DAMG 7245 – Big Data Systems and Intelligent Analytics.
Group – 7

Arvind Ranganath Raghuraman
Kaushik Jayaprakash
Riya Singh

Project Summary.

In our Assignment, we have built a web app that will take a pdf or url, convert it into markdown and store the output files in S3. So we have used FastAPI for Backend and Streamlit for Frontend. We have created different endpoints for Different Services such as PyMUPDF, LXML, Microsoft Doc Intelligence, Tesseract and Beautiful Soup. For these services, we have a common router wherein we process all the endpoints into a single folder and our frontend makes the request to it and then we deployed it. Overall, we are able to extract Images, Tables and Text from PDFs with accuracy albeit some little bottlenecks.

POC on Data Extraction for PDFs

So we did some proof of concepts wherein we tried and tested what work and what doesn't best to our knowledge.

1. Data Extraction:

○

Summary Table

Library	Text Extraction	Tables	Images	OCR Support	Manipulation
PyPDF2	✓	✗	✗	✗	✓
pdfplumber	✓	✓	Limited	✗	✗
PyMuPDF (fitz)	✓	✓	✓	✗	✓
Camelot	✗	✓	✗	✗	✗
Tabula-py	✗	✓	✗	✗	✗
PDFMiner	✓	✗	✗	✗	✗
Tika	✓	✗	Limited	✗	✗
Slate	✓	✗	✗	✗	✗
Pytesseract	✓ (OCR-based)	✗	✓	✓	✗
pdfrw	✗	✗	✗	✗	✓

Key Differences

When to Use Which Library?

PyMuPDF:

For text extraction and Image Extraction.

PyTesseract:

For extracting text from scanned PDFs or images.

We can Combine with PyMuPDF or PDFPlumber for hybrid text extraction.

PDFPlumber:

For extracting tables, structured data, or text with precise positioning.

When you need detailed access to PDF elements (e.g., fonts, shapes).

PyPDF2:

For basic text extraction or PDF manipulation (e.g., merging, splitting).

When working with simple PDFs with selectable text.

Conclusion

Use PyMuPDF for general-purpose PDF parsing and modification.

Use PyTesseract for OCR on scanned PDFs.

Use PDFPlumber for extracting tables or structured data.

Use PyPDF2 for basic tasks like merging, splitting, or simple text extraction.

For complex tasks, you can combine these libraries (e.g., PyMuPDF for rendering PDFs as images and PyTesseract for OCR).

Aspect	Python Parsing (Manual)	Microsoft Document Intelligence (Azure)
Technology	Open-source libraries (PyPDF2, pdfplumber, etc.)	Enterprise AI service (Azure Form Recognizer)

Aspect	Python Parsing (Manual)	Microsoft Document Intelligence (Azure)
Capability	Basic text, table, and image extraction	Advanced OCR with AI for text, tables, charts, and layouts
Scanned PDFs	Limited or requires additional OCR libraries	Fully supported with AI-powered OCR
Customization	High – you control the logic entirely	Limited – depends on the AI model
Setup	Install Python libraries locally	Requires Azure account and setup
Cost	Free (open source)	Paid service (Azure pricing applies)
Accuracy	Varies based on PDF structure	High, especially for complex PDFs
Output Format	Extracted manually into Markdown	AI processes and generates structured data

Open-source tools like PyPDF2, pdfplumber, PyMuPDF (Fitz), Tesseract (OCR), and BeautifulSoup offer a flexible and cost-effective way to extract text, tables, and images from PDFs. They provide high accuracy, especially with structured documents, but may struggle with noisy or scanned PDFs when relying on OCR. While these tools allow full customization and no API rate limits, they require coding expertise and efficient memory management, especially for large and complex PDFs. Although free and open-source, their setup and maintenance can be time-consuming for large-scale operations.

Enterprise solutions like Microsoft Document Intelligence, Google Document AI, and AWS Textract offer high-speed, AI-powered document processing with optimized cloud infrastructure. They excel at text and table extraction, even from scanned PDFs, but often lack image extraction support. These services require no coding, making them user-friendly, but they offer less control over data extraction and have batch processing limitations. While they are scalable for enterprises, they can become expensive for large volumes. Additionally, their pre-trained AI models handle complex layouts well but provide limited customization options compared to open-source tools.   So following are the results obtained.

Comparison and Conclusion of POC

1. Accuracy & Output Format – Microsoft Form Recognizer and Google Document AI both provide structured outputs, but their accuracy and formatting may vary depending on the document type. Google Document AI often excels in handling complex layouts with dense text, while Form Recognizer is optimized for structured forms.
2. Customization & Training – Microsoft Form Recognizer allows custom model training for specific document types, while Google Document AI provides pre-trained models for various document formats, making it easier to get started without training.
3. Processing Speed & Scalability – Google Cloud's ability to process entire pages gives it an advantage for large documents, whereas Microsoft Form Recognizer's two-page limit may require workarounds for bulk processing.
4. Table Extraction Complexity – Google Document AI's table extraction method differs, potentially making post-processing more complex depending on the structure of the tables.

POC on Webpages

We also did a POC on Beautiful soup and lxml, we found out the following. BeautifulSoup and LXML are both widely used for web scraping, but when it comes to extracting structured elements like tables and images, they exhibit key differences. BeautifulSoup is excellent for parsing HTML/XML with a simple and intuitive API, making it easier to navigate and manipulate webpage elements. However, it lacks built-in support for extracting tables and images efficiently. While it can locate <table> tags, extracting structured data often requires additional logic, such as looping through rows and columns manually. Similarly, image extraction requires handling <img> tags and reconstructing full URLs, making it less effective for direct extraction. On the other hand, LXML is a more powerful and faster XML/HTML parser that can handle large datasets efficiently, but it also struggles with tables and images in real-world scraping scenarios. While LXML's XPath support makes element traversal more robust, it doesn't inherently extract tabular data in a structured format like libraries such as Pandas or PyMuPDF can. For image extraction, LXML faces similar challenges as BeautifulSoup, requiring extra steps to reconstruct image links and download files. In conclusion, while both tools excel at parsing and navigating web pages, they are not ideal for extracting tables and images directly, and more specialized libraries such as PyMuPDF for PDFs or Pandas for tables should be considered for better accuracy and efficiency.

POC on Docling/Markitdown

Finally we tried our hands on Docling and Markitdown. Docling is effective for extracting text and structuring it in a Markdown-compatible format, but it falls short when it comes to handling images. It does not properly extract embedded images from documents, requiring additional processing to preserve visual elements. Markitdown, on the other hand, is specialized in converting structured text into Markdown format, making it useful for formatting documents in a lightweight and readable way. However, it does not support images or tables, which significantly limits its utility for documents that contain rich content beyond plain text. While both tools perform well for text-based extraction and Markdown conversion, neither is a comprehensive solution for extracting images and tables, making them less suitable for processing complex document structures. For a more complete solution, integrating additional tools like PyMuPDF or Pandas for structured extraction could help overcome these limitations.

Comparison Summary:

Microsoft Docs AI and Google Document AI both efficiently parse PDFs but do not support image extraction. Google Document AI uses a different process for table extraction and can handle entire pages, whereas Microsoft Form Recognizer has a two-page processing limit. While integration is straightforward, enterprise features remain restricted in the free versions. Google Document AI is more scalable for large documents, whereas Form Recognizer supports custom model training for specific document types.

Conclusion:

Our findings confirm that open-source tools are highly customizable and cost-effective, but they require manual configurations to handle images, tables, and charts. Enterprise AI solutions offer higher accuracy and scalability but come at a financial cost and with limited features. The best way is to provide users with an option to choose their own preference which can work the best results for them.