Wed Dec 25 2024

Member-only story

Why PyMuPDF4LLM is the Best Tool for Extracting Data from PDFs (Even if You Didn’t Know You Needed It)

Anoop Maurya · Follow
Published in Python in Plain English
7 min read · 5 days ago

Listen Share More

Photo by Claudio Schwarz on Unsplash

Stuck behind a paywall? Read for Free!

You know how it goes when working with PDFs, especially when those PDFs are dense research papers or documents packed with tables, images, and metadata. For anyone in the AI world — particularly those fine-tuning large language models (LLMs) or building knowledge retrieval systems like RAG (Retrieval-Augmented Generation) — extracting the right data is crucial. But let's be honest: not all PDF extraction tools are up to the task.

In my journey through various data extraction tools, I've come across many that *almost* did the job, but something was always missing. Either they couldn't capture the structure well, or they fell flat when extracting complex tables and images. That was until I discovered **PyMuPDF4LLM** — a tool that quickly became my go-to for all things PDF. But enough with the intro — let's dive into what makes PyMuPDF4LLM such a game changer.

The PDF Dilemma We've All Faced

Picture this: you're working on a project that involves fine-tuning an LLM for domain-specific tasks — say, legal documents or scientific papers. You need to extract text while maintaining the structure, pull out some tables, and get high-quality images for model training. Sounds simple, right? Wrong.

With most PDF tools, you either get garbled text that loses context or — worse — images that are more "potato quality" than useful data. Worse still, some tools completely miss tables or mess up word alignments. If you're lucky, you might get a decent text output, but what about the images and tables that are often key to training more comprehensive models?

That's where **PyMuPDF4LLM** comes in. It's flexible, fast, and can handle just about any data extraction task thrown at it. Whether you're building an LLM-based application or just need to prepare data for fine-tuning, this tool has you covered.

. . .

Why PyMuPDF4LLM Works So Well for LLM Tasks

1. **Versatile Markdown Extraction**: Whether you're extracting basic text or structuring it for fine-tuning, PyMuPDF4LLM delivers everything in an easily readable Markdown format. It's fast, efficient, and ensures that everything — text, images, tables, and metadata — is captured in a format that LLMs can work with directly.

2. **Flexibility in Application**: PyMuPDF4LLM isn't just good for text extraction. You can extract data by page, save it in formats needed for RAG tasks, or directly pipe it into frameworks like LlamaIndex (for embeddings and document search). The beauty is in its flexibility.

3. **Comprehensive Extraction**: This tool doesn't just scrape plain text. It also offers **image extraction**, **word-by-word chunking**, **table extraction**, and more. This kind of granularity is invaluable when you're working with PDFs in various complex applications, from knowledge base creation to sentiment analysis.

. . .

If you enjoyed the article and want to show some love:

- **Clap** 50 times — each one helps more than you think! 👏

- **Follow** me here on Medium and subscribe for free to catch my latest posts. 🫶

- Let's connect on **LinkedIn**, check out my projects on **GitHub**, and stay in touch on **Twitter**!

. . .

PyMuPDF4LLM in Action: Extracting Data with Precision

Let me walk you through some real-life examples of how I've used PyMuPDF4LLM in my own work, from basic text extraction to more complex use cases like chunking, table extraction, and more. The versatility of this tool is truly impressive, making it the best option for many LLM-related tasks.

First install PyMuPDF4LLM using pip:

```
pip install pymupdf4llm
```

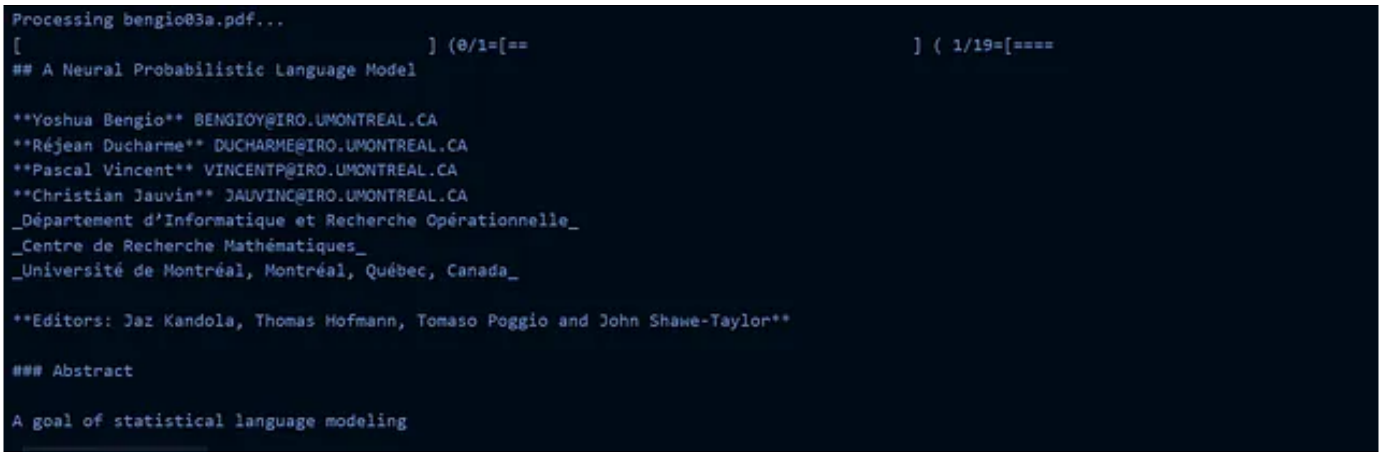
Use Case 1: Basic Markdown Extraction

This one is simple: you have a PDF and just want the content in a clean, Markdown-friendly format. PyMuPDF4LLM can extract the text, complete with headers, lists, and other formatting, making it easy to work with.

```
import pymupdf4llm

# Extract PDF content as Markdown
md_text = pymupdf4llm.to_markdown("input.pdf")
print(md_text[:500])  # Print first 500 characters
```

In this example, I've used PyMuPDF4LLM to extract the contents of a research paper into a Markdown format. Why Markdown? It's ideal for fine-tuning models because it retains structure and formatting, which can be crucial for generating coherent responses from LLMs.



```
Processing bengio03a.pdf...
[                                        ] (0/1=[==                                        ] ( 1/19=[====
## A Neural Probabilistic Language Model

**Yoshua Bengio** BENGIOY@IRO.UMONTREAL.CA
**Réjean Ducharme** DUCHARME@IRO.UMONTREAL.CA
**Pascal Vincent** VINCENTP@IRO.UMONTREAL.CA
**Christian Jauvin** JAUVINC@IRO.UMONTREAL.CA
_Département d'Informatique et Recherche Opérationnelle_
_Centre de Recherche Mathématiques_
_Université de Montréal, Montréal, Québec, Canada_

**Editors: Jaz Kandola, Thomas Hofmann, Tomaso Poggio and John Shawe-Taylor**

### Abstract

A goal of statistical language modeling
```

image by Author

Use Case 2: Extracting Specific Pages

Sometimes you only need part of a document — maybe just a specific section for training. With PyMuPDF4LLM, you can target exactly which pages to extract, saving both time and resources.

```python
import pymupdf4llm

# Extract only pages 10 and 11
md_text = pymupdf4llm.to_markdown("bengio03a.pdf", pages=[10, 11])
print(md_text[:500])  # Print first 500 characters
```

This feature is particularly helpful when dealing with massive documents. For example, extracting specific chapters or sections makes it easy to train LLMs on only the most relevant information.

Use Case 3: Saving Markdown to a File

After extracting the data, you might want to save it in a format that's easy to revisit or share. PyMuPDF4LLM makes it a breeze.

```python
import pymupdf4llm
import pathlib

md_text = pymupdf4llm.to_markdown("bengio03a.pdf")
pathlib.Path("output.md").write_bytes(md_text.encode())
print("Markdown saved to output.md")
```

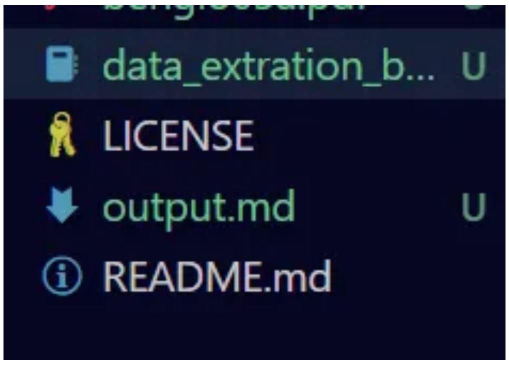


image by Author

Storing your extracted data in Markdown allows for easy collaboration or review, especially when preparing data for LLM fine-tuning. The flexibility in saving makes it perfect for building datasets incrementally.

Use Case 4: Extracting Data as LlamaIndex Documents

One of the trickier tasks in LLM work is transforming PDFs into a document format compatible with LlamaIndex (formerly GPT Index). PyMuPDF4LLM simplifies that process.

```python
import pymupdf4llm

llama_reader = pymupdf4llm.LlamaMarkdownReader()
llama_docs = llama_reader.load_data("bengio03a.pdf")
print(f"Number of LlamaIndex documents: {len(llama_docs)}")
print(f"Content of first document: {llama_docs[0].text[:500]}")
```

Whether you're using these documents for embedding, search, or any knowledge-based tasks, PyMuPDF4LLM does the heavy lifting, ensuring the data is structured and easy to query.

Use Case 5: Image Extraction

Extracting images along with text is often overlooked but incredibly important, especially for documents containing figures, diagrams, or charts. PyMuPDF4LLM handles it seamlessly.

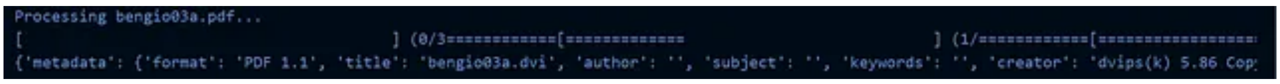
```
md_text_images = pymupdf4llm.to_markdown(doc="bengio03a.pdf",
                                         pages=[1, 11],
                                         page_chunks=True,
                                         write_images=True,
                                         image_path="images",
                                         image_format="jpg",
                                         dpi=200)
print(md_text_images[0]['images'])  # Print image information from the
```

This is perfect for when you're working with technical documents or reports that include lots of visual data. The quality of the extracted images is impressive and works well even for high-resolution images.

Use Case 6: Chunking with Metadata

For fine-tuning tasks, breaking down PDFs into manageable chunks with metadata is critical. PyMuPDF4LLM allows you to do this efficiently.

```
md_text_chunks = pymupdf4llm.to_markdown(doc="bengio03a.pdf",
                                         pages=[0, 1, 2],
                                         page_chunks=True)
print(md_text_chunks[0])  # Print the first chunk
```



```
Processing bengio03a.pdf...
[                                        ] (0/3============[=============                                        ] (1/============[==================
{'metadata': {'format': 'PDF 1.1', 'title': 'bengio03a.dvi', 'author': '', 'subject': '', 'keywords': '', 'creator': 'dvips(k) 5.86 Cop
```

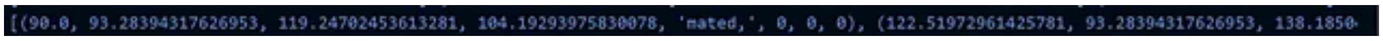
image by author

This use case is fantastic for preparing data for LLMs, where having properly chunked text with metadata can significantly improve model performance.

Use Case 7: Word-by-Word Extraction

Sometimes, you need more control over the text you extract, especially for tasks like sentiment analysis or text generation that require precise word-level data.

```
md_text_words = pymupdf4llm.to_markdown(doc="bengio03a.pdf",
                                        pages=[1,2],
                                        page_chunks=True,
                                        write_images=True,
                                        image_path="images",
                                        image_format="jpg",
                                        dpi=200,
                                        extract_words=True)
print(md_text_words[0]['words'][:5])  # Print the first 5 words from t
```



```
[(90.0, 93.28394317626953, 119.24702453613281, 104.19293975830078, 'mated,', 0, 0, 0), (122.51972961425781, 93.28394317626953, 138.1850
```

image by author

This level of extraction is incredibly helpful for NLP tasks, where word positioning and context matter a great deal.

Use Case 8: Table Extraction

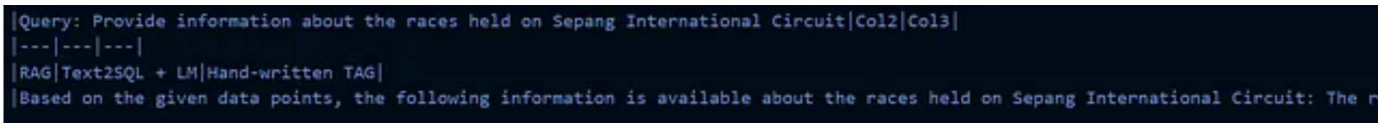
Last but not least, tables are notoriously hard to extract from PDFs without losing their formatting. PyMuPDF4LLM handles this gracefully, making sure tables come out clean and ready to use.

```python
import pymupdf4llm
import json

md_text_tables = pymupdf4llm.to_markdown(doc="bengio03a.pdf",
                                         pages=[12],  # Specify pages
                                         )

print(md_text_tables)
```

Tables often hold critical data, especially in technical documents, and PyMuPDF4LLM ensures that these are extracted accurately, ready for downstream analysis.



```
|Query: Provide information about the races held on Sepang International Circuit|Col2|Col3|
|---|---|---|
|RAG|Text2SQL + LM|Hand-written TAG|
|Based on the given data points, the following information is available about the races held on Sepang International Circuit: The r
```

image by author

Final Thoughts: Is PyMuPDF4LLM the Best? I'd Say So!

After using PyMuPDF4LLM extensively across multiple projects, I can confidently say it's hands down the most versatile and reliable PDF data extraction tool for LLM tasks. It's not just about getting the text or images — it's about getting **everything** you need in a format that LLMs can easily work with.

So, whether you're building a RAG system, fine-tuning an LLM, or just need a solid extraction tool for PDFs, give PyMuPDF4LLM a try. It's streamlined, efficient, and in my experience, it simply works.

. . .

Additional Resource:

Complete Code:https://github.com/imanoop7/PDF-Data-Extractor
Official WebSite: https://pymupdf.readthedocs.io/en/latest/pymupdf4llm/#
My GitHub: https://github.com/imanoop7
LinkedIn: www.linkedin.com/in/anoop-maurya-908499148
X: https://x.com/imanoop_7