PDF parsing with Tasks and Images

PDF Parsing → Text Extraction

The first and most crucial stage in document processing is extracting text from a PDF. PDFs are available in a variety of forms; some are just text-based documents, while others have intricate layouts, tables, or scanned images. From all these variations, a good parser must consistently retrieve useable text. The main subtasks involved in PDF text extraction are broken down below.

1. **Loading the PDF document**

   The process begins by opening the PDF file using a PDF parser library. At this stage, the system checks the document structure, metadata, number of pages and determines whether the PDF is text based or image based.

2. **Iterate page by page**
   Pages are processed one page at a time. This allows the system to isolate content, maintain order and handle page specific complexities. Page level iteration ensures that text is extracted in the correct sequence and prevents mixing different sections together.
3. **Extract raw text blocks**
   For each page, the extractor pulls out raw text elements. These can come as paragraphs, lines or fragmented blocks depending on how the PDF was created. The goal here is to capture everything exactly as it appears before applying any cleanup or formatting.

4. **Clean and normalize text**
   Raw extracted text often contains unnecessary line breaks, spacing issues, encoding errors, or hidden characters. This step fixes those problems by:
   - Removing extra whitespace
   - Correcting broken sentences
   - Fixing Unicode or encoding issues
   - Normalizing punctuation and symbols

   This produces clean, readable text ready for downstream processing.

5. **Handle special cases**

Not all PDFs are straightforward. Some require additional techniques:

- Scanned PDFs → Need OCR (Optical Character Recognition) to convert images into text
- Rotated or skewed text → Must be detected and corrected before extraction
- Complex layouts (tables and columns) → Require layout aware parsing to avoid jumbled output

Handling these cases ensures robust extraction across different document types.

6. **Store extracted text for chunking**

After the text has been finalised and cleaned, it is typically stored in a database, JASON, or other structured format. After that, the saved text is prepared for chunking which divides the content.

Tools commonly used

Depending on the type of document and the requirements of the extraction process, a variety of tools and libraries can assist in extracting information from PDF documents. Every tool has unique capabilities:

PyMuPDF: In just a few milliseconds, text, graphics, and layout data can be retrieved using the quick and effective PyMuPDF (formerly known as Fitz) library. It is appropriate for situations where low resources needs and performance are crucial.

PDFPlumber: Third library is renowned for its precise extraction of structured text. PDFPlumber separates table sells, preserves spacing and extracts text in paragraphs with accuracy.

Tesseract OCR: Tesseract is a free and open-source OCR engine. Tesseract is used to create machine readable text from scanned or image-based PDF files that lack digital text. To build a robust, production quality pipeline for parsing PDF documents, developers frequently integrate the features of several libraries.

How text extraction work?

Text extraction from a PDF follows a clear and logical process. The goal is to take whatever is inside the PDF which are paragraphs, headings, tables or even scanned images and turn it into clean readable text that can be used later for chunking's and embeddings, or search. Below is the process of how texts are being extracted:

1. **Opening the PDF**
   a. The first step is simply loading the PDF using a library like PyMuPDF or PDFPlumber. This allows the system to inspect the document, check how many pages it has and then understand whether the content is digital text or scanned text.
2. Go through the PDF page by page
   a. The extractor reads each and every page individually after the file is opened. Maintaining the natural reading sequence and making it simpler to deal with things like page numbers or headers afterward are two benefits of doing it page by page.
3. **Extract text blocks**
   a. The tool extracts the text from each page. It extracts "blocks" like paragraphs, headings or brief text excerpts rather than collecting everything as a single, enormous string. As a result, the text seems cleaner and more like it did in the original source.
4. **Preserve the reading order**
   a. Text is not always stored in PDFs in the order that people read it. As a result, the extractor arranges the blocks in the proper left to right and then top to bottom reading order. This stage guarantees that the sentences are coherent rather than disorganised.
5. **Clean the extracted text**
   a. The raw text usually needs a bit of cleanup. This includes:
      i. Removing unnecessary line breaks
      ii. Cleaning up page numbers or repeating headers and footers
      iii. Fixing odd characters or encoding issues
      iv. Joining broken sentences that got split during extraction
   b. This step transforms messy raw text into something clean and readable.
6. **Return clean text for chunking**
   a. The system returns plain text that is prepared for the subsequent processing step which is typically chunking, embedding, or storing in a vector database, after everything has been cleaned and arranged.

PDF Parsing → Image Extraction

The inclusion of photos in PDF instructions is crucial since they include essential visual components such equipment illustrations and diagrams. All of the images in a PDF must be identified and appropriately extracted during processing in order to be incorporated into subsequent multimodal workflow procedures.

All photos are extracted, converted into a clear, useable format (JPEG or PNG), and given meaningful file names for convenient access. In order to maintain the connection between the image and its source, all pertinent metadata is likewise retained with the image.

How does image extraction actually work?

Typically, PyMuPDF is used to search the PDF pages throughout the extraction process. Every image on a given site will be extracted, saved in PNG format and then kept locally. In order to help in reference back during processing, metadata such as the page number and the coordinates of the specific image which will also be gathered when scanning through PDF pages

Challenges in PDF Parsing

Although a PDF may appear straightforward at first, it is actually very difficult to extract the structured data from a PDF. Optical Character Recognition (OCR) is often required to view the text contained in a PDF since it frequently does not exist in digital form.

Another problem that can have is that some image files were developed using multiple formats, making them more difficult to locate and remove. Additionally, we can discover that a large portion of the text is disorganised and does not follow the logical flow of a paragraph.

Concept Understanding

When we talk about PDF parsing, we simply mean pulling out all the useful information from a PDF manual from a PDF manual. This isn't just the text manuals usually have a mix of:

1. Normal paragraphs
2. Images and diagrams
3. Tables
4. Page layout details
5. The exact positions of objects on a page

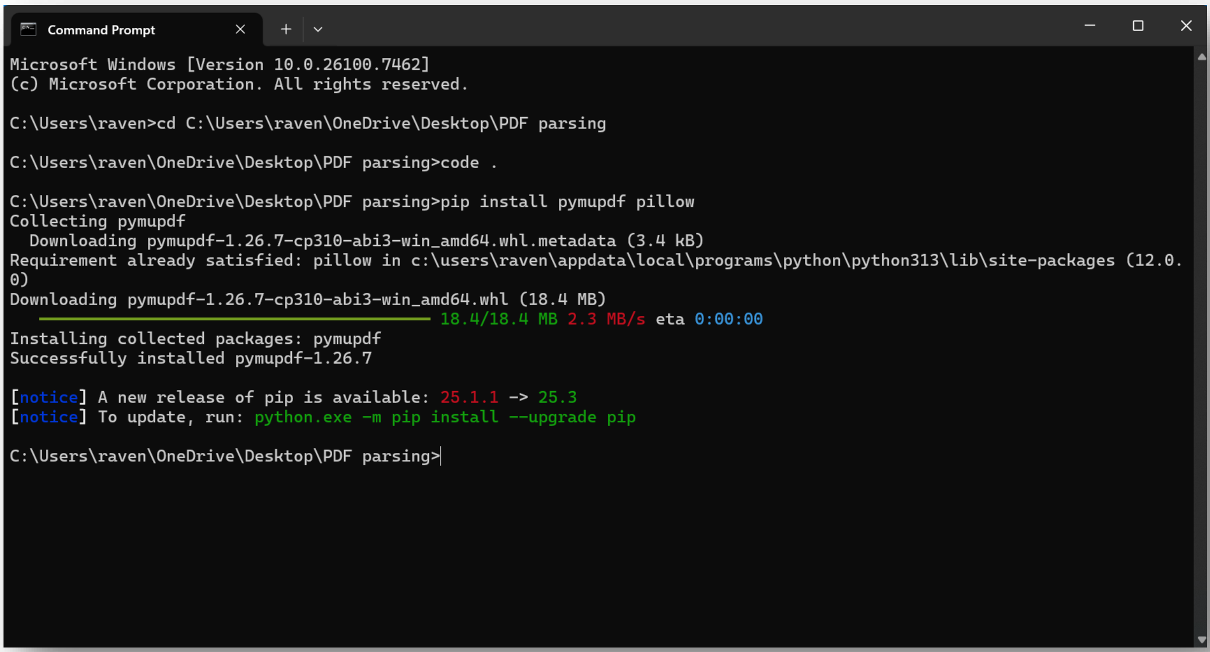
Because these manuals contain different types of content, where we need accurate parsing to build a multimodal RAG system that can understand both text ad visuals. To do these in python, the commonly used libraries are:

1. PyMuPDF (fitz): which is great for extracting both texts and images
2. pdfPlumber: very good at handling text and values
3. Pillow: Used to process and save images

This is the overall process which is simple:

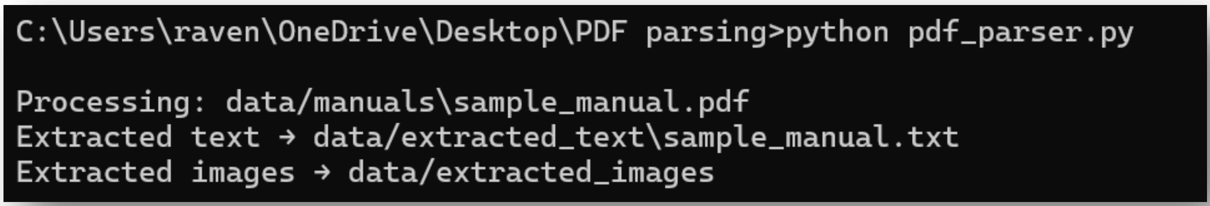
1. Open the PDF
2. Read the text from each page
3. Pull out any images on the page
4. Save those images
5. Collect useful metadata like pages numbers, coordinates and image captions.

Testing



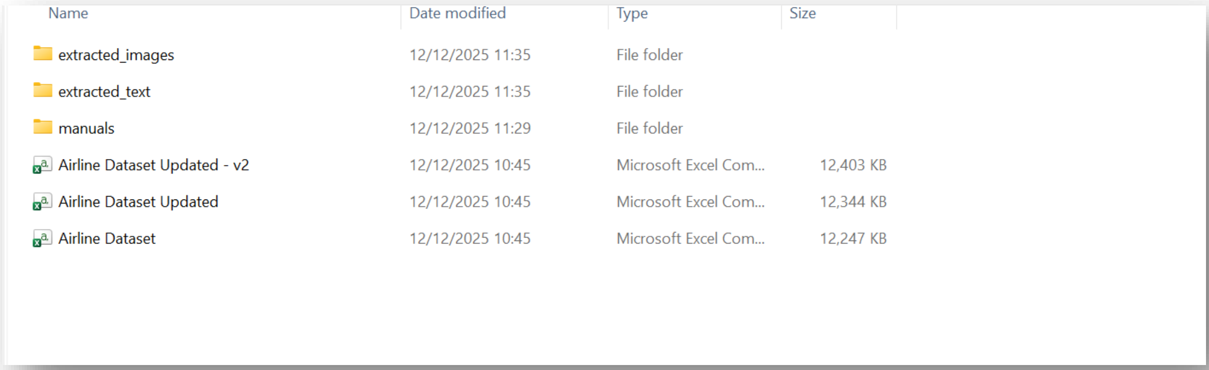
```
Microsoft Windows [Version 10.0.26100.7462]
(c) Microsoft Corporation. All rights reserved.

C:\Users\raven>cd C:\Users\raven\OneDrive\Desktop\PDF parsing

C:\Users\raven\OneDrive\Desktop\PDF parsing>code .

C:\Users\raven\OneDrive\Desktop\PDF parsing>pip install pymupdf pillow
Collecting pymupdf
  Downloading pymupdf-1.26.7-cp310-abi3-win_amd64.whl.metadata (3.4 kB)
Requirement already satisfied: pillow in c:\users\raven\appdata\local\programs\python\python313\lib\site-packages (12.0.
0)
Downloading pymupdf-1.26.7-cp310-abi3-win_amd64.whl (18.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.4/18.4 MB 2.3 MB/s eta 0:00:00
Installing collected packages: pymupdf
Successfully installed pymupdf-1.26.7

[notice] A new release of pip is available: 25.1.1 -> 25.3
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Users\raven\OneDrive\Desktop\PDF parsing>
```

Figure 1: Running the python code in CMD



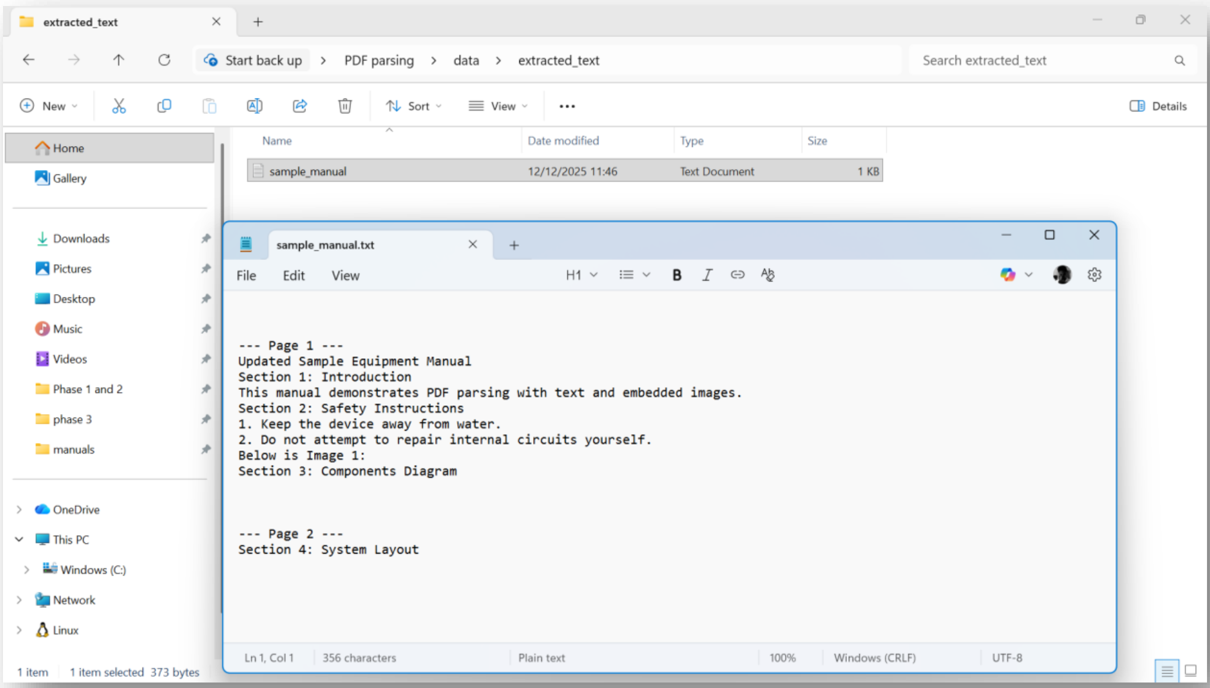
```
C:\Users\raven\OneDrive\Desktop\PDF parsing>python pdf_parser.py

Processing: data/manuals\sample_manual.pdf
Extracted text → data/extracted_text\sample_manual.txt
Extracted images → data/extracted_images
```

Figure 2:Output is successful



Name	Date modified	Type	Size
extracted_images	12/12/2025 11:35	File folder	
extracted_text	12/12/2025 11:35	File folder	
manuals	12/12/2025 11:29	File folder	
Airline Dataset Updated - v2	12/12/2025 10:45	Microsoft Excel Com...	12,403 KB
Airline Dataset Updated	12/12/2025 10:45	Microsoft Excel Com...	12,344 KB
Airline Dataset	12/12/2025 10:45	Microsoft Excel Com...	12,247 KB

Figure 3: Folder after running the script



Start back up > PDF parsing > data > extracted_text

Name	Date modified	Type	Size
sample_manual	12/12/2025 11:46	Text Document	1 KB

sample_manual.txt

```
--- Page 1 ---
Updated Sample Equipment Manual
Section 1: Introduction
This manual demonstrates PDF parsing with text and embedded images.
Section 2: Safety Instructions
1. Keep the device away from water.
2. Do not attempt to repair internal circuits yourself.
Below is Image 1:
Section 3: Components Diagram


--- Page 2 ---
Section 4: System Layout
```

Ln 1, Col 1 | 356 characters | Plain text | 100% | Windows (CRLF) | UTF-8

1 item | 1 item selected 373 bytes

Figure 4: Extracted text

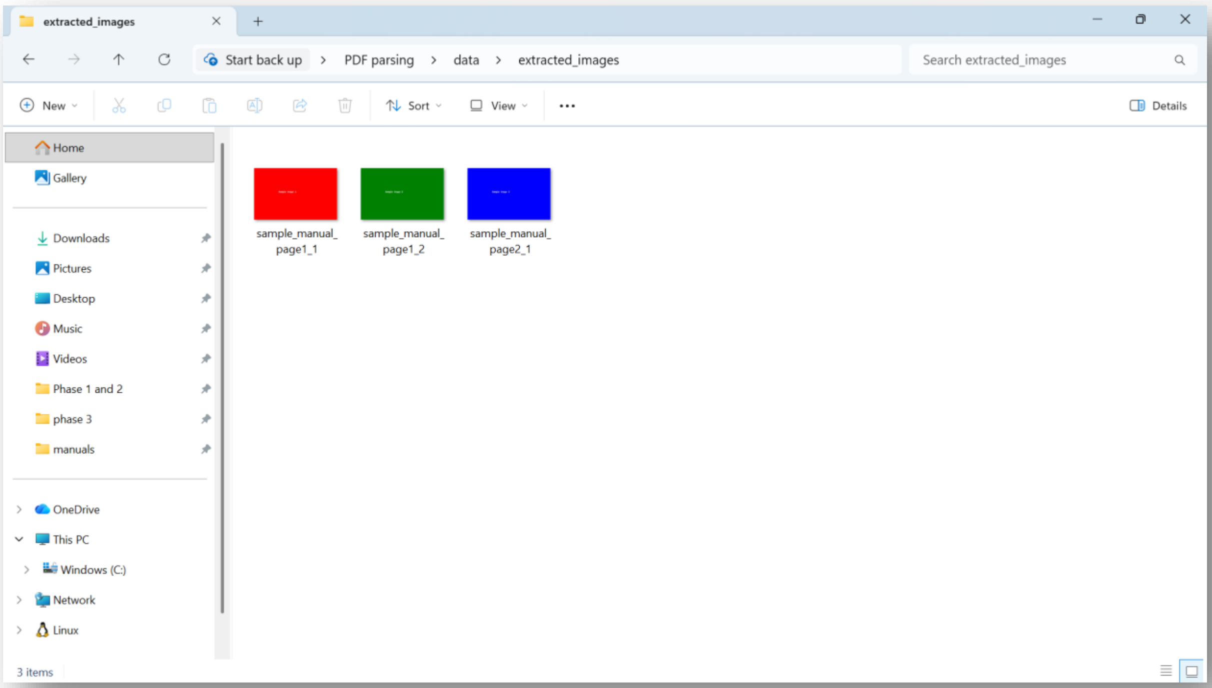


Figure 5: Extracted Images

<u>What I did?</u>

I used PyMuPDF to develop a PDF parser. Each page's text and images are retrieved by the parser, which also stores them independently and obtains metadata such as page and image sizes. With digital PDFs, text extraction was quite successful, but, with scanned PDFs, it was not very clean. Although the image extraction was perfect, some graphics appeared to be low resolution and others to be rotated. Adding OCR functionality for scanned PDFs, would be the next step.

<u>Summary</u>

PDF parsing creates structured informational content from unstructured guides that are difficult to comprehend. During this procedure, the system will extract the text from the PDF, extract any embedded images, extract the table data from the PDF, and build the necessary metadata. The PDF's structured sections will be able to be chunked, embedded, and stored in a vector database for further use.

Text chunking strategies

Chunking is the process of dividing long text into shorter sections known as "chunks", which improves the efficiency of vector searches, embeddings and other retrieval techniques. A full manual content cannot be read by Large Language Models (LLP) in a single sitting. In this sense, chunking will enhance both the LLM's accuracy and performance.

Why is chunking needed?

Chunking is important for AI because it divides large datasets into manageable chunks that helps the LLM's overcome context window limitations, increase the processing efficiency, improve the retrieval accuracy, lowering the computational cost and also help the models concentrate on useful semantic meaning for better, clearer AI outputs.

Key reasons for chunking in AI

1. Overcomes LLM context limits by splitting long documents into smaller chunks that fit within token boundaries.
2. Improves retrieval accuracy in RAG by helping the system find the right information and reduce hallucinations.
3. Boosts processing efficiency because smaller chunks are quicker and cheaper to embed, store and search.
4. Preserves context and meaning through strategic chunking rather that random splits.
5. Optimizes embedding quality while balancing precision and thematic understanding
6. Supports scalability, allowing the AI systems to handle increasing data volumes without performance issues.

Common chunking methods

Using techniques like Fixed-Size (by tokens/chars), Sentence/Paragraph-Based, Sliding Window (with overlap), Recursive Chunking (hierarchical separators), advanced Semantic Chunking (by topic/meaning) or Agentic Chunking (AI-driven decisions) to balance context preservation and manageability, common chunking methods in AI, particularly for Retrieval-Augmented Generation (RAG), seek to divide lengthy documents into meaningful segments (chunks) for better context.

1. **Basic / Structured Methods**
   a. **Fixed-Size Chunking** → Splits text into equal lengths, simple but may break context
   b. **Line by line / Sentence** Based → Each line or sentence becomes a chunk, keeping grammar intact
   c. **Paragraph Based** → Uses natural paragraph breaks, preserving coherent ideas.
   d. **Section / Heading Based** → Splits according to document structure, keeping topics grouped.
2. **Advanced / Context Aware Methods** →
   a. **Sliding Window** → Creates overlapping chunks so context flows smoothly between splits.
   b. **Recursive Chunking** → Splits using multiple separators (for example paragraph → sentence → space) to avoid cutting related text apart.
   c. **Semantic Chunking** → Breaks text where the meaning or topic changes, guided by embeddings.

d. **Content Aware / Structure Aware** → Adapts chunking to content type (code, tables, lists) for better accuracy.
e. **Entity Based Chunking** → Groups text around important names, places, or concepts to keep related info together.
f. **Hybrid Chunking** → Combines different techniques (eg → semantic + sliding window) for stronger performance.

So recommended strategy?

Hybrid chunking → Combines multiple chunking methods (e.g., semantic + sliding window + paragraph-based) to balance context, accuracy, and retrieval performance, making it the most reliable approach for real-world AI systems.

What Metadata should each chunk have?

Every chunk of text should include important metadata so it can be traced, organized and efficiently retrieved. The key fields are:

1. chunk_id → A unique identifier for the chunk.
2. page_number → The PDF page the chunk came from.
3. text → The actual content of the chunk
4. section_title → The heading or subheading the chunk belongs to.
5. token_count → Number of tokens, useful for managing LLM limits
6. source_pdf → The name of the original PDF file.
7. chunk_type → Specifics that this chunk is text
8. timestamp → When the chunk was created or processed.

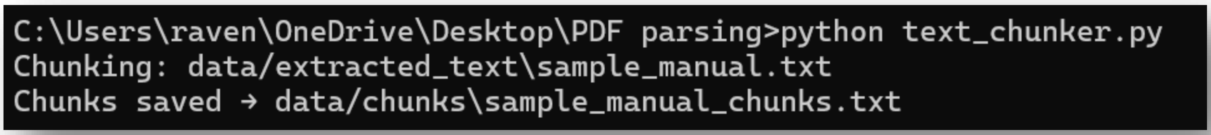
Concept Understanding

The main purpose of chunking is to make it easier for the system to search through a huge PDF file by breaking material up into smaller and meaningful chunks. There are a number of techniques used for this which includes:

1. Splitting the text into sections of the same size
2. Splitting it based on meaning (semantic chunking)
3. Creating sections that barely overlap
4. Or simply use each page in its full form

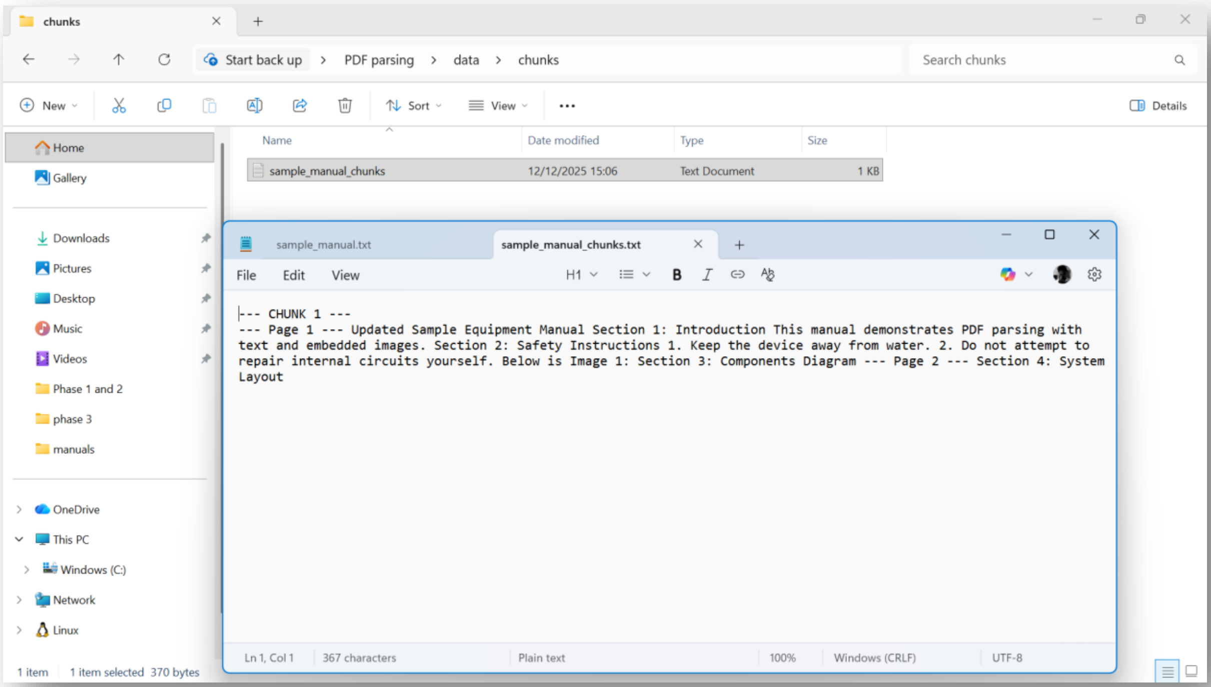
Effective chunking of the data will enable a RAG system to find the right information more accurately, improving its performance.

Testing



```
C:\Users\raven\OneDrive\Desktop\PDF parsing>python text_chunker.py
Chunking: data/extracted_text\sample_manual.txt
Chunks saved → data/chunks\sample_manual_chunks.txt
```

Figure 6: Testing in CMD



```
--- CHUNK 1 ---
--- Page 1 --- Updated Sample Equipment Manual Section 1: Introduction This manual demonstrates PDF parsing with
text and embedded images. Section 2: Safety Instructions 1. Keep the device away from water. 2. Do not attempt to
repair internal circuits yourself. Below is Image 1: Section 3: Components Diagram --- Page 2 --- Section 4: System
Layout
```

Figure 7: Sample Manual Chunks

Using PyMuPDF, I have developed a PDF parser. Each page's text and photos were extracted by the parser and stored separately, along with metadata like the page number and image dimensions. For digitally generated PDFs, text extraction performs rather well. Nevertheless, the output from scanned PDFs was noisy. Although several schematics are rotated or of low quality, the image was accurate. To handle scanned PDFs by adding OCR functionality, a few further steps are needed.

Summary

Chunking makes it easier to organise, embed, and retrieve huge PDF documents by breaking it up into shorter, more manageable chunks. A smooth, dependable pipeline for creating embeddings is ensured, vector search speed is increased, and RAG performance is strengthened by using the proper chunking method.

Metadata and linking images to text

One of the most important steps in creating a multimodal RAG system is connecting images to text. Technical manuals typically contain a large number of illustrations, labels, tables, and other visual aids meant to highlight specific textual passages. These images convey extremely important information and must be appropriately connected to the instructions or phrases they depict.

Otherwise, the system will handle text and images separately if they are not correctly linked. Therefore, the RAG system may just retrieve the written explanations and not give the supporting diagrams when a query calls for some kind of visual context, such as wiring diagrams, component layout, safety symbols, or step-by-step drawings. This will compromise the manual's informational integrity.

In order for the RAG system to receive both text and images simultaneously, accurate linkage ensures that each visual component stays linked to its pertinent content. Building a dependable multimodal retrieval system that can handle intricate, technical, and instruction-rich materials requires this.

Metadata and Linking images to texts

1. **Metadata organizes the document's structure**
   a. It records essential information such as page numbers, section titles and element positions. This creates a clear map of how every part of the document is arranged.
2. **Metadata links text chunks to related visual content**
   a. Text blocks can be connected to diagrams, images or captions using shared attributes like page location or bounding boxes. This ensures that descriptions and visuals stay paired together.
3. **Metadata keeps tables aligned with their context**
   a. Tables often summarise or expand on nearby text. By storing table specific metadata, such as extraction coordinates and associated section headings, the system knows exactly where the table belongs and what it supports.
4. **Metadata captures page layout information**
   a. Details about the layout help maintain the original flow of information so the system retrieves content in a meaningful sequence.
5. **Metadata enables precise context aware search**
   a. When all elements are properly connected, a query can trigger retrieval of both text and relevant image, table or diagram. This significantly improves the accuracy of multimodal RAG systems.

Implement Approach

Phase 3 of this projects document processing procedure now includes metadata generation and image text pairing. After the PDF manuals are parsed, the text is retrieved and divided into manageable chunks for embedding. Images are retrieved on a different location with names that includes details about the original document and page.

Each text chunk then has metadata created for it, including chunk identification, page citation, and the name of the original document. Text and image elements can be connected using the previously mentioned metadata elements. This ensures that both components are connected throughout the whole process.

The succeeding process, such as vector representation, similarity search in multimodal content and RAG answer generation are made easier by this logical handling of the input.

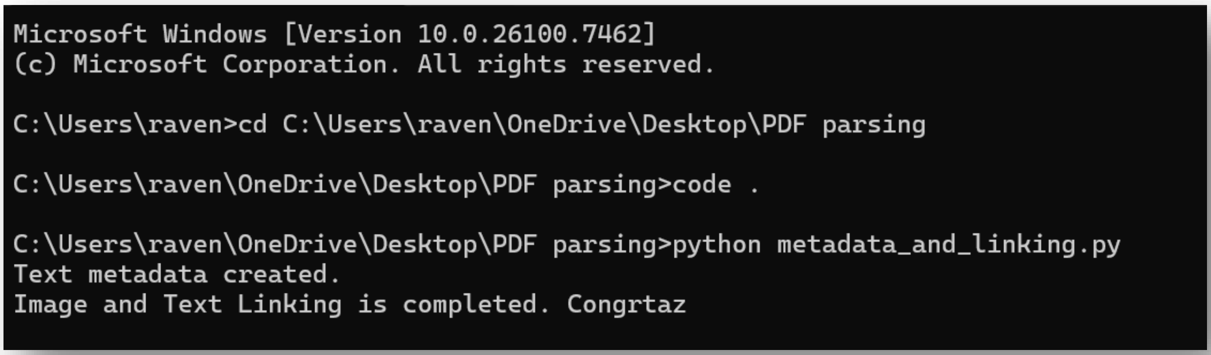
Linking Strategy used in this project

Page level metadata linking is the main linking technique used in the project in concern. The text passages that come from the same page in the original PDF are linked to each extracted image. This is a practical and effective method of preserving text visual contextual alignment without needlessly increasing computational complexity.

Technical manuals, which typically arrange their diagrams and graphics adjacent to the instructions or descriptions they illustrate, prove to be especially well suited for page level linking. This basic approach provides good baseline performance and simplicity, but it could miss fine grained interactions in documents with dense layout.

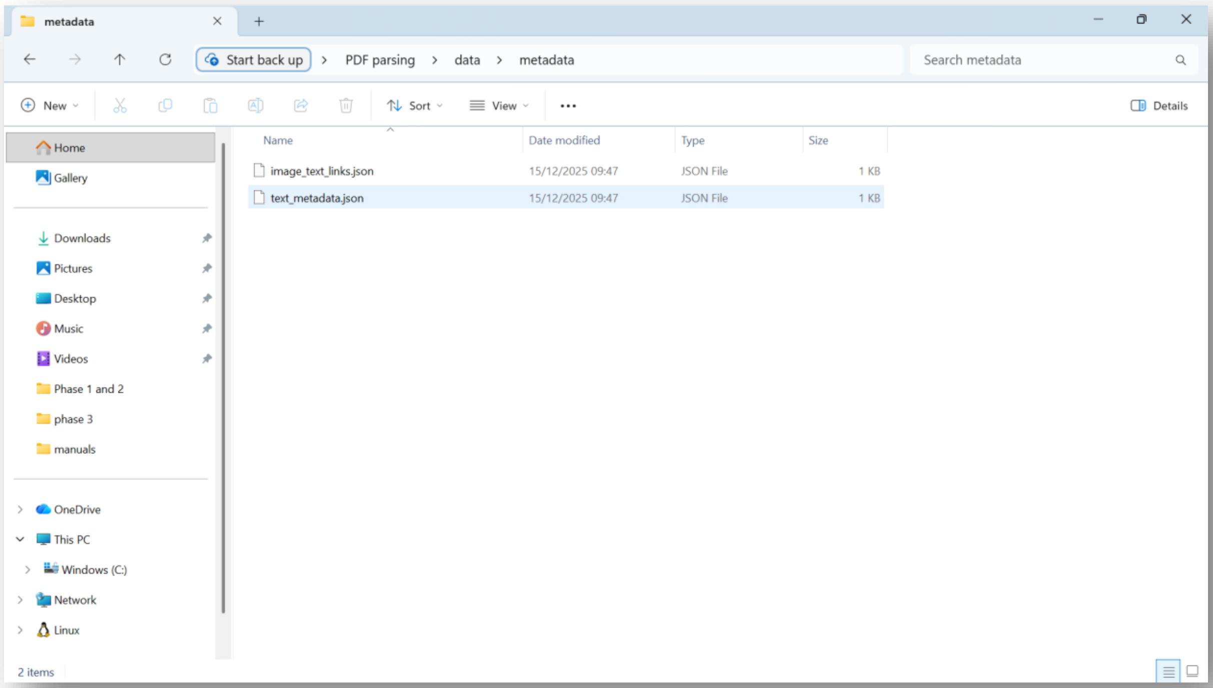
This might be expanded to include coordinate-based linking which uses positional data to link images to nearby text or semantic linking, which uses multimodal embeddings like CLIP to link images to important text chunks based on meaning rather than layout.

Testing



```
Microsoft Windows [Version 10.0.26100.7462]
(c) Microsoft Corporation. All rights reserved.

C:\Users\raven>cd C:\Users\raven\OneDrive\Desktop\PDF parsing

C:\Users\raven\OneDrive\Desktop\PDF parsing>code .

C:\Users\raven\OneDrive\Desktop\PDF parsing>python metadata_and_linking.py
Text metadata created.
Image and Text Linking is completed. Congrtaz
```

Figure 8: Creating the link between Image and Text



Name	Date modified	Type	Size
image_text_links.json	15/12/2025 09:47	JSON File	1 KB
text_metadata.json	15/12/2025 09:47	JSON File	1 KB

Figure 9: Links being created in the folder

The working code

To generate structured metadata for text segments and link photos to related content based on a page level correlation, a single Python script has being used. The Python script creates a JDON file for later multimodal RAG learning after receiving inputs from the parsing of PDF files and chunking of texts.