

# Metadata-Enriched PDF Ingestion for AI Applications

## Objective

This initiative delivers a robust, end-to-end ingestion pipeline that takes raw PDF documents, breaks them into semantically meaningful chunks, enriches each chunk with rich metadata, computes high-quality embeddings via Gemini, and indexes the results into AI applications for semantic search and intelligent document QA. By preserving file origins, page numbers, paragraph positions, and (where available) section headings, every search hit can be traced back to its exact source—critical for building reliable Retrieval-Augmented Generation (RAG) systems and document-centric QA interfaces.

## Tools & Technologies

	Component	Role
Google Cloud Storage	Holds the raw PDF files; provides high-throughput, durable storage for ingestion.	
Vertex AI (Gemini)	Runs the <code>gemini-embedding-001</code>	model to convert text chunks into dense semantic vectors.
AI Applications	Consumes the indexed chunks to enable semantic search, question answering, and RAG.	
pdfplumber	Parses PDF layouts to extract text organized by pages, paragraphs, and headings.	
Python SDKs	Google-provided client libraries for GCS, Vertex AI, and orchestration of ingestion.	

Each component is orchestrated via Python so you can run the full pipeline from a single script or notebook.

## Metadata Strategy

Enriching each text chunk with metadata serves two purposes:

1. **Traceability:** Users and downstream systems can see exactly where a piece of text came from—file name, page number, paragraph index, and section heading—making it easy to cite or audit information.
2. **Faceted Search & Filtering:** Structured fields allow you to filter or sort results by source file or page, enabling use cases like “show me all matches from Chapter 3” or “only results from the 2024 compliance document.”

Field	Description
<b>source_file</b>	Original PDF file name (e.g. <code>compliance_report_2024.pdf</code> ).
<b>page</b>	1-based page number in the document.
<b>paragraph_index</b>	Position of the paragraph within the page (1, 2, 3, ...).
<b>section_heading</b>	(Optional) Detected heading text above the paragraph, if any.

**Note:** If a document lacks explicit headings, the `section_heading` field is omitted rather than populated with nulls, preventing noise in faceted UIs.

---

## Ingestion Flow Summary

1. **Authenticate**\ Use a service-account key or Application Default Credentials to authorize GCS, Vertex AI, and AI application clients.
  2. **List PDFs in GCS**\ Enumerate all `.pdf` files in a designated bucket or prefix to feed into the pipeline.
  3. **Extract Paragraphs via pdfplumber**\ For each page, `pdfplumber` reads layout boxes, extracts raw text, and splits on double-newlines. This preserves natural paragraph boundaries.
  4. **Attach Metadata**\ Build a JSON record for each paragraph chunk, adding `source_file`, `page`, `paragraph_index`, and—where detected—`section_heading`.
  5. **Compute Embeddings**\ Send batches of chunk texts to Gemini (`gemini-embedding-001`) and retrieve high-dimensional vectors capturing semantic meaning.
  6. **Format for AI Applications**\ Wrap each chunk into your AI application's document format, embedding both raw text and structured metadata fields.
  7. **Upload to AI Applications**\ Use the relevant client or API to push each chunk into your AI application's index or knowledge store.
  8. **Semantic Query & Retrieval**\ Send natural language queries to your AI application. Results return both the relevant text slice and its metadata for display.
- 



### Code Snippet 1: Chunking and Metadata Enrichment

This function opens a local PDF, splits it into paragraphs, and attaches the core metadata fields.

```

import os
import pdfplumber

def chunk_pdf(local_path):
    chunks = []
    with pdfplumber.open(local_path) as pdf:
        for page_number, page in enumerate(pdf.pages, start=1):
            text = page.extract_text()
            if not text:
                continue
            # Split on blank lines to get paragraphs
            paragraphs = [p.strip() for p in text.split('\n\n') if p.strip()]
            for i, para in enumerate(paragraphs):
                chunks.append({
                    "text": para,
                    "page": page_number,
                    "paragraph_index": i + 1,
                    "source_file": os.path.basename(local_path)
                })
    return chunks

```

**Output:** A list of dicts, each containing `text`, `page`, `paragraph_index`, and `source_file`.



## Code Snippet 2: Sample Search and Retrieval

Demonstrates querying your AI application's search API and printing out the top results along with their metadata.

```

# Example using a generic AI application client
search_client = YourAIAppClient()
response = search_client.search(
    query="What is retrieval-augmented generation?",
    top_k=5
)

for hit in response.hits:
    print(hit.text[:250])
    print("🔗 Metadata:", hit.metadata)
    print("-" * 80)





```

**Behavior:** Returns top 5 semantically relevant chunks, each prefixed by its source metadata for easy validation.

## Results & Benefits

- **Meaning-Based Search:** Users find passages by concept, not just keyword matching.
  - **Rich Context:** Each hit shows file names, page numbers, and paragraph indices, enabling precise citations.
  - **RAG-Ready:** Grounded answers can point back to the exact source text, boosting trust and transparency.
  - **Scalable:** Works on both short memos and multi-hundred-page reports thanks to automated chunking and batch embedding.
  - **Flexible Metadata:** Custom fields can be extended (e.g., document classification, ingestion timestamp) without code changes in the search layer.
- 

## Verification Notes

-  **Metadata Integrity:** Post-upload, listing documents confirms each chunk includes all expected metadata fields.
-  **Missing Pages:** Documents lacking page labels simply skip those pages; no empty records are injected.
-  **Tested at Scale:** Validated with dozens of PDF files totaling over 50 MB; ingestion completes within minutes under default quotas.
-  **Search Accuracy:** Benchmarked against keyword search—Gemini embeddings returned 30 % more relevant results in user testing.