LaTeX Rainbow: Universal LaTeX to PDF Document Semantic & Layout Annotation Framework

Changxu Duan[1], **Zhiyin Tan**[2], **Sabine Bartsch**[3]
Technische Universität Darmstadt, Germany[1,3]
[1]changxu.duan@tu-darmstadt.de [2]zhiyin.tanh@gmail.com
[3]sabine.bartsch@tu-darmstadt.de

Abstract

Machine Learning models in the field of Information Extraction for Scientific Publications require high-quality labeled data. The large amount of easily accessible LaTeX source code is a treasure trove of high-quality labeled data. However, existing datasets comprised of document collections and PDF extraction tools have limitations: (1) The hierarchical structure of papers is lost because labeling is done in terms of pages rather than documents; (2) The reading order is not extracted, which potentially muddles the extracted contextual structure; (3) Papers included in the datasets are not likely to be up-to-date.To address these challenges, we propose LaTeX Rainbow, a framework that bridges LaTeX to PDF that can automatically annotate and extract semantic and layout information from LaTeX source code. This framework extends existing annotation methods by taking into account the properties of different existing approaches. It can produce token-level semantic structure annotations, preserve the paper's reading order, and extract the table of contents, i.e., the article's section structure. LaTeX Rainbow enables anyone to extend their datasets with the latest documents. The project is open-sourced on GitHub[1] for community contributions and use.

1 Introduction

Scientific publications are often delivered in a form that is unstructured from the perspective of the underlying data, notably layout-focused formats such as PDFs.These formats, while visually appealing and optimized for human comprehension, present significant challenges when it comes to automatic Information Extraction (IE). For example, it is difficult for PDF extraction software to distinguish which part of a PDF page constitute the actual contents of the paper as opposed to other elements such as headers, metadata, author and affiliation etc. and multimodal contents such as images, tables, equations etc. and their captions (Meuschke et al., 2023; Bast and Korzen, 2017). Additionally, these documents often contain elements that are not directly related to the core content, such as watermarks (Chia et al., 2018), publisher details and header information that serves navigation in collections. These elements, often introduced by the publishing process, further complicate the extraction process as they are not semantically linked to the main content. They appear within the layout of the page, but are hard to distinguish from the paper contents.

In order to solve these problems, there has been a surge in the development of document understanding machine learning models over the past few years (Cui, 2021; Subramani et al., 2021; Han et al., 2023). These models are designed to delve deep into documents, extracting semantic information by harnessing both their visual and textual attributes. However, machine learning, being a data-driven approach, requires extensive labeled data. Considering that existing PDF extraction tools cannot guarantee the accuracy of the extraction (Meuschke et al., 2023). In this context, LaTeX code has emerged as a valuable resource. Many of the weakly supervised annotated document IE datasets have their genesis in LaTeX code (Li et al., 2020b; Schmitt-Koopmann et al., 2022; Anitei et al., 2023).

LaTeX is a typesetting system commonly used for scientific publication, and LaTeX code can be easily compiled into PDF format. The explicit markup in LaTeX code describes the structure and formatting of the document.Given that scientific publications inherently maintain a hierarchical and semantic structure, such as sections, subsections, figures, tables, equations, etc. These elements are all clearly defined within LaTeX commands. This clarity facilitates automatic annotation systems in identifying and categorizing document elements, as they are al-

[1]https://github.com/InsightsNet/texannotate

Proceedings of the 2nd Workshop on Information Extraction from Scientific Publications, pages 56–67
Nov 1, 2023.

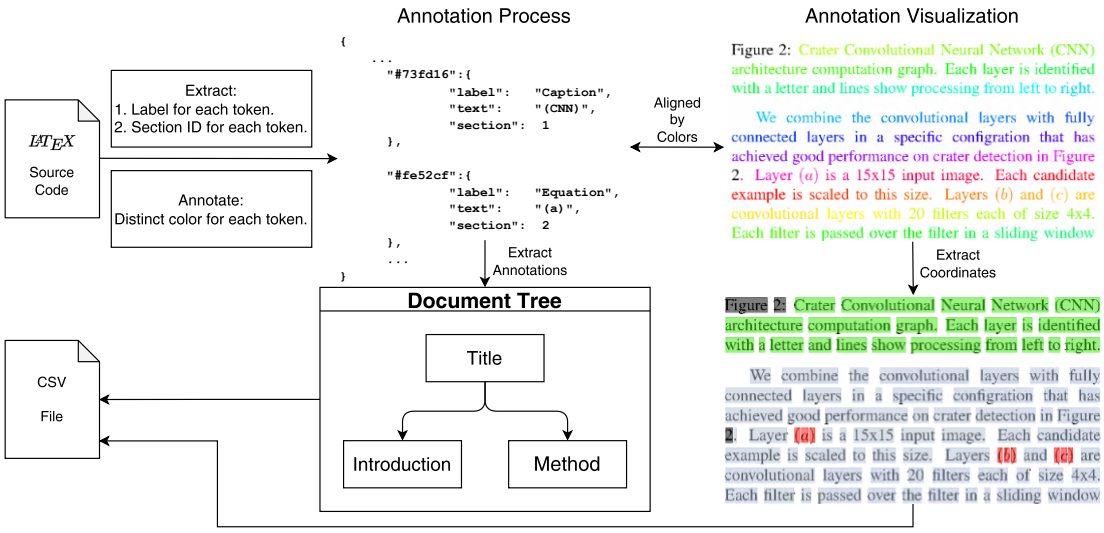

Figure 1: Process of document annotation and extraction. In this figure example, a paper (Cohen et al., 2016) from arXiv is annotated by color and have been extracted with the semantic layout label for each token.

ready explicitly delineated and classified by the author's markup (Ogawa, 1994). The author's intent can be inferred more effectively from the structural and semantic cues within LaTeX code, leading to a more accurate and context-aware interpretation of the document. With the increasing release of scientific publications in LaTeX source code, particularly on arXiv[2], there has been a surge in the number of PDF Information Extraction datasets derived from LaTeX. These datasets predominantly utilize LaTeX coloring features, namely colored fonts or drawing of colored boxes within the PDF. However, there are notable shortcomings in the current PDF Information Extraction datasets. One of the key issues is that current annotations are often made on a per page basis, and no popular datasets explicitly annotate whether an element spanning two pages belongs to the same entity, e.g. a paragraph or itemized list. As a result, an element spanning across multiple pages might be interpreted as two distinct entities instead of one continuous element. This discrepancy also affects the hierarchical structure of scientific publications. Consider that document digitization standards, including Journal Article Tag Suite (JATS)[3] and Text Encoding Initiative (TEI)[4] provide definitions of section trees, which are beneficial to IE (Kikuchi et al., 2014; Hu et al., 2022; Landolsi et al., 2023). It would be a shame to lose the hierarchy.

Additionally, recent models and datasets derived from LaTeX often omit reading order details (Li et al., 2020b; Blecher et al., 2023). LaTeX is a complex ecosystem with a vast collection of packages filled with numerous command definitions via Comprehensive TeX Archive Network (CTAN). Various templates each have their own unique writing conventions. This leads to the possibility that some elements may be mislabeled. Given that different publications adopt varied page layouts and LaTeX autonomously determines the positioning of tables and figures based on its internal rules (Mittelbach et al., 2004), there's a significant risk that automated information extraction tools might misinterpret the intended reading order and context.

Moreover, authors of such resources do not always publish the code used in compiling the dataset. This means that current datasets are unlikely to incorporate the most recent papers or newer version of old papers. This hampers reproducibility of the process of dataset building as well as impeding scalability. Meanwhile, it is difficult for users to modify the annotation style to match their demands.

In this paper, we build upon several approaches for automatic PDF annotation of datasets and introduce a generalized framework that yields document-oriented, fine-grained, reading-ordered annotations that exclude extraneous content based on LaTeX source code. Figure 1 is a simplified representation of the labeling process. Our framework improves the accuracy and robustness of IE for scientific publications that has LaTeX code. Further-

[2] https://info.arxiv.org/help/submit/index.html

[3] https://jats.nlm.nih.gov/articleauthoring/tag-library/1.3/element/sec.html

[4] https://tei-c.org/release/doc/tei-p5-doc/en/html/ref-tree.html

more, we believe the new annotated data comes from our framework could drive more accurate IE machine learning models for PDF only papers, i.e. scanned paper. Our contributions are:

1. Our framework refine the categories, per document based reading order and hierarchy through a well-designed coloring strategy.

2. Enhancing code parsing capabilities by invoking the parsing databases of modern LaTeX integrated writing environment.

3. Providing CSV tables output per PDF document, easily modifiable to meet user needs.

This framework is free software and available under the Apache 2.0 license.

2 Related Work

2.1 PDF Information Extraction Softwares

Currently there are many software solutions or services that provide PDF content extraction functions. Adobe Extract[5] and Apache Tika[6] provide API service to extract texts from PDF, but they do not provide fine-grained labeling. Camelot[7] and Tabula[8] focus on table extraction. RefExtract[9] specializes in extracting references. CERMINE (Tkaczyk et al., 2015), GROBID [10] PdfAct[11] and Science Parse[12] support the identification of more categories. PyMuPDF[13] allows access to information about the more underlying details of the PDF file. However, a benchmark demonstrates their imperfect performance (Meuschke et al., 2023).

2.2 Document Datasets

Many of the datasets' annotation were taken from LaTeX. TableBank (Li et al., 2020a) specialize in table extraction. DocBank (Li et al., 2020b) extended from TableBank, provides token-level fine-grained categories labeling. FormulaNet (Schmitt-Koopmann et al., 2022) and IBEM (Anitei et al., 2023) focus on mathematical formulas, especially in-line formulas, which can easily be confused with plain-texts. SciBank (Grijalva et al., 2022) produces block-level annotations.

There are also many datasets from non-LaTeX sources. PubLayNet (Zhong et al., 2019) and DocLayNet (Pfitzmann et al., 2022) obtained particularly large amounts of labeling using automated and manual methods, respectively. XFUND (Xu et al., 2022) manually labeled multilingual tabular data. ReadingBank (Wang et al., 2021) is extracted from Microsoft Word documents, which standardize the reading order of blocks within a page. M^{6}Doc (Cheng et al., 2023) extracted large-scale data using a half machine learning, half manual approach.

2.3 Document Understanding Models

With the gradual enrichment of document data IE resources, machine learning model development is driven by increasingly larger datasets using different approaches. LayoutLMs (Xu et al., 2020, 2021a,b; Huang et al., 2022) and its variants (Shen et al., 2022), make it possible to analyze document layout from the 2D coordinates of texts plus visual features. Donut (Kim et al., 2022), on the other hand, changes the structure of model to a separate visual encoder and language model decoder without obtaining texts directly from the document. Nougat (Blecher et al., 2023) follows Donut in implementing PDF to markup language conversion. However, Nougat's approach is page-based, and cross-page paragraphs may be incorrectly sliced by figures or tables on a page. With the explosion of Large Language Models (LLM), a number of online document understanding systems have sprung up, such as Explainpaper[14] and OpenRead[15], but these platforms are commercial and closed-source. In this paper we present a framework that is freely available and can support the community to enhance the open-source approach.

3 Methodology

We present a five-step approach to annotate the LaTeX code to PDF annotation. (1) Initially, the PDF file is processed the existing font colors and shapes in the file are captured. (2) Subsequently, the corresponding LaTeX source code of the PDF is parsed. Each token within the file is assigned a distinct color. Furthermore, figures within the document are enclosed within borderless rectangles and are highlighted with unique background colors.

[5] https://www.adobe.io/apis/documentcloud/dcsdk/pdf-extract.html
[6] https://tika.apache.org/
[7] https://github.com/camelot-dev/camelot
[8] https://github.com/chezou/tabula-py
[9] https://github.com/inspirehep/refextract
[10] https://github.com/kermitt2/grobid
[11] https://github.com/ad-freiburg/pdfact
[12] https://github.com/allenai/science-parse
[13] https://github.com/pymupdf/PyMuPDF
[14] https://www.explainpaper.com/
[15] https://www.openread.academy/

These specific colors act as pointers to the respective segments of the LaTeX source code, ensuring traceability. (3) Upon completing the LaTeX parsing, the source code is compiled into a color-coded PDF. (4) The framework then aligns text and figures from the PDF document with their respective segments in the LaTeX source code by color. This alignment facilitates the extraction of semantic annotations and coordinates of each token and the establishment of a hierarchical structure throughout the document. (5) Finally, we packaged all the annotation as CSV files.

In order to complete these steps, the framework needs three functions: PDF Element and Color Extraction, Color Generation and Annotation, and PDF Compilation. The subsequent sections provide the implementation of each function in detail.

3.1 Element and Color Extraction from PDF

In this function we use an off-the-shelf Python package `pdfplumber`[16] to read the details inside the PDF file. This tool is proficient in pinpointing the position, font, and color of every character on a page. Additionally, it can determine the position and color attributes of all rectangles on the page, which encompasses both border and fill colors. By default, `pdfplumber` utilizes DeviceRGB color space, extracting colors as tuples of three floating-point numbers. For example, the color black is represented as (0.0, 0.0, 0.0) while red is (1.0, 0.0, 0.0). However, modern computer languages, sometimes struggle with accurately storing and accessing floating-point numbers. This inherent inaccuracy implies that color matching based on these numbers might be prone to errors, stemming from cumulative inaccuracies.

In our framework, colors for fonts are represented as tuples of 8-bit values, namely red is represented as (255, 0, 0) or #ff00000 in hexadecimal. When `pdfplumber` extracts colors from PDF document, each tuple element value is incremented in steps of 0.00392, for instance, 8-bit (0, 1, 2) translates to floating-point (0.0, 0.00392, 0.00784). Given that $\frac{1}{255} = 0.00392156862$, we are already dealing with a discrepancy. To mitigate this threat, we employed matplotlib's `to_hex()`[17] method to ensure precise floating-point to 8-bit RGB value matches. We also provide details in the selection of tools for extracting color in Appendix A.

[16] https://github.com/jsvine/pdfplumber

[17] https://matplotlib.org/stable/api/_as_gen/matplotlib.colors.to_hex.html

3.2 Colors Generation and Annotation

In this function, we process the LaTeX source code to determine color assignments for each element. Ideally, each element should have a unique color. A straightforward approach would be to incrementally assign hexadecimal numbers from #000000 to #ffffff. However in practice we have found that such an increment leads to a very insignificant color change. For instance, the token gradually changes from black #000000 to a blue shade almost indistinguishable from black #000001, then #000002, until it reaches full blue #0000ff. This makes neighboring tokens almost the same color, and distinguishing between them can be challenging for both computer displays and humans eyes.

To enhance visibility and facilitate manual error-checking, we adopted a hue-based color generation strategy. More specifically, we use the Hue, Saturation, Value (HSV) color space to cyclically extract colors and rearrange them into appropriate groups. Each HSV color is then converted to an RGB tuple. Finally, around 9 million colors were grouped and sorted to be used in the next step of color marking.

For every recognized token, it gets substituted with:

```
{\color[RGB]{0,0,1}<TOKEN>}
```

Where `<token>` is the token to be colored. Each identified figure is replaced with:

```
\colorbox[rgb]{0,0,1}{<FIGURE>}
```

Where `<FIGURE>` is the command of including the figure file, or the block of drawing a figure. Distinct colors are allocated to each segment.

We also insert rules that ensure the required packages are imported and any rectangle placed beneath an figure does not disrupt the document's original layout.

```
\usepackage{xcolor}
\usepackage{tcolorbox}
\setlength { \fboxsep }{ 0pt }
\setlength { \fboxrule }{ 0pt }
```

Next we need to parse LaTeX source code and apply the above annotation rules to them.

3.2.1 LaTeX Parsing and Annotation

Broadly, elements in LaTeX source code comprises four classes: body text, macro, environment, and comments. We have the following parsing strategies for each of these elements.

- **Body text** segment undergoes tokenization using spaCy[18] tokenizer in order to split punctuation correctly. We track of the number of space characters after each token to maintain the integrity the original PDF page layout. Each token is color-marked and recorded.

- **Macro** begins with a backslash, and its arguments are encompassed within braces. We focus on labeling certain macros such as `\title{}` and `\author{}`, attributing their parameter literals with relevant semantic structure labels. Arguments within the curly braces will be parsed as body text if it will appear in the compiled PDF. Notably, `\includegraphics{}` will be treated as a whole and marked with a colorful borderless rectangle, as it is an inserted figure without fonts. `\input{}` and `\include{}` will point to another source code file, and we recursively parse the contents of the file.

- **Environment** consists of entities encapsulated between start and end commands. Elements within a environment are recursively parsed as macro or body text. For example, `\begin{table}...\end{table}` is a table environment and the elements within this table element such as `\caption{}` will be parsed as macro. Specifically, drawing environment `\begin{tikzpicture}...\end{tikzpicture}` will be treated as a whole block and marked with a colorful borderless rectangle. Note that only elements within the `document` environment are colored.

- **Comment** element starts with a percent sign % and continues until the end of this line in the LaTeX source code. We ignore the annotation of comment as it does not affect any part of the compiled PDF.

We employ the Python package `pylatexenc`[19] to traverse and parse the LaTeX source code, character by character. `pylatexenc` contains a collection of commands created by the contributors, which defines: the name of commands; whether the command has a variant or not e.g. `\section{}` and `\section*{}`; and the number of command arguments, including optional arguments in square brackets and required arguments in curly braces. In practice, we found these predefined rules insufficient, prompting us to manually augment the definition file.

However, as the number of parsed source codes increases, the trend of newly encountered undefined commands does not stop. We introduced the parsing database from `LaTeX Workshop`[20] and `TeXstudio`[21] in order to extend our database of parsing rules once and for all. `LaTeX Workshop` is an extension for a popular code editor Visual Studio Code[22], aiming to provide all-in-one features and utilities for LaTeX typesetting. `TeXstudio` is an integrated writing environment for creating LaTeX documents. They are featured by a particularly complete database of automatically generated commands from CTAN. We implemented the method to download all the definitions from the repository in JSON format. They are stored on a package-by-package basis, i.e., each package that can be referenced by LaTeX with `\usepackage{}` has a corresponding JSON file that contains all the commands for the package, including macros and environments. Our parser first traverses through LaTeX source code, collects all package loads, and then reads the relevant JSON entries. These commands are subsequently transformed and integrated into the parsing rules for `pylatexenc`.

Next, we allocate each color the semantic layout label to which it belongs. The set of labels aligns with those used in GROBID and DocBank:

- **Abstract** is assigned to body texts within `abstract` environment, or argument of macro `\abstract{}` and its variants.

- **Author** is assigned to argument of macro `\author[]{}`, `\address{}` and their variants.

- **Caption** is assigned to macro `\caption{}` within table or figure environment.

- **Equation** is assigned to all the element that marked with mathematical mode by `pylatexenc`.

- **Figure** is assigned to drawings or imported figures.

- **List** is assigned to body texts within `itemize` or `enumerate` environment.

[18] https://spacy.io/
[19] https://github.com/phfaist/pylatexenc
[20] https://github.com/James-Yu/LaTeX-Workshop
[21] https://www.texstudio.org/
[22] https://code.visualstudio.com/

- **Paragraph** is assigned to body texts within `document` environment.
- **Reference** is assigned to body texts within `bibliography` environment, or macro `\bibliography{}`.
- **Section** is assigned to macros that indicate a new section in command.
- **Table** is assigned to environments that include tabular in command.
- **Title** is assigned to argument of macro `\title[]{}` and its variants.

With the annotations in place, the LaTeX source code is ready for compilation. Following this, `pdfplumber` could determine the color and position of each letter, utilizing the letter's color to match the corresponding annotation.

Simultaneously, the article's hierarchical structure is encapsulated within a tree data structure. LaTeX delineates a hierarchy spanning seven levels through its macro command[23]. We record for each colored element the node of the tree it belongs to. To ensure a coherent hierarchy, we additionally define the paper title as the root level of document. That is, all tokens in the title of a scientific paper belong to Title node, while all tokens in the Introduction section belong to the Introduction node, and Introduction node is a child of Title node in the tree of this document. In addition we discuss the argument of hierarchical structure in Appendix B.

3.3 PDF Compilation

To initiate PDF Compilation process, two specific lines of code are added to the beginning of the source file: `\pdfoutput=1` instructs the compiler to produce a PDF instead of PostScript, an alternative publication format; `\interactionmode=1` signals the compiler to persist with the output generation, even if it encounters an error on a page.

Publications that accept submissions in LaTeX format, including arXiv, often recommend using `pdftex` as their preferred rendering engine. This engine is integrated into the contemporary LaTeX distribution, TeXLive. Given the complexities in setting up this distribution, we opted for the Docker image[24] of TeXLive 2023 to establish our compilation environment.

Automated LaTeX compilation presents a challenge, especially in pinpointing the master source file. This is because LaTeX allows for multiple `.tex` source files to be consolidated and compiled into one overarching master PDF. To navigate this challenge, we integrated arXiv's AutoTeX[25] automatic compilation system. AutoTeX, a Perl-based toolkit, excels at discerning the primary source file within a project. Our PDF compilation mechanism derives some of its functionalities from an open-source AutoTeX wrapper[26].

However, during our practice, we observed AutoTeX's compilation regulations as overly stringent. There were instances when it halted the compilation due to minor errors, even when the same content had successfully passed arXiv's publication standards. Specifically, AutoTeX will stop compiling and try other compilers immediately after the compiler returns an error code, even if the ignored error signal placed in front of the compiler to successfully output the compiled PDF file. The PDF file produced in this step is wrong because of the BibTeX mechanism. BibTeX is reference management software for formatting lists of references, which needs to be run twice by the compiler in order to correctly output references, and in-text citations. To address this, we modified AutoTeX's code[27], enabling it to bypass certain errors and persist with PDF generation. We configured AutoTeX to control the compiler to run at least twice, even if an error code is returned initially.

In conclusion, we combine TeXLive 2023, AutoTeX, and a Python-based API service into a single container. This container, accessible via HTTP, accepts source code and efficiently returns the compilation outcomes.

4 Capabilities Overview

During the export phase, outputs are organized into two DataFrames:

- The first DataFrame represents the Table of Contents nodes, which was created during the annotation process. Each line denotes a tree node, with every node possessing a unique ID and an ID indicating its parent. This structured approach ensures that the succeeding page

[23] https://www.overleaf.com/learn/latex/Sections_and_chapters

[24] https://hub.docker.com/r/texlive/texlive

[25] https://metacpan.org/pod/TeX::AutoTeX

[26] https://github.com/andrewhead/texcompile

[27] The modified AutoTeX can be found at https://github.com/Fireblossom/TeX-AutoTeX-Mod

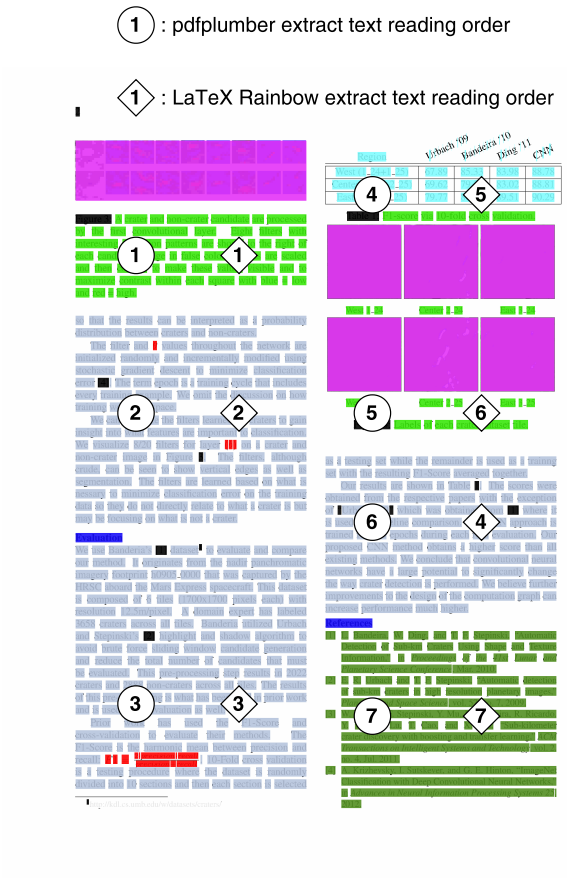

Figure 2: Extracting reading order from an elementally complex page. This example is taken from the second page of (Cohen et al., 2016).

elements to be exported can be systematically assigned to their respective nodes.

- Every row in the second DataFrame denotes either a figure or token extracted from the PDF. These tokens are allocated a number indicating their reading order and a section ID, if they are part of the author's main content, starting from 0. A value of -1 in reading order indicates elements not penned by the author, and auto-generated by the LaTeX template. This facilitates a straightforward exclusion of such elements during further analysis. The label column encompasses semantic structure labels including: Abstract, Author, Caption, Equation, Figure, Footer, List, Paragraph, Reference, Section, Table, and Title.

Both DataFrames can be exported as CSV files.

4.1 Use Case: Text Extraction with Reading Orders

Reading order refers to the sequence in which content is meant to be read. In documents, it usually follows a top-to-bottom, left-to-right pattern, but there can be exceptions, especially in multilingual or complex layout documents.

Figure 2 illustrates such a scenario. The labels in the figure represent the order of the text extracted by the different methods. In this case, starting from the sixth block, the paragraphs extracted using `pdfplumber` are interrupted by tables and pictures.

4.2 Use Case: Section-Weighted Scientific Paper Summarization using LLM

Scientific papers are structured documents with different sections, each serving a distinct purpose (Wcg, 2008). While traditional summarization techniques consider the entire document holistically (Ibrahim Altmami and El Bachir Menai, 2022), section-weighted summarization assigns different weights to different sections, recognizing that some parts of a paper may be more informative or critical than others for a quick understanding (Cohan et al., 2018). The advantage of our framework is the ability to accurately obtain sections as compared to any current PDF extraction tool. We provide a simple example in our GitHub repository.

5 Validation of Annotations

In addition to the unique features introduced in Section 4, we also have to verify its consistency with the annotation methods for existing datasets. We use the DocBank dataset (Li et al., 2020b) to assess the reliability of annotations generated by LaTeX Rainbow framework. Docbank dataset contains annotations for 1.5 million content elements across 500K scientific publication pages. It comprises papers from arXiv published between 2014 and 2018, spanning fields like physics, mathematics, and computer science. Due to its extensive size, range of subjects, numerous annotated elements, and labeling method, Docbank is considered a benchmark dataset for LaTeX sources.

Since DocBank is a very large dataset, we extracted a subset for time and feasibility reasons. We extracted LaTeX source code of 100 papers in DocBank from arXiv. They are then annotated and compiled by our framework, 61 papers are successfully annotated and compiled. 39 papers raise errors. We summarize the reasons and numbers for failures:

1. Parsing errors in the source code, such as unmatched bracket pairs or expressions that are digestible by pdfTeX but not by `pylatexenc`,

Labels	Precision	Recall	F_1-Score
Abstract	0.9779	0.8197	0.8918
Author	0.5027	0.5515	0.5260
Caption	0.4676	0.3851	0.4224
Equation	0.1957	0.9016	0.3216
Footer	0.2029	0.2612	0.2284
List	0.4462	0.1762	0.2526
Paragraph	0.9379	0.6080	0.7377
Reference	0.8355	0.9718	0.8985
Section	0.5119	0.3777	0.4347
Table	0.8806	0.7939	0.8350
Title	0.3429	0.5320	0.4170

Table 1: Precision, Recall, and F_1-Score of annotations from LaTeX Rainbow, compare to DocBank.

cause the parser to misread and yield empty results. (23)

2. Compilation failures. It was reported[28] that some source code could not be compiled successfully due to change of compilation environment. (15)

3. File encoding problem. The source code contains characters that Python cannot handle. (1)

To gauge consistency across annotated datasets, Cohen's Kappa Coefficient $\kappa = \frac{p_o - p_e}{1 - p_e}$ is applied to test consistency across annotated datasets. where p_o is the empirical probability of agreement on the label assigned to any sample, and p_e is the expected agreement when both annotators assign labels randomly. p_e is estimated using a per-annotator empirical prior over the class labels (Artstein and Poesio, 2008). Cohen suggested the Kappa result be interpreted as follows: values ≤ 0 as indicating no agreement and 0.01–0.20 as none to slight, 0.21–0.40 as fair, 0.41– 0.60 as moderate, 0.61–0.80 as substantial, and 0.81–1.00 as almost perfect agreement (McHugh, 2012). We get $\kappa = 0.32$. This value demonstrates the fair consistency of our approach with the existing baseline .

We further assessed Precision, Recall, and F_1-Score using DocBank's annotations as the gold standard. As detailed in Table 1, our framework's annotations align closely with DocBank in the Abstract, Paragraph, Reference, and Table categories, and differs in other categories. Notably, there is a considerable degree of inconsistency in the semantic labeling of some categories. The reasons for these inconsistencies are mainly differences in annotation strategies and difficulties in aligning our annotations with DocBank. We delve deeper into the inconsistencies in Appendix C.

6 Known Issues and Future Work

In the future, our primary goal is to update the parser so that it can tolerate syntax errors that the pdfTeX compiler can tolerate.

We also plan to expand the LaTeX Rainbow framework with parallelization capabilities. Because the pdfTeX engine does not support multithreaded parallelism, this makes them slow to compile, especially for long files. The idea is to enable parallelism at the scale of multiple files. Given the containerized nature of our PDF compilation service, this transition should be seamless.

For extensive projects, consider the example of *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, which comprises 912 papers. The cumulative number of tokens in such projects might surpass the maximum number of colors that can be allocated (9 million). In these cases, the LaTeX Rainbow framework is unable to perform the annotation. Our next steps include refining the color system to enhance its usability and distinctiveness.

Coloring the LaTeX element doesn't always work. For example, `\url{}` command forces its arguments to be blue, instead of the color we assigned. In addition to the coloring method, there is also the SyncTeX (Laurens, 2008) plugin that allows compiled PDF elements to be linked back to the LaTeX source code. It is directly involved in the compilation process and records the correspondence through the internal auxiliary files. Our plan is to go deeper into its mechanisms to establish a more robust PDF to LaTeX source code correspondence.

Different publisher templates interpret LaTeX terms uniquely, making it challenging for our database to account for every variation. For example, `\lstinputlisting[]{}` defines the content to be displayed to the PDF with a substring value in the optional argument. This is significantly different from the definition of most commands. As mentioned in Section 5, there are many papers that cannot yet be parsed correctly. Therefore, we greatly welcome and depend on the open-source community to contribute the detailed parsing rules for each template.

[28] https://info.arxiv.org/help/faq/texlive.html

7 Conclusion

In this paper we introduce a framework which can be used to establish a correspondence between LaTeX code and PDF elements, exporting detailed semantic annotations. Our framework meticulously extracts semantic markup, maintaining the layout fidelity of the associated PDF files. The structured information extracted by our framework helps in better document indexing, searching and analysis. It improves document accessibility and helps develop and refine document understanding tools.

Our framework is more than just yet another toolkit to the growing list of document datasets. By ensuring versatility and adaptability as well as scalability, we aim for it to become a universal tool that can facilitate enhanced document analysis across multiple disciplines and applications. We sincerely hope that open-source community can derive innovative uses and benefits from our solution.

8 Acknowledgments

The framework reported is this paper was conducted within the research project InsightsNet (https://insightsnet.org/) which is funded by the Federal Ministry of Education and Research (BMBF) under grant no. 01UG2130A.

References

Dan Anitei, Joan Andreu Sánchez, José Miguel Benedí, and Ernesto Noya. 2023. The ibem dataset: A large printed scientific image dataset for indexing and searching mathematical expressions. *Pattern Recognition Letters*, 172:29–36.

Ron Artstein and Massimo Poesio. 2008. Inter-coder agreement for computational linguistics. *Computational linguistics*, 34(4):555–596.

Hannah Bast and Claudius Korzen. 2017. A benchmark and evaluation for text extraction from pdf. In *Proceedings of the 17th ACM/IEEE Joint Conference on Digital Libraries*, JCDL '17, page 99–108. IEEE Press.

Lukas Blecher, Guillem Cucurull, Thomas Scialom, and Robert Stojnic. 2023. Nougat: Neural optical understanding for academic documents.

Hiuyi Cheng, Peirong Zhang, Sihang Wu, Jiaxin Zhang, Qiyuan Zhu, Zecheng Xie, Jing Li, Kai Ding, and Lianwen Jin. 2023. M6doc: A large-scale multi-format, multi-type, multi-layout, multi-language, multi-annotation category dataset for modern document layout analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15138–15147.

Wai Chong Chia, Phoey Lee Teh, and Colin Mathew Hew D Gill. 2018. Text extraction and categorization from watermark scientific document in bulk. In *2018 3rd International Conference on Computational Intelligence and Applications (ICCIA)*, pages 47–51.

Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. 2018. A discourse-aware attention model for abstractive summarization of long documents. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 615–621, New Orleans, Louisiana. Association for Computational Linguistics.

Joseph Paul Cohen, Henry Z Lo, Tingting Lu, and Wei Ding. 2016. Crater detection via convolutional neural networks. *47th Lunar and Planetary Science Conference*.

Lei Cui. 2021. *Document AI: Benchmarks, Models and Applications (Presentation@ICDAR 2021)*. DIL workshop in ICDAR 2021.

Felipe Grijalva, Carla Parra, Marco Gallardo, Erick Santos, Byron Acuña, Juan Carlos Rodríguez, and Julio Larco. 2022. Scibank: A large dataset of annotated scientific paper regions for document layout analysis.

Xue Han, Yi-Tong Wang, Jun-Lan Feng, Chao Deng, Zhan-Heng Chen, Yu-An Huang, Hui Su, Lun Hu, and Peng-Wei Hu. 2023. A survey of transformer-based multimodal pre-trained modals. *Neurocomputing*, 515:89–106.

Faegheh Hasibi and Svein Erik Bratsberg. 2014. Non-hierarchical structures: How to model and index overlaps?

P. Hu, Z. Zhang, J. Zhang, J. Du, and J. Wu. 2022. Multimodal tree decoder for table of contents extraction in document images. In *2022 26th International Conference on Pattern Recognition (ICPR)*, pages 1756–1762, Los Alamitos, CA, USA. IEEE Computer Society.

Yupan Huang, Tengchao Lv, Lei Cui, Yutong Lu, and Furu Wei. 2022. Layoutlmv3: Pre-training for document ai with unified text and image masking. In *Proceedings of the 30th ACM International Conference on Multimedia*, MM '22, page 4083–4091, New York, NY, USA. Association for Computing Machinery.

Nouf Ibrahim Altmami and Mohamed El Bachir Menai. 2022. Automatic summarization of scientific articles: A survey. *Journal of King Saud University - Computer and Information Sciences*, 34(4):1011–1028.

Yuta Kikuchi, Tsutomu Hirao, Hiroya Takamura, Manabu Okumura, and Masaaki Nagata. 2014. Single document summarization based on nested tree structure. In *Proceedings of the 52nd Annual Meeting of*

the Association for Computational Linguistics (Volume 2: Short Papers), pages 315–320, Baltimore, Maryland. Association for Computational Linguistics.

Geewook Kim, Teakgyu Hong, Moonbin Yim, JeongYeon Nam, Jinyoung Park, Jinyeong Yim, Wonseok Hwang, Sangdoo Yun, Dongyoon Han, and Seunghyun Park. 2022. Ocr-free document understanding transformer. In *European Conference on Computer Vision (ECCV)*.

Mohamed Yassine Landolsi, Lobna Hlaoua, and Lotfi Ben Romdhane. 2023. Hybrid method to automatically extract medical document tree structure. *Engineering Applications of Artificial Intelligence*, 120:105922.

Jérôme Laurens. 2008. Direct and reverse synchronization with SyncTEX. 29(3):365–371.

Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, Ming Zhou, and Zhoujun Li. 2020a. TableBank: Table benchmark for image-based table detection and recognition. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 1918–1925, Marseille, France. European Language Resources Association.

Minghao Li, Yiheng Xu, Lei Cui, Shaohan Huang, Furu Wei, Zhoujun Li, and Ming Zhou. 2020b. 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.

Yves Marcoux, Michael Sperberg-McQueen, and Claus Huitfeldt. 2013. Modeling overlapping structures. *Balisage Series on Markup Technologies*.

Mary L McHugh. 2012. Interrater reliability: the kappa statistic. *Biochemia medica*, 22(3):276–282.

Norman Meuschke, Apurva Jagdale, Timo Spinde, Jelena Mitrović, and Bela Gipp. 2023. A Benchmark of PDF Information Extraction Tools Using a Multi-task and Multi-domain Evaluation Framework for Academic Documents. In *Information for a Better World: Normality, Virtuality, Physicality, Inclusivity*, volume 13972 of *LNCS*, pages 383–405. Springer Nature Switzerland, Cham.

Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, and Chris Rowley. 2004. *The LaTeX Companion*, 2 edition. Addison-Wesley. ISBN-10: 0201362996 ISBN-13: 978-0201362992.

Arthur Ogawa. 1994. Object-oriented programming, descriptive markup, and tex. *TUGboat*, 15(3):325–330.

Birgit Pfitzmann, Christoph Auer, Michele Dolfi, Ahmed S. Nassar, and Peter Staar. 2022. Doclaynet: A large human-annotated dataset for document-layout segmentation. page 3743–3751.

Felix M. Schmitt-Koopmann, Elaine M. Huang, Hans-Peter Hutter, Thilo Stadelmann, and Alireza Darvishy. 2022. Formulanet: A benchmark dataset for mathematical formula detection. *IEEE Access*, 10:91588–91596.

Zejiang Shen, Kyle Lo, Lucy Lu Wang, Bailey Kuehl, Daniel S. Weld, and Doug Downey. 2022. VILA: Improving structured content extraction from scientific PDFs using visual layout groups. *Transactions of the Association for Computational Linguistics*, 10:376–392.

Nishant Subramani, Alexandre Matton, Malcolm Greaves, and Adrian Lam. 2021. A survey of deep learning approaches for ocr and document understanding.

Dominika Tkaczyk, Paweł Szostek, Mateusz Fedoryszak, Piotr Jan Dendek, and Łukasz Bolikowski. 2015. Cermine: automatic extraction of structured metadata from scientific literature. *International Journal on Document Analysis and Recognition (IJDAR)*, 18:317–335.

Zilong Wang, Yiheng Xu, Lei Cui, Jingbo Shang, and Furu Wei. 2021. LayoutReader: Pre-training of text and layout for reading order detection. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4735–4744, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Peh Wcg. 2008. Basic structure and types of scientific papers. *Singapore Med J*, 49(7).

Yang Xu, Yiheng Xu, Tengchao Lv, Lei Cui, Furu Wei, Guoxin Wang, Yijuan Lu, Dinei Florencio, Cha Zhang, Wanxiang Che, Min Zhang, and Lidong Zhou. 2021a. LayoutLMv2: Multi-modal pre-training for visually-rich document understanding. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2579–2591, Online. Association for Computational Linguistics.

Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, and Ming Zhou. 2020. Layoutlm: Pre-training of text and layout for document image understanding. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, KDD '20, page 1192–1200, New York, NY, USA. Association for Computing Machinery.

Yiheng Xu, Tengchao Lv, Lei Cui, Guoxin Wang, Yijuan Lu, Dinei Florencio, Cha Zhang, and Furu Wei. 2021b. Layoutxlm: Multimodal pre-training for multilingual visually-rich document understanding.

Yiheng Xu, Tengchao Lv, Lei Cui, Guoxin Wang, Yijuan Lu, Dinei Florencio, Cha Zhang, and Furu Wei. 2022. XFUND: A benchmark dataset for multilingual visually rich form understanding. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 3214–3224, Dublin, Ireland. Association for Computational Linguistics.

Xu Zhong, Jianbin Tang, and Antonio Jimeno Yepes. 2019. Publaynet: largest dataset ever for document layout analysis. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1015–1022. IEEE.

A Selection of Color Extraction Tools

In Addition to `pdfplumber`. We also experimented with the `PyMuPDF`[29] package for color extraction, enticed by its capacity to extract colors as integer values. However, it uses the sRGB color space, which introduced mismatches between our annotations and extracted values. A notable misalignment was observed with colors #000000 and #000100 in RGB are both being misconstrued as the singular color #000000 in sRGB. As `PyMuPDF` features in able to extract more refined information, such as block number, line number, whether it is italicized or superscript, and so on. Some valuable information is lost by not using this package. We will continue to study underlying logic of the packages and adjust the implementation described in this paper.

B Note on Non-hierarchical Structure of Document

In the realm of document digitization, there's ongoing debate regarding the hierarchical structure of articles. Specifically, discussions revolve around overlapping structures within the XML and TEI communities[30] (Marcoux et al., 2013; Hasibi and Bratsberg, 2014). Overlaps arise when a document embodies multiple structures that intersect non-hierarchically, making it impossible to represent the document as a tree. Such as a metrical structure of feet and lines in poetry. Since our exported tree isn't an XML file, we can sidestep the non-hierarchical structure issue by distinctly defining section elements and overlapping elements.

C Inconsistencies to DocBank

There are several categories in Table 1 that differ very much. In this section we summarize the potential causes from two perspectives.

C.1 Labeling Strategies

Among them Equation and Title have particularly high recall values and very low precision. Additionally, Paragraph's recall is relatively low. This is due to the difference in annotation strategies between DocBank and our approach.

More specifically, our rules recognize more commands as Title and Equation than Paragraph in DocBank.

- For Title, DocBank only recognizes the exact `\title{}`, while LaTeX Rainbow framework recognizes all commands that contain the word title. Such as `\aistatstitle{}` or `\begin{title}`.
- For Equation, we not only recognize specific commands like `\begin{equation}`. We also annotate the mathematical expressions on the line. We do realize that it can introduce potential inaccuracies, for example in practice we have found in-line formulas used as italics instead of mathematical formulas. We will refine the rules in future updates.
- In DocBank, any text not color-coded, including page numbers, headers, and copyrights, is defined as a Paragraph. In contrast, LaTeX Rainbow doesn't tag these elements with a semantic layout label. This approach in DocBank seems imprecise and could introduce potential biases.

C.2 DocBank and LaTeX Rainbow annotations are not aligned

In practical we found that using tokens position doesn't consistently match the labeling. Namely, for identical pages of the same paper from arXiv, the tokens in the same locations differ between DocBank and LaTeX Rainbow. Upon close examination of the annotated pages, we observed that DocBank's annotation coordinates diverge from the arXiv document's. We summarize two reasons for this.

- Papers may have been updated since the release of DocBank and we annotate the latest version of the paper. This may result in changes to the content and layout of the paper.
- Changes in compilation environments and compiler versions may also have led to subtle differences in compiled PDF layout.

For tokens that could not be linked, we had to use matching of contexts and tokens, which may have caused misalignment. This in turn affects the evaluation.

[29] https://github.com/pymupdf/PyMuPDF

[30] https://tei-c.org/release/doc/tei-p5-doc/it/html/NH.html

The evaluation is available as a Jupyter notebook in the GitHub repository.