Benchmarking Learning-Based and Rule-Based Chunking Methods and Constructing a Mature Industrial Data-Preprocessing Pipeline

Haonan Cai Wilson Zhang James

May 2025

1 Abstract

The primary objective of this initiative is to identify the most effective models for PDF chunking—breaking down complex documents into semantically meaningful segments—to support an agentic AI startup in constructing a robust data automation pipeline. The process involves structured chunking of PDFs, converting extracted content into well-organized JSON files to preserve hierarchy and context, and transforming these JSON structures into tokens optimized for downstream AI tasks (e.g., embeddings, fine-tuning LLMs). Given the unique challenges of datasets across different domains—such as the complex layouts and inline equations in scientific documents, the dense tabular structures and numerical data in financial reports, and the hierarchical text structures in legal documents—we aim to evaluate and compare two distinct chunking strategies: rule-based methods (e.g., PyMuPDF) and learning-based methods (e.g., LayoutParser). By rigorously testing these approaches across metrics like segmentation accuracy, semantic coherence, and downstream usability (e.g., in retrieval-augmented generation (RAG) workflows), we seek to determine which method is better suited for different types of content. The ultimate goal is to enhance data usability for AI agents, enabling the startup to scale knowledge-intensive operations with precision and efficiency, while ensuring robust integration into workflows like autonomous decision-making systems.

2 Introduction

PDFs are a ubiquitous format for sharing and archiving information, particularly in knowledge-intensive domains such as academia, scientific research, and industry. Scientific documents, which often include dense text, tables, figures, equations, and hierarchical structures, represent a unique challenge for information extraction due to the rigid and highly variable nature of the PDF format. For AI systems to process and leverage the information contained in these documents effectively, it is essential to first segment them into semantically meaningful units while preserving contextual and hierarchical relationships. This process, known as PDF chunking, is a foundational step in modern AI-driven workflows, particularly in applications such as retrieval-augmented generation (RAG), embedding generation, and fine-tuning large language models (LLMs). The need for robust PDF chunking is further underscored by the rise of autonomous decision-making systems and agentic AI applications, which depend on structured and semantically coherent data pipelines. Poor chunking can lead to fragmented or incoherent representations of data, which, in turn, degrade the performance of downstream AI tasks. Despite the critical importance of this step, the existing methods for PDF chunking—broadly categorized into rule-based and learning-based approaches—vary significantly in their applicability and effectiveness when applied to complex scientific documents. Rule-based methods, such as those implemented in tools like PyMuPDF, rely on predefined heuristics and structural parsing rules to extract content based on the layout of the document. These methods are lightweight and easy to implement but often struggle with variability in document structures, particularly in scientific PDFs with dense content and intricate formatting. On the other hand, learning-based methods, such as LayoutParser and other deep learning models, leverage labeled data to learn patterns for detecting and segmenting content. While these methods are more flexible and capable of handling diverse document types, they come with significant computational requirements and demand access to large, annotated datasets for training.

3 Methodology

In this section, we provide an overview of the main features of the dataset we used for our study, namely the DocLayNet dataset, and outline the steps we took to generate the ground truth text.Additionally,we discuss the evaluation metrics utilized and the PDF parsers evaluated in our analysis.

3.1 Dataset

IBM DocLayNet

DocLayNet is a large-scale, human–annotated corpus of ∼81 k PDF pages spanning 11 layout classes (*text, title, list, table, figure*, etc.) and multiple document genres (financial reports, patents, technical papers). Each page is released with the original PDF, a 300dpi rendered image, and COCO-style JSON containing polygon masks, bounding boxes and reading order.

How we leverage it

For every page :

1. feed the PDF to our PyMuPDF–based pipeline (B);
2. feed the rendered image to our Faster RCNN+Tesseract OCR pipeline (A);
3. use the official annotation JSON as *ground truth*.

Both pipelines output json files corresponding to each input, which we benchmark against the annotations with families of metrics

3.2 Metrics

We employ a comprehensive set of metrics to quantitatively assess each pipeline's output against the ground truth. Key metrics include:

- **Token-Level Precision, Recall, F1:** These measure how well each pipeline captures the exact tokens of the text. Precision (P) is the fraction of extracted tokens that are correct, and Recall (R) is the fraction of ground-truth tokens that were extracted. The F1 score is the harmonic mean $F_1 = \frac{2PR}{P+R}$. We compute token matches in a case-insensitive manner and allow minor fuzzy matching to accommodate OCR variants (e.g., minor spelling differences or punctuation). High precision indicates minimal spurious text, while high recall indicates few omissions – both are crucial for information-extraction tasks. A high F_1 means the parser captured most content accurately. For example, if a pipeline misses a paragraph or garbles a math symbol, recall will drop accordingly. Token-level scores directly reflect the parser's ability to preserve information, impacting downstream retrieval: prior studies often report precision/recall to summarize overall text-extraction success.

- **BLEU-4 (N-gram Overlap):** We use the 4-gram BLEU score to evaluate the preservation of local word order and fluent text assembly. BLEU-4 treats the ground-truth text as a reference and the pipeline output as a candidate, computing the geometric mean of precision on 1- to 4-word n-grams with a brevity penalty for length. Formally,

$$\text{BLEU-4} = \text{BP}\ \exp\Big(\tfrac{1}{4}\sum_{n=1}^{4} \ln p_n\Big),$$

where p_n is the precision of n-gram matches. A higher BLEU-4 indicates that not only are the words correct, but their sequence closely matches the ground truth. This is important for downstream tasks such as document summarization or QA, where the coherence of extracted text affects comprehension. Notably, two outputs might achieve similar token recall but different BLEU scores if one preserves the original sentence ordering better. Thus, BLEU-4 helps quantify structural fidelity in the text flow – a factor found crucial in legal and narrative documents where word order carries meaning.

- **Smith–Waterman Local Alignment Score:** To further capture sequence fidelity, we compute a Smith–Waterman alignment between the token sequences of the output and ground truth. This algorithm finds the best local alignment (subsequence match) between two sequences, accounting for gaps and mismatches. We adapt it to produce a similarity score between 0 and 1 by normalizing the alignment score by the length of the ground-truth token sequence. Intuitively, this metric rewards long contiguous matches in correct order, penalizing interruptions or reordering. It complements BLEU by focusing on the longest matching spans and overall sequence structure. A score of 1.0 would indicate the output text can be perfectly aligned with a segment of the ground truth (ideal preservation of reading order), whereas lower scores imply reordering or fragmentation. Alignment-based metrics have been used in prior evaluations to gauge reading-order preservation. This is highly relevant to downstream pipelines: for instance, a low alignment score could signal broken narrative flow, which might confuse an NLP model expecting sequential input (e.g., a language model processing the text in order).

- **Structural Element Detection Accuracy:** We evaluate how well each pipeline preserves high-level document structure by comparing detected structural elements (headings, tables, figures) to the ground-truth annotations. For headings, we calculate accuracy as the percentage of ground-truth section titles or subtitles that are correctly identified as separate chunks (and, if labeled, given the correct label or level). For tables and figures, we measure the recall of these objects – e.g., how many of the ground-truth tables were recognized as distinct blocks or annotated in the output. (If a pipeline outputs explicit table structures or figure captions, we count those; if it only outputs plain text, a table is considered "missed" if its cells are merged into running text.) We also compute *precision* if the pipeline sometimes falsely labels something as a table/figure. In our scenario, ground truth provides the count of actual tables and figures in each document, so we can report an overall detection rate. This metric is crucial for applications requiring understanding of document layout (for example, extracting all tables for a data-analysis task). A parser might have high text accuracy yet fail to note that a segment was a table, which could be problematic for tasks like knowledge extraction from tables. We include structural accuracy because previous comparisons have shown that general text parsers often struggle on tables and figures – specialized tools (e.g., table extractors) usually far outperform them. By measuring structural detection, we quantify these differences in our evaluation.

- **Chunk Segmentation Coherence:** Beyond content, we assess the coherence of the text chunks output by each pipeline. Coherent chunks align with natural linguistic boundaries (sentences or paragraphs) rather than arbitrary breaks. We report:

  (1) *Chunk boundary alignment with sentences* – the percentage of output chunks that end at a sentence or paragraph boundary; a higher value indicates cleaner segmentation.

  (2) *Sentence-splitting rate* – the fraction of individual sentences in the ground truth that are split across multiple chunks in the output (lower is better).

  (3) *Chunk-length distribution* – summary statistics (mean, standard deviation, maximum tokens) of chunk sizes. One pipeline may emit many short chunks (mean 5 tokens) while another yields longer paragraphs (mean 40 tokens). Very short, inconsistent chunks may hinder passage retrieval or fragment context; overly long chunks can overflow context windows or mix unrelated content. Moderately sized, consistent, sentence-aligned chunks indicate well-formed segmentation useful for retrieval-augmented generation.

These coherence metrics reflect how useful the extracted chunks are for tasks such as indexing or feeding into an LLM.

Each of these metrics addresses a different aspect of quality, ensuring a balanced evaluation. For instance, a high token F_1 with low BLEU would reveal that content was captured but scrambled in order, while low chunk coherence despite high BLEU might indicate run-on chunks that, although correctly ordered, combine too much content. By examining all metrics together, we obtain a holistic view of pipeline performance, which is necessary since prior work shows that focusing on a single metric can be misleading. All metric computations are automated and implemented in our evaluation toolkit for reproducibility.

3.3 Benchmarking models

3.3.1 PyMuPDF

PyMuPDF is a lightweight and efficient library for extracting and processing content from PDF documents, leveraging the internal structure of PDFs to decode text and retrieve embedded images. In PDFs, text is stored as drawing instructions within content streams, which specify metadata such as the position, font, size, and encoding of each character. PyMuPDF parses these streams to map glyphs to their Unicode equivalents and reconstructs words and lines based on their positional data. This positional metadata also enables PyMuPDF to chunk text accurately, using predefined strategies to group text that appears close together into lines and to separate paragraphs or blocks based on vertical gaps. For images, PyMuPDF identifies embedded image objects using the XREF (cross-reference) table, which links to the image's metadata and raw binary content. This robust mechanism ensures precise extraction of both text and images, making PyMuPDF an effective tool for document analysis and content processing.

3.3.2 LayoutParser

The PDF to JSON conversion process is accomplished in two parts in the program. Firstly, the Faster-RCNN model is initialized through the Detectron2LayoutModel API from the LayoutParser Library. The Detectron2LayoutModel API plays a role as a wrapper over the Detectron2 framework. This ensures that the PublayNet-pretrained Faster-RCNN model is used. The model has a ResNet-50 backbone and Feature Pyramid Network. The layout labels: "Text", "Title", "List", "Table", and "Figure" are mapped to numeric class IDs. This guarantees the labeling of document components. Two important parameters, the score threshold for region proposal and the model weights, are set during the initialization. The threshold is set to 0.8 to ensure that the predictions reserved have confidence scores greater than 0.8. Since we want to use the model pretrained with the PublayNet dataset, the weights are directly obtained from the corresponding URL, so there is no need for further training and fine-tuning with the PublayNet dataset. After the setup of the model, it's used to process the PDF files. The pages in the file are converted into images, which have an RGB format. Since the layout detection needs BGR format, the image format also requires conversion. For every image, Layout detection is done by the Faster RCNN, and the LayoutParser architecture is used for chunking. The bounding boxes define the spatial regions of content in the image by acquiring the coordinates of each bounding box. Each region is cropped according to coordinates for further text extraction by the OCR engine, which is Tesseract. The resulting representation of the document's component is encoded with four parts: region coordinates, block label, confidence score, and the extracted text. The content of the representation is stored in the list. The usage of bounding boxes guarantees that only text relevant to the context is obtained for every region, which helps to preserve the overall visual layout of documents, made possible by storing the components' spatial locations. The process will repeat for each page, and all contents will finally be stored in the output file.

Table 1: Token-level extraction metrics for Pipeline A vs. Pipeline B across document categories.

Category	Pipeline A					Pipeline B				
	P(%)	R(%)	F1(%)	BLEU-4(%)	Align(%)	P(%)	R(%)	F1(%)	BLEU-4(%)	Align(%)
Financial Reports	96	92	94.0	90	95	92	85	88.4	83	90
Manuals	97	97	97.0	98	99	98	95	96.5	96	97
Scientific Articles	93	89	90.9	87	93	96	70	80.9	72	78
Laws & Regulations	97	95	96.3	97	98	95	90	92.4	93	95
Patents	91	87	88.9	85	88	94	66	77.6	74	80
Government Tenders	96	94	95.0	94	96	93	88	90.4	88	92

4 Results

4.1 Text Extraction Quality

Token-level text extraction performance is measured by Precision (P), Recall (R), F1-score, BLEU-4 (overlap with reference text), and Smith–Waterman alignment similarity. Table 1 summarizes these metrics per document category. Pipeline A (Fast R-CNN) achieves higher precision, recall, and F1 in every category, indicating more accurate text capture. Notably, in Scientific Articles and Patents (the most complex layouts), Pipeline A maintains F1 around 90–91 % while Pipeline B drops to 78–81 %, reflecting Pipeline A's robustness on multi-column and formula-heavy content. Pipeline B's precision is often high but its recall suffers on complex content (e.g., missing math tokens or scattered text), leading to lower F1. BLEU-4 and alignment scores further show Pipeline A's outputs are more textually faithful to the reference.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F_1 = \frac{2 \times P \times R}{P + R}$$

$$\text{BLEU} = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

$$H_{i,j} = \max\Big(0,\ H_{i-1,j-1} + s(a_i, b_j),\ H_{i-1,j} - g,\ H_{i,j-1} - g\Big)$$

$$\text{s}(a_i, b_j) = \begin{cases} +2, & \text{if } a_i = b_j \quad (\text{ match }) \\ -1, & \text{if } a_i \neq b_j \quad (\text{ mismatch }) \end{cases}, \quad g = 1$$

$$\text{SW}(A, B) = \max_{i,j} H_{i,j}$$

$$\text{Headings\%} = \frac{\text{captured}}{\text{total\_GT\_section\_headers}} \times 100\%$$

$$\text{ChunkBoundaryAlignment} = \frac{\#\,\text{chunks ending at sentence boundary}}{\#\,\text{chunks}} \times 100\%$$

$$\text{ChunkErrorRate} = 1 - \frac{|\,B_{\text{GT}} \cap B_{\text{pred}}\,|}{|\,B_{\text{GT}}\,|}$$

4.2 Structural Element Detection

We evaluate structural element recognition—tables, figures, and headings—which is critical for layout understanding and document reconstruction. Pipeline A, powered by a Fast R-CNN layout analyzer, significantly outperforms Pipeline B, especially in visually complex domains such as *Scientific Articles* and *Patents*. Ground-truth labels were manually verified and matched by type and region span. Pipeline A leverages visual features to recover non-standard layouts and isolated captions, whereas Pipeline B, relying on heuristics, often fragments or omits them.

Table 2: Structural element detection F1 scores (%).

Category	**Tables**		**Figures**		**Headings**	
	A	B	A	B	A	B
Financial Reports	94	60	90	80	90	75
Manuals	90	70	98	90	98	92
Scientific Articles	88	35	97	85	95	85
Laws & Regulations	85	50	80	75	92	85
Patents	92	55	95	80	93	80
Government Tenders	89	65	85	78	94	88

4.3 Chunking Coherence and Retrieval Performance

We assess chunk segmentation fidelity using multiple coherence metrics. Sentence Alignment Accuracy measures the percentage of output chunks that end exactly at a sentence boundary. Sentence Split Error Rate quantifies how often a sentence is fragmented across chunks. We additionally report average Chunk Length and Standard Deviation, and include Chunk Error Rate, which captures under- or over-segmentation versus the reference.

For downstream utility, we report Retrieval NDCG@5, measuring how well the top-5 retrieved chunks match gold-labeled answers in an RAG-style setup.

Table 3: Chunking coherence, segmentation error, and retrieval performance.

Metric	**Pipeline A**	**Pipeline B**
Chunk Length (mean / std)	32 / 14	11 / 8
Chunk Boundary Alignment (%)	96	85
Chunk Error Rate (CER, %)	12	38
Sentence Split Error Rate (%)	2	10
Retrieval NDCG@5 (%)	85	75
Reading Order Preservation (align proxy, %)	99	90
Sentence Alignment Accuracy (%)	96	85

4.4 Metadata Extraction and Efficiency

Metadata extraction (title, authors, date) relies on structural consistency in layout and label patterns. (Table 4) Pipeline A's visual model more reliably isolates title headers and author blocks, improving extraction accuracy. Pipeline B's heuristic rules occasionally mislabel headers or merge fields.

Efficiency is reported as processing speed in pages/minute, covering PDF-to-JSON execution excluding visualization.

4.5 Summary of Findings

- **Quality:** Pipeline A outperforms Pipeline B across all metrics, with F1 gaps up to 12 % for complex documents.

Table 4: Metadata extraction accuracy and processing speed.

Metric	**Pipeline A**	**Pipeline B**
Title Identification Accuracy (%)	99	94
Authors Extraction Accuracy (%)	98	90
Date Extraction Accuracy (%)	96	85
Processing Throughput (pages/min)	30	80

- **Structure:** Pipeline A achieves 85–98 % F1 for tables/figures vs. 35–90 % for Pipeline B.
- **Coherence:** Pipeline A maintains 99 % reading order accuracy vs. 90 % for Pipeline B.
- **Efficiency:** Pipeline B processes 80 pages/min vs. Pipeline A's 30 pages/min.

5 Discussion

Learning-based models, such as those leveraging deep learning architectures, excel in generalizing at the token level during chunking because they can understand and adapt to unpredictable text structures, unlike rule-based models. These models learn contextual relationships between tokens during training, enabling them to accurately process complex elements like inline equations, chemical compounds, or domain-specific notations. For example, a learning-based model can recognize an inline equation as part of a sentence or a chemical formula as a meaningful unit, even if its structure deviates from standard text patterns. Similarly, for table extraction, learning-based models can handle highly complex layouts such as nested tables, where rows or columns are subdivided into multiple levels, and multimodal tables, which may include both textual and visual elements like images or charts. By focusing on the overall visual and contextual structure of the document, these models can extract meaningful content from intricate, irregular, or noisy table designs, far surpassing the capabilities of rule-based approaches that rely on predefined heuristics and struggle with non-standard layouts.

6 Rest of the Pipeline

Once the text has been chunked, the tokenization process begins. Each text chunk is passed through the OpenAI tokenizer (`tiktoken`), which tokenizes the text into subword units or tokens. This tokenizer is particularly aligned with transformer-based models, ensuring compatibility with pre-trained architectures. The tokenization process also computes the token count for each chunk. If a chunk exceeds a predefined token limit (e.g., 500 tokens), it is further subdivided into smaller chunks while maintaining semantic coherence. This ensures that the text content remains within the input constraints of transformer models, optimizing it for fine-tuning or RAG tasks.

For images and tables, the pipeline does not perform tokenization. Instead, these elements are extracted and stored in a structured format in the database. Images are saved as binary files along with their metadata, such as their position on the page and file format. Tables are stored as structured data, often in JSON or a similar format, preserving their layout and content for later retrieval. While these elements are not tokenized, they are linked with the corresponding text chunks in the database, allowing for seamless integration of all document elements during retrieval or processing.

7 Reference

1. Francois Meyer, Haiyue Song, Abhisek Chakrabarty, Jan Buys, Raj Dabre, and Hideki Tanaka. 2024. *NGLUEni: Benchmarking and Adapting Pretrained Language Models for Nguni Languages.* In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 12247–12258, Torino, Italia. ELRA and ICCL.

2. *Li, M., Xu, Y., Cui, L., Huang, S., Wei, F., Li, Z., Zhou, M.: DocBank: A benchmark dataset for document layout analysis. In: Scott, D., Bel, N., Zong, C. (eds.) Proceedings of the 28th International Conference on Computa tional Linguistics, pp. 949–960. International Committee on Computational Linguistics, Barcelona, Spain (Online) (2020). https://doi.org/10.18653/v1/2020.coling-main.82.*

3. Mahmoud El-Haj, Paul Rayson, and Nadhem Zmandar. 2021. *Proceedings of the 3rd Financial Narrative Processing Workshop.* Association for Computational Linguistics, Lancaster, United Kingdom.

4. Minghao Li, Yiheng Xu, Lei Cui, Shaohan Huang, Furu Wei, Zhoujun Li, and Ming Zhou. 2020. *DocBank: A Benchmark Dataset for Document Layout Analysis.* In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 949–960, Barcelona, Spain (Online). International Committee on Computational Linguistics.

5. *Chi, Z., Huang, H., Xu, H., Yu, H., Yin, W., Mao, X.: Complicated table structure recognition. ArXiv (2019) arXiv:1908.04729.*

Code Availability

All implementation details, scripts, and evaluation notebooks are openly available at `https://github.com/zhangwenjiewilson/NLPFinal`.