 [10]。
  - 整合性：與 LlamaIndex 框架無縫整合，支援遞歸檢索（Recursive Retrieval）所需的節點物件生成。
- 劣勢與限制：
  - 成本結構：作為 SaaS 服務，其定價模式為每頁計費（例如免費用量後每頁 0.3 點數，複雜模式更貴）。對於擁有百萬級文檔庫的企業，成本可能成為考量 [11]。
  - 資料隱私：需將文件上傳至 LlamaCloud，對於受限於 GDPR 或金融合規（On-premise only）的場景可能不適用。

3.2.2 Unstructured.io (開源 ETL 霸主)

Unstructured 是一個功能全面的開源 ETL 庫，提供從 Partitioning 到 Cleaning 的一站式服務。

- 核心優勢：
  - 部署靈活性：提供 Docker 映像檔與 Python 庫，支援完全本地化部署（Air-gapped環境），確保資料不出內網，這是金融與國防領域的首選 [12]。
  - 格式廣泛性：支援超過 25 種文件格式（PPT, HTML, EML, EPUB 等），是目前支援度最廣的開源工具。
  - 細粒度元素控制：解析結果將文檔拆解為 Title, NarrativeText, ListItem, Table 等具體元素類別，便於後續針對不同類型元素應用不同的清洗或嵌入策略 [12]。
- 劣勢與限制：
  - 效能瓶頸：在開啟高精度 OCR（hi_res 策略）時，處理速度顯著慢於 LlamaParse。測試顯示單頁處理可能需 50 秒以上，且隨頁數增加呈線性增長，大規模處理需自行搭建龐大的 GPU 集群 [10]。
  - 表格精度：雖然整合了 Tesseract 等工具，但在處理無框線表格或複雜合併單元格時，準確率（約 75%）低於 LlamaParse [10]。

3.2.3 Docling (IBM Research)

Docling 是 IBM 研究院開源的新秀，專注於文檔佈局分析。

- 核心優勢：
  - 極致的表格精度：在 RD-TableBench 等基準測試中，Docling 在表格單元格識別準確率上達到 97.9%，超越了許多商業方案 [10]。
  - 結構保留：非常擅長識別文檔的閱讀順序與邏輯層級。

- 劣勢：
  - 速度：處理速度隨頁數線性增長（例如 50 頁需 65 秒），在超長文檔處理上不如 LlamaParse 的並行架構高效 [10]。
  - 生態整合：相較於 Unstructured，其與 LangChain/LlamaIndex 的整合生態尚在發展中。

3.3 工具選擇決策矩陣

評估維度	**LlamaParse**	**Unstructured.io (Open Source)**	**Docling**	**PyMuPDF / LangChain**
最佳適用場景	複雜報表、含大量表格與圖表的 PDF、需快速上線的 RAG	銀行/醫療等需私有化部署場景、多種異構格式混合	科研論文、需極高精度表格分析的場景	簡單純文字合約、即時性要求極高且格式簡單的場景
表格處理能力	⭐⭐⭐⭐⭐ (Markdown 重建)	⭐⭐⭐ (OCR 辨識)	⭐⭐⭐⭐⭐ (結構化物件)	⭐ (僅文字提取)
處理速度	極快 (~6s/doc)	慢 (50s+/page @ hi_res)	中等 (線性增長)	極快 (毫秒級)
隱私合規	需上傳雲端 (SaaS)	可完全本地部署 (Local/Docker)	可本地部署	本地執行
成本模式	付費 API (有免費額度)	免費 (自付運算資源)	免費 (開源)	免費

---

4. 第二階段：資料清洗與標準化工程

原始解析出的文本往往充滿雜訊。資料清洗的目標是提高「信噪比」（Signal-to-Noise Ratio），確保嵌入向量能精確反映內容語意。

4.1 文本正規化 (Text Normalization)

文本正規化是基礎但關鍵的步驟，主要解決編碼與格式問題。

- **Unicode 修復 (Mojibake Fixing)**：PDF 提取常導致編碼錯誤（如 schÃ¶n 應為 schön）。
  - 工具推薦：**ftfy (Fix Text For You)**。這是 Python 社群的標準庫，能自動偵測並修復損壞

的 Unicode 字符串 [15]。
    - 實作代碼：ftfy.fix_text(raw_text)。
- 空白與格式標準化：移除多餘的換行符、製表符與連續空白，這些通常是 PDF 排版留下的痕跡。
    - 工具推薦：**clean-text** 庫或 Unstructured 的 clean 函數 [14]。
    - 功能：clean(text, extra_whitespace=True, bullets=True) 可移除項目符號與多餘空白。

4.2 結構性雜訊去除 (Structural Noise Removal)

結構性雜訊是指那些對人類閱讀有幫助，但對語意檢索有害的元素。

- 頁眉與頁尾 **(Headers & Footers)**：若不移除，檢索「聯絡方式」時可能會匹配到每一頁頁腳的公司地址，導致檢索結果被單一文檔的不同頁面佔滿（Diversity Collapse）。
    - 實作策略：
        1. 基於元數據：若使用 Unstructured，可直接過濾 Category == 'Header' 的元素。
        2. 基於正則表達式 **(Regex)**：針對常見模式（如 "Page \d+ of \d+"）編寫 Regex 規則進行移除。
        3. 基於邊界框 **(Bounding Box)**：在解析階段設定 bbox 參數，忽略頁面頂部與底部 10% 的區域 [18]。
- 引用文獻與註釋：學術論文中的 `` 或 (Smith, 2020) 可能干擾語意連貫性。可使用 unstructured.cleaners.core.replace_citation 或自定義 Regex \[\d+\] 去除 [14]。

4.3 隱私敏感資訊處理 (PII Redaction)

在將資料送入 Embedding 模型（尤其是公有雲模型）前，必須處理個人識別資訊（PII）。

- 實作策略：
    - 使用 **Microsoft Presidio** 或 **Hugging Face** 的 **PII** 模型 掃描文本。
    - 將識別出的 Email、電話、信用卡號替換為佔位符（如 <EMAIL>, <PHONE>），既保留了實體類型的語意，又保護了隱私 [19]。

---

5. 第三階段：進階分塊策略與架構設計

分塊（Chunking）是 RAG 系統中影響檢索效能最深遠的變數。Chunk 過大包含雜訊，Chunk 過小則語意破碎。

5.1 固定大小與重疊分塊 (Fixed-size with Overlap)

這是最基礎的策略，使用固定字符數（如 500）切分，並保留重疊（如 50）。

- 工具：LangChain RecursiveCharacterTextSplitter。
- 最佳實踐：雖然簡單，但建議使用**遞歸（Recursive）**策略，優先在段落（\n\n）、句子（\n、.）邊界切分，而非硬性切斷單詞 [20]。
- 適用性：適合結構鬆散、缺乏明確章節的文檔。

5.2 語意分塊 (Semantic Chunking)

這是一種由數據驅動的進階策略，旨在讓每個 Chunk 包含一個完整的「話題」(Topic)。

- 演算法邏輯：
  1. 將文檔按句子拆分。
  2. 計算相鄰句子的嵌入向量餘弦相似度。
  3. 設定一個閾值(Threshold，如 95 百分位數)。當相鄰句子的相似度低於此閾值時，判定為話題轉換，在此處建立切分點 [23]。
- 優勢：大幅提升檢索的語意純度，避免一個 Chunk 包含兩個不相關的主題。
- 實作代碼概念 **(LangChain)**：

```Python
from langchain_experimental.text_splitter import SemanticChunker
from langchain_openai.embeddings import OpenAIEmbeddings

text_splitter = SemanticChunker(
    OpenAIEmbeddings(),
    breakpoint_threshold_type="percentile" # 或 "standard_deviation"
)
docs = text_splitter.create_documents([text])
```

- 代價：計算成本較高，需對全文進行 Embedding 計算。

5.3 父文檔檢索策略 (Parent Document Retriever / Small-to-Big)

此策略解決了「檢索粒度」與「生成粒度」的矛盾。檢索需要精細的特徵(小塊)，生成需要完整的上下文(大塊)。

- 架構機制：
  1. 大塊(**Parent Chunk**)：將文檔切分為較大的塊(如 2000 tokens)，存入 DocStore。
  2. 小塊(**Child Chunk**)：將每個大塊進一步切分為多個小塊(如 200-400 tokens)。
  3. 索引與檢索：對小塊進行 Embedding 並存入向量庫。
  4. 生成：當檢索命中某個小塊時，透過 ID 映射回傳其對應的**父文檔(大塊)**給 LLM [25]。
- 優勢：顯著提升了 RAG 回答的完整性與連貫性，特別適合需要綜合上下文的複雜問答。

5.4 代理式分塊 (Agentic Chunking)

利用 LLM 本身來判斷分塊邊界。雖然成本最高，但對於法律合約等對邏輯完整性要求極高的文檔，效果最佳。LLM 會閱讀文本並決定：「這一條款應該獨立成塊」[26]。

---

6. 第四階段：多模態與複雜數據處理(表格與圖片)

傳統 RAG 常忽略表格與圖片，導致「資訊黑洞」。現代 RAG 必須具備多模態處理能力。

6.1 表格處理：Markdown 序列化與摘要索引

表格數據若被展平為純文字，其二維結構將丟失，導致 "Row 1 Col 1" 與 "Row 2 Col 1" 的關聯斷裂。

- 策略一：**Markdown 序列化 (Markdown Serialization)**
  - 機制：使用 LlamaParse 將表格轉為 Markdown 格式（| Product | Price |）。
  - 優勢：大多數現代 LLM（GPT-4, Claude 3.5）在預訓練階段已見過大量 Markdown 表格，能完美理解其結構並進行推理 [9]。
- 策略二：表格摘要 **(Table Summarization)**
  - 機制：
    1. 提取表格內容。
    2. 使用 LLM 生成一段自然語言摘要（例如：「此表顯示該公司 2023 Q3 營收增長 15%，主要來自雲端業務...」）。
    3. 僅對摘要進行 **Embedding**。
    4. 檢索命中摘要後，將**原始表格數據（Markdown 或 HTML）**注入 Prompt。
  - 理由：避免表格中的大量數字干擾向量檢索的語意匹配 [28]。

6.2 圖片處理：多模態 RAG (Multimodal RAG)

- 圖片摘要 **(Image Captioning)**：
  - 使用 VLM（如 GPT-4o, LLaVA）為圖片生成詳細描述。
  - 將描述文字向量化並存入索引。
  - 檢索時匹配描述文字，回傳圖片引用或內容。
- 多向量檢索器 **(Multi-Vector Retriever)**：
  - 這是一種通用的架構模式（LangChain 支援），允許一個文檔有多個向量表示（如：原始文本向量 + 摘要向量 + 假設性問題向量），任一向量命中均回傳同一份文檔內容 [30]。

---

7. 第五階段：元數據本體工程與混合檢索

元數據（Metadata）是 RAG 系統的導航圖。高品質的元數據能實現混合檢索（**Hybrid Search**），即「關鍵字過濾 + 向量搜索」。

7.1 元數據本體設計 (Metadata Ontology)

應根據業務需求設計元數據架構：

- 來源元數據：filename, page_number, source_url, author.
- 時間元數據：date_created, fiscal_year.
- 語意元數據：keywords, summary, category, sentiment.

7.2 自動化元數據提取 (Automated Extraction)

不應依賴人工標註，而應在攝取階段自動生成元數據。

- 關鍵詞與實體提取：使用 NLP 工具（如 Spacy, TextRank）或 LLM 提取文檔中的關鍵實體（公司名、人名、產品名）存入 keywords 欄位 [32]。
- 文檔摘要：對每個 Chunk 生成一句話摘要，存入元數據。
- **HyDE (Hypothetical Document Embeddings)**：這是一種強大的增強技術。
  - 機制：讓 LLM 針對該 Chunk 生成 3-5 個「用戶可能會問的問題」。
  - 應用：將這些問題拼接成字符串存入元數據或獨立向量化。這能顯著提升問題與答案之間的語意匹配度 [5]。

7.3 自查詢檢索器 (Self-Querying Retriever)

利用提取的元數據，我們可以實作 LangChain 的 SelfQueryRetriever。

- 工作流：
  1. 用戶輸入：「我想找 2023 年關於 AI 安全的報告」。
  2. LLM 分析查詢，拆解出：
     - **Query**: "AI 安全"
     - **Filter**: year == 2023 and category == 'report'
  3. 系統對向量資料庫執行帶過濾條件的搜索，大幅提升精確度 [5]。

---

8. 第六階段：品質保證與自動化評估 (Ragas Framework)

如何量化前處理的效果？不能憑感覺。**Ragas (Retrieval Augmented Generation Assessment)** 是目前業界標準的評估框架。

8.1 核心評估指標 (Key Metrics)

Ragas 提供了幾個與前處理直接相關的指標：

1. 上下文召回率 **(Context Recall)**：
   - 定義：檢索到的上下文是否包含了回答問題所需的所有事實？
   - 前處理啟示：若此分數低，通常意味著分塊策略有誤（Chunk 太小切斷了資訊）或解析失敗（表格資訊遺失）[36]。
2. 上下文精確度 **(Context Precision)**：
   - 定義：在檢索到的多個 Chunk 中，相關內容是否排名靠前？
   - 前處理啟示：若此分數低，意味著雜訊太多（頁眉頁尾未清洗）或嵌入模型被不相關資訊干擾。
3. 忠實度 **(Faithfulness)**：
   - 定義：生成的答案是否完全基於檢索到的上下文？
   - 前處理啟示：雖主要受 LLM 影響，但若上下文結構混亂（如崩壞的表格），LLM 更容易產生幻覺。

8.2 評估驅動開發 (Evaluation-Driven Development)

建議建立一個「黃金數據集」(Golden Dataset)，包含數十個 對。在調整任何前處理參數(如 chunk_size, parsing_method)後，自動運行 Ragas 評估，觀察指標變化，以數據驅動優化決策 [38] 。

---

9. 總結與架構建議

9.1 綜合工具推薦表

功能模組	開源/免費方案	企業/付費方案	推薦理由
PDF 解析	Unstructured (Local), PyMuPDF	**LlamaParse**, Adobe API	對於複雜表格與非結構化 PDF, LlamaParse 是目前的最佳選擇。
資料清洗	**ftfy**, **clean-text**	無	結合 Regex 與 NLP 庫進行定製化清洗是標準做法。
分塊策略	RecursiveCharacter TextSplitter	SemanticChunking (OpenAI/Cohere)	始於 Recursive，若資源允許則升級至 Semantic 或 Agentic Chunking。
向量資料庫	Chroma, PGVector, FAISS	Pinecone, Weaviate Cloud	PGVector 適合已有 PostgreSQL 架構的團隊；Pinecone 適合無伺服器架構。
評估框架	**Ragas**, TruLens	Arize Phoenix	Ragas 的指標定義最為嚴謹且易於整合。

9.2 未來展望：Agentic Ingestion

隨著 Agent 技術的發展，未來的資料前處理將從「靜態流水線」轉向「動態代理」。**Agentic Ingestion** 指的是由 AI Agent 自主瀏覽文檔，判斷哪些部分是表格、哪些是雜訊，並動態決定分塊策略與元數據標籤。這將進一步降低 RAG 系統對硬編碼規則的依賴，實現真正的全自動化知

識攝取。

對於當下的實踐者而言，建議從**「LlamaParse 解析 + Markdown 處理 + 父文檔檢索 + Ragas 評估」**這一黃金組合入手，這是在成本、效能與開發複雜度之間取得最佳平衡的架構路徑。

1

引用的著作

1. Build an unstructured data pipeline for RAG | Databricks on AWS, 檢索日期：1月 27, 2026, https://docs.databricks.com/aws/en/generative-ai/tutorials/ai-cookbook/quality-data-pipeline-rag
2. Practical tips for retrieval-augmented generation (RAG) - The Stack Overflow Blog, 檢索日期：1月 27, 2026, https://stackoverflow.blog/2024/08/15/practical-tips-for-retrieval-augmented-generation-rag/
3. RAG Chunking Strategies: Practical Guide for Retrieval-Augmented Generation (in Java) | by Vishal Mysore | Dec, 2025, 檢索日期：1月 27, 2026, https://medium.com/@visrow/rag-chunking-strategies-practical-guide-for-retrieval-augmented-generation-in-java-0e73dce33623
4. Choosing the Right Chunking Strategy: What Nobody Tells You, 檢索日期：1月 27, 2026, https://medium.com/@manojkotary/choosing-the-right-chunking-strategy-what-nobody-tells-you-8829e2cb99f8
5. Metadata for RAG: Improve Contextual Retrieval | Unstructured, 檢索日期：1月 27, 2026, https://unstructured.io/insights/how-to-use-metadata-in-rag-for-better-contextual-results
6. Leveraging Metadata in RAG Customization | deepset Blog, 檢索日期：1月 27, 2026，https://www.deepset.ai/blog/leveraging-metadata-in-rag-customization
7. Fix RAG Hallucinations at the Source: Top PDF Parsers Ranked 2025 | by Jiten Bhalavat, 檢索日期：1月 27, 2026, https://infinityai.medium.com/3-proven-techniques-to-accurately-parse-your-pdfs-2c01c5badb84
8. Parsing PDFs with LlamaParse: a how-to guide - LlamaIndex, 檢索日期：1月 27, 2026，https://www.llamaindex.ai/blog/pdf-parsing-llamaparse
9. How to Prepare PDFs for RAG Pipelines (with Examples), 檢索日期：1月 27, 2026, https://medium.com/@cgtyklnc/how-to-prepare-pdfs-for-rag-pipelines-with-examples-570ea5efb1fc
10. PDF Data Extraction Benchmark 2025: Comparing Docling, Unstructured, and LlamaParse for Document Processing Pipelines - Procycons, 檢索日期：1月 27, 2026，https://procycons.com/en/blogs/pdf-data-extraction-benchmark/
11. LlamaIndex Pricing Guide: Everything You Must Know Before Investing - ZenML Blog, 檢索日期：1月 27, 2026，https://www.zenml.io/blog/llamaindex-pricing

12. Overview - Unstructured, 検索日期：1月 27, 2026, https://docs.unstructured.io/open-source/introduction/overview
13. Open-Source Unstructured Data ETL with Unstract, Ollama, DeepSeek, and PostgreSQL, 検索日期：1月 27, 2026, https://unstract.com/blog/open-source-document-data-extraction-with-unstract-deepseek/
14. Cleaning - Unstructured document, 検索日期：1月 27, 2026, https://docs.unstructured.io/open-source/core-functionality/cleaning
15. fixes text for you — ftfy 4.1 documentation - Read the Docs, 検索日期：1月 27, 2026, https://ftfy.readthedocs.io/en/v4.2.0/
16. rspeer/python-ftfy: Fixes mojibake and other glitches in Unicode text, after the fact. - GitHub, 検索日期：1月 27, 2026, https://github.com/rspeer/python-ftfy
17. clean-text - PyPI, 検索日期：1月 27, 2026, https://pypi.org/project/clean-text/
18. Core Functionality - Unstructured 0.12.6 documentation - Read the Docs, 検索日期：1月 27, 2026, https://unstructured.readthedocs.io/en/main/core.html
19. Mastering Data Cleaning for Fine-Tuning LLMs and RAG Architectures | AI Alliance, 検索日期：1月 27, 2026, https://thealliance.ai/blog/mastering-data-cleaning-for-fine-tuning-llms-and-r
20. Mastering Chunking Strategies for RAG: Best Practices & Code Examples - Databricks Community, 検索日期：1月 27, 2026, https://community.databricks.com/t5/technical-blog/the-ultimate-guide-to-chunking-strategies-for-rag-applications/ba-p/113089
21. Chunking Strategies for LLM Applications - Pinecone, 検索日期：1月 27, 2026, https://www.pinecone.io/learn/chunking-strategies/
22. Chunking Techniques with Langchain and LlamaIndex - LanceDB, 検索日期：1月 27, 2026, https://lancedb.com/blog/chunking-techniques-with-langchain-and-llamaindex/
23. Semantic Chunking Definitive Guide: Free Python Code Included | by Blue sky | Medium, 検索日期：1月 27, 2026, https://medium.com/@hasanaboulhassan_83441/semantic-chunking-definitive-guide-free-python-code-included-a06044ab0543
24. A guide to understand Semantic Splitting for document chunking in LLM applications : r/LangChain - Reddit, 検索日期：1月 27, 2026, https://www.reddit.com/r/LangChain/comments/1erxo60/a_guide_to_understand_semantic_splitting_for/
25. Multi-Vector Retriever for RAG on tables, text, and images - LangChain Blog, 検索日期：1月 27, 2026, https://www.blog.langchain.com/semi-structured-multi-modal-rag/
26. Implement RAG chunking strategies with LangChain and watsonx.ai - IBM, 検索日期：1月 27, 2026, https://www.ibm.com/think/tutorials/chunking-strategies-for-rag-with-langchain-watsonx-ai
27. Mastering RAG: Precision techniques for table-heavy documents - KX, 検索日期：1月 27, 2026, https://kx.com/blog/mastering-rag-precision-techniques-for-table-heavy-docum

ents/
28. LangChain-OpenTutorial/10-Retriever/07-MultiVectorRetriever ..., 検索日期：1月 27, 2026, https://github.com/LangChain-OpenTutorial/LangChain-OpenTutorial/blob/main/10-Retriever/07-MultiVectorRetriever.ipynb
29. LLM Study Diary: Decoding LangChain's Official Multimodal RAG Sample - Medium, 検索日期：1月 27, 2026, https://medium.com/llm-study-diary-a-beginners-path-through-ai/llm-study-diary-decoding-langchains-official-multimodal-rag-sample-b4a645bdabe2
30. RAG: Multi Vector Retriever - Kaggle, 検索日期：1月 27, 2026, https://www.kaggle.com/code/marcinrutecki/rag-multi-vector-retriever
31. LangChain: MultiVectorRetriever Quick Reference - Kaggle, 検索日期：1月 27, 2026, https://www.kaggle.com/code/ksmooi/langchain-multivectorretriever-quick-reference
32. Advanced RAG techniques part 1: Data processing - Elasticsearch ..., 検索日期：1月 27, 2026, https://www.elastic.co/search-labs/blog/advanced-rag-techniques-part-1
33. Advanced RAG Techniques for High-Performance LLM Applications - Graph Database & Analytics - Neo4j, 検索日期：1月 27, 2026, https://neo4j.com/blog/genai/advanced-rag-techniques/
34. Self-querying retrievers with Elasticsearch: Unleashing your metadata, 検索日期：1月 27, 2026, https://www.elastic.co/search-labs/blog/self-querying-retrievers
35. How to Build a RAG System with a Self-Querying Retriever in LangChain - Medium, 検索日期：1月 27, 2026, https://medium.com/data-science/how-to-build-a-rag-system-with-a-self-querying-retriever-in-langchain-16b4fa23e9ad
36. RAG evaluation metrics: How to evaluate your RAG pipeline with Braintrust - Articles, 検索日期：1月 27, 2026, https://www.braintrust.dev/articles/rag-evaluation-metrics
37. RAG Evaluation Metrics Explained: A Complete Guide with Examples | by Sanjeeb Panda | Jan, 2026, 検索日期：1月 27, 2026, https://medium.com/@sanjeebmeister/rag-evaluation-metrics-explained-a-complete-guide-with-examples-dea8bf4467db
38. Evaluate RAG pipeline using Ragas in Python with watsonx - IBM, 検索日期：1月 27, 2026, https://www.ibm.com/think/tutorials/evaluate-rag-pipeline-using-ragas-in-python-with-watsonx
39. Evaluate a simple RAG system - Ragas, 検索日期：1月 27, 2026, https://docs.ragas.io/en/latest/tutorials/rag/
40. Effective Practices for Architecting a RAG Pipeline - InfoQ, 検索日期：1月 27, 2026, https://www.infoq.com/articles/architecting-rag-pipeline/
41. Advanced RAG: Techniques & Concepts | Data Science Collective - Medium, 検索日期：1月 27, 2026, https://medium.com/data-science-collective/advanced-rag-techniques-concepts

-e0b67366c5cf
42. What Is Chunking in RAG (Retrieval-Augmented Generation) and Why It Matters for AI, 検索日期：1月 27, 2026, https://medium.com/@sangitapokhrel911/chunking-in-rag-the-secret-sauce-behind-smarter-ai-responses-71c99ef70f9a
43. Parsing Word, CSV, Excel, JSON, and SQL Data for Retrieval-Augmented Generation (RAG), 検索日期：1月 27, 2026, https://medium.com/@sangitapokhrel911/parsing-word-csv-excel-json-and-sql-data-for-retrieval-augmented-generation-rag-a0798b8d5405
44. High-Precision RAG for Table Heavy Documents... (Using LangChain, Unstructured.io, & KDB.AI) | by Ryan Siegler | KX Systems | Medium, 検索日期：1月 27, 2026, https://medium.com/kx-systems/high-precision-rag-for-table-heavy-documents-using-langchain-unstructured-io-kdb-ai-22f7830eac9a
45. A Beginner's Guide to Evaluating RAG Pipelines Using RAGAS - Analytics Vidhya, 検索日期：1月 27, 2026, https://www.analyticsvidhya.com/blog/2024/05/a-beginners-guide-to-evaluating-rag-pipelines-using-ragas/