PDF-to-Tree: Parsing PDF Text Blocks into a Tree

Anonymous ACL submission

Abstract

In PDF documents, the reading order of text blocks is missing, which can hinder machine understanding of the document's content. Existing works try to extract one universal reading order for a PDF file. However, applications, like Retrieval Augmented Generation (RAG), require breaking long articles into sections, subsections and table cells for better indexing. For this reason, this paper introduces a new task and dataset, PDF-to-Tree, which organizes the text blocks of a PDF into a tree structure. Since a PDF may contain thousands of text blocks, far exceeding the number of words in a sentence, this paper proposes a transition-based parser that uses a greedy strategy to build the tree structure. Compared to the parser for plain text, we also use multi-modal features to encode the parser state. Experiments show that our approach achieves an accuracy of 93.93%, surpassing the performance of baseline methods by an improvement of 12.22%.

1 Introduction

Document AI is a research field that has emerged in recent years. It focuses on automating the reading, comprehension, and analysis of data in PDF documents. These documents can be either scanned or digital-born(rendered) files. Although many PDFs are digital-born, their formats were designed for layout purposes. As a result, the structural information retained within them is often incomplete, which can hinder machine understanding of the document's content.

There's a lot of research on PDF layout analyzing, like categorizing text blocks and predicting the relationships between them. Earlier studies relies on purely visual features, such as, DeepDeSRT(Schreiber et al., 2017), PDFTableDection(Hao et al., 2016), VisualDetection(Soto and Yoo, 2019) and dhSegment(Ares Oliveira et al., 2018) Later research incorporates visual, textual, and positional information. Such works inlude LayoutLM serials (Xu et al., 2020), (Xu et al., 2021a), (Xu et al., 2021b), DocStruct(Wang et al., 2020), SPADE(Hwang et al., 2021), StructuralLM(Li et al., 2021a) and StrucTexT(Li et al., 2021b). There are also many datasets developed in this area, including, RVL_CDIP(Harley et al., 2015) FUNSD(Jaume et al., 2019a), EPHOIE(Jaume et al., 2019b), PubLayNet(Zhong et al., 2019), and SROIE(Huang et al., 2019) and DocBank(Li et al., 2020).

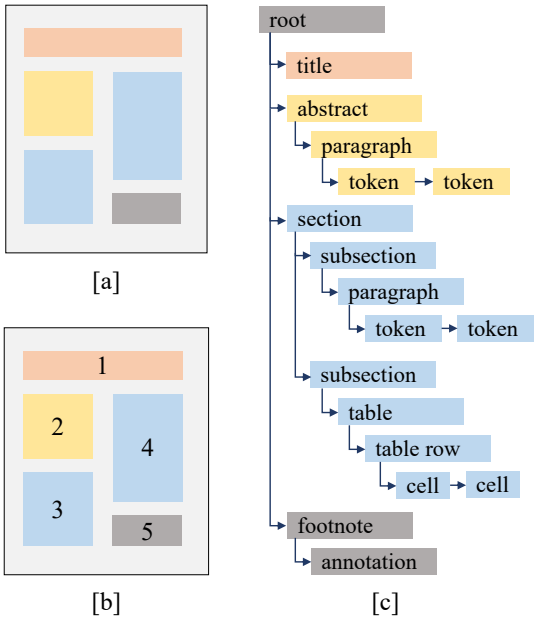
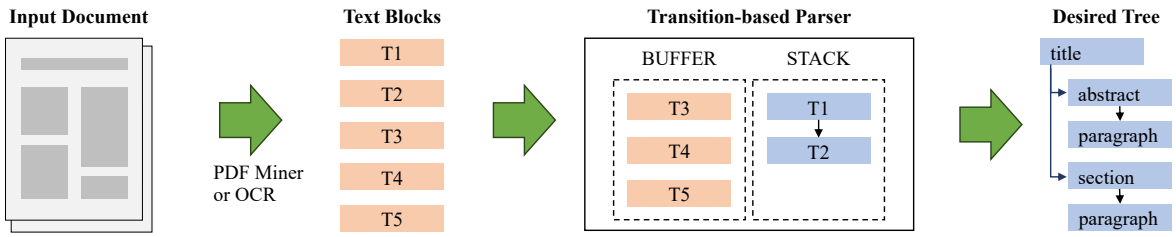


Figure 1: In a PDF([a]), text blocks are independent of each other and don't have a specific order. The reading order prediction task([b]) can partially resolve this issue. However there's only one text sequence, footnotes, captions and other irrelevant text blocks are inserted into the main text sequence. This might lead to confusion. Additionally, some applications, like RAG, require breaking down long articles into sections for better content retrieval. To tackle this issue, we propose a new task and dataset, PDF-to-Tree task([c]), which organizes text blocks into a tree structure for downstream task to retrieve.



Step	Stack	Buffer	Transition	A ← ∅
0	[ROOT]	[T1, T2, T3, T4, T5]	ARC(ROOT, T1, title)	AU (ROOT, T1, title)
1	[ROOT, T1]	[T2, T3, T4, T5]	ARC(T1, T2, abstract)	AU (T1, T2, abstract)
2	[ROOT, T1, T2]	[T3, T4, T5]	SHIFT	A
...	...	...	...	...

Figure 2: We leverage a transition-based parser to predict tree structure of a document from a sequence of input text blocks. The input document is processed by PDF Miner or OCR tools to get text block sequence. Then the sequence is processed by the parser to predict a serial of transition actions that build the tree. The parser archive this by using a buffer to hold input sequence and a stack to hold intermediate tree nodes. In each step, the parser predicts a *shift* or *arc* operation that pops elements from the buffer and reconstructs the tree in the stack.

To address the issue of missing structure information in PDF documents, existing research develops the task of reading order prediction. This involves predicting a global rank for each text block and linking them into a text sequence. Such efforts include LayoutParser(Shen et al., 2021), LayoutReader(Wang et al., 2021), ERNIE-layout(Peng et al., 2022).

However, we believe that placing all text blocks in the same sequence is insufficient. This is because there are independent document elements like footnotes and captions, which should have their own reading order. Especially for RAG applications, which require breaking down long documents into sections, subsections and table rows, a more detailed method is needed to represent the content of PDF documents. Therefore, we introduce the PDF-to-Tree task and dataset. By organizing text blocks into a tree structure, we aim to solve the issue of complex document structures. We manually annotate the tree structures of 9,310 PDF pages. As shown in Figure 1, through the PDF-to-Tree task, text blocks in a PDF are organized into a tree structure. Compared to reading unordered text blocks directly from a PDF file, downstream tasks can accurately access the content needed from the document through its tree structure. This dataset will be published on GitHub after the anonymity period ends.

Additionally, we propose a transition-based parser that effectively completes the PDF-to-Tree task. There are multiple ways to construct a tree from a sequence. Considering a PDF may contain thousands of text blocks, depending on the document's length, we opt for a transition-based parser to address this issue. Compared to other algorithms (the minimum spanning tree or pairwise linking methods), the time and memory complexity of a transition-based parser scales linearly with the length of the document. Unlike sentence dependency parsing, text blocks in PDF documents contain more information besides text, such as visual details and layout. We also use features from different modalities to encode the parser state.

In general, our main contributions are in three folders: 1) We introduce a new task and dataset for digital document understanding, called PDF-to-Tree. This task converts PDFs into a tree structure, making it easier for downstream tasks, like RAG, to precisely locate content in the articles. 2) We develop a transition-based parser for implementing PDF-to-Tree. This approach scales linearly with document length, and can handle PDFs made up of thousands of text blocks. We also use multi-modal features to encode the parser state. 3) Our experiments show that our method achieves an accuracy of 93.93%, which is 12.12% higher than the baseline methods.

2 Method

In this section, we discuss how to reconstruct document structure from a sequence of input text blocks.

2.1 PDF-to-Tree

To reconstruct the tree structure of a document, we leverage a transition-based parser. Given an input document d, we first extract all text blocks

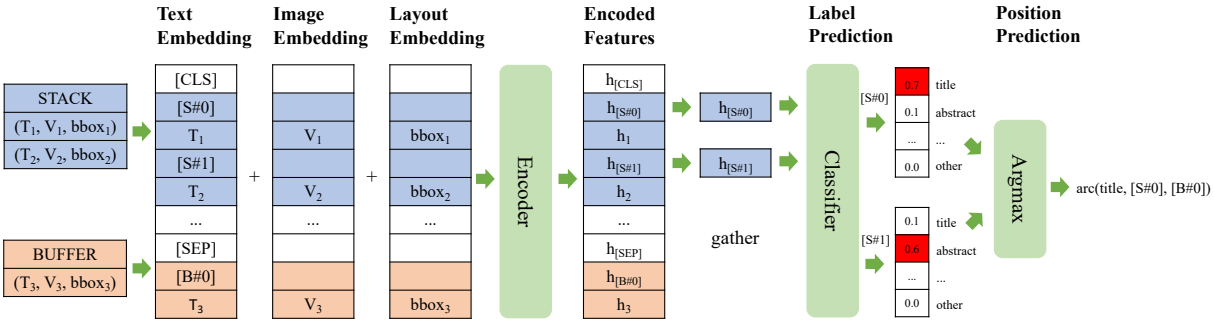


Figure 3: An overall illustration of transition prediction

$[T_1, T_2, ..., T_n]$ from d with PDF Miner or OCR tools depends on whether the input is a scanned or digital born document. Then a transition-based parser turns text block sequence into a tree. Let's denote the final desired tree by the set of arcs in that tree, $\hat{A} = [(head, tail, label), ...]$. Now the goal of PDF-to-Tree is to predict $\hat{A}$.

We can archive that via a transition-based parser with a configuration c consists of a stack s, a buffer b, and a set of arcs A. In the initial state, $A = \phi$, $s = [ROOT]$, and $b = [T_1, T_2, ..., T_n]$. In each step, the classifier predicts actions based on the content of s and b, as shown in Figure 2. At the end of each step, A is updated by adding the predicted arc into the set, $A \leftarrow A \cup [predicted_arc]$. Eventually, we will get our desired tree, $A \rightarrow \hat{A}$.

Specifically, in each step we predict the following actions:

- SHIFT - pop the first element and push it into the stack.
- ARC - create a new arc from any element of the stack to the first element in the buffer and predict the label of arc.

As Equation 1 shows, our parser needs to predict not only the label of the arc but also the starting point of the arc. In practice, we limit the start position of an arc with in a fixed size window of the stack to ease the prediction process.

$$label, arc_start = classifier(a, b, A) \quad (1)$$

2.2 Transition Prediction

Figure 3 illustrates the process of PDF-to-Tree predicting transition actions based on the current configuration at the current step. First, embeddings are created for the nodes in the stack and buffer. Text, bounding boxes and their corresponding images are concatenated together in the order they appear in the stack. The concatenated sequence is then separately embedded for text, image, and layout. These embeddings are combined according to their positions and served as inputs for the encoder. The encoder produces hidden state for each node, and classification is performed to obtain labels and the starting position of arcs.

Given the current configuration of a parser, denoted as $c = (s, b, A)$, we use the following notations to introduce the out model. n_i represents a node from either s or b. t_i represents the text within the node n_i, $bbox_i$ represents the bounding box of n_i, and v_i represents the image information of n_i.

2.2.1 Text Embedding

We connect the t_i from nodes in both s and b in sequential order to form a sequence S. Before each node n_i, a special token is inserted as a separator. For nodes in s, we use the special character $[S\#i]$, and for nodes in b, we use $[B\#i]$. As a result, we obtain the sequence S as shown in Equation 2.

$$S = [CLS], [S\#0], t_0, [S\#1], t_1, ..., [SEP], [B\#0], t_n. \quad (2)$$

2.2.2 Layout Embedding

For each node, n_i, besides embedding t_i, it's also necessary to embed layout information, $bbox_i$. To be specific, we employ four distinct types of layout embedding, including absolute position, relative position, bounding box size(width and height), and font size. Absolute position refers to the coordinates $bbox_i = (x_0^i, y_0^i, x_1^i, y_1^i)$ of the bounding box. Relative position indicates the position of the bounding box relative to the first node in the buffer, $bbox_{b_0} = (x_0^{b_0}, y_0^{b_0}, x_1^{b_0}, y_1^{b_0})$. If $bbox_i$ and $bbox_{b_0}$ are from different pages, then based on the page number, the model will add

the corresponding page height to the y-coordinate of the bounding box below. All coordinates are normalized to the range of 0 to 1000. The results of embedding b_i are also ordered according to the corresponding t_i sequence, and the embeddings' outcomes at each position are averaged.

$$Emb_{layout} = \begin{cases} (x_0^i, y_0^i, x_1^i, y_1^i) \\ (x_0^i, y_0^i, x_1^i, y_1^i) - (x_0^{b_0}, y_0^{b_0}, x_1^{b_0}, y_1^{b_0}) \\ (w, h), \text{width and height} \\ (fs), \text{font size} \end{cases} \quad (3)$$

2.2.3 Image Embedding

Out model embeds visual inputs with LayoutLM(Xu et al., 2020). Specifically, LayoutLMv1/v2 employs ResNet to embed images, while LayoutLMv3 uses a transformer to embed image patches. For situations where arcs span across pages, page images are concatenated. All page images, including those stitched together for spanning page elements, are resized to 512 x 512.

2.2.4 Label and Position Prediction

After completing the embedding for all the modalities, the model will sum the embeddings for corresponding positions together. Then, it will encode the sequence of embeddings to get the hidden state, denoted as h_S, for the input sequence S. For each node in stack s, the model extract the hidden value at the position of the special character $[S\#i]$ from h_S, to get $h_{[S\#0]}, h_{[S\#1]}, ..., h_{[S\#n]}$. As Equation 4 shows, the model will put the selected hidden state through a bi-linear module and obtain classification results for each node in the stack. Let's denote $label_{[S\#i]}$ as the label of node n_i in the stack, and $score_{[S\#i]}$ as the corresponding score. Finally, the model will take $[S\#i]$ with the maximum score($score_{[S\#i]}$) as the start position for the predicted arc and corresponding label as arc label, as Equation 5 illustrates.

$$label_{[S\#i]}, score_{[S\#i]} = classifier(h_{[S\#i]}) \quad (4)$$

$$arc_start = \underset{[S\#i]}{argmax}(score_{[S\#i]}) \quad (5)$$

3 Experiments

In this section, we dive into the implementation details of PDF-to-Tree and conduct experiments on the PDF-to-Tree dataset. Also, we create a baseline with SPADE(Hwang et al., 2021) and StrucTexTv1/v2(Li et al., 2021b), (Yu et al., 2023). Those models are commonly used approaches in structured text understanding. All experiments were carried out using one to eight NVIDIA Tesla A800 80GB GPUs.

Labels	Train	Test	Dev
# of documents	1,040	129	129
# of pages	7,554	786	970

Table 1: Statistics of training, development, and testing sets

3.1 Dataset

We introduce a new dataset called PDF-to-Tree, which annotates the text blocks in each document into a tree structure. In contrast, previous entity linking datasets, such as FUNSD(Jaume et al., 2019a), EPHOIE(Jaume et al., 2019b), SROIE(Huang et al., 2019), mainly focused on information extraction, containing only partial text blocks from single pages, such as forms. As shown in Figure 4, the annotation information consists of two parts: text blocks with bounding boxes, and arcs that represent the relationships between the text boxes. Please refer to Appendix A for more details.

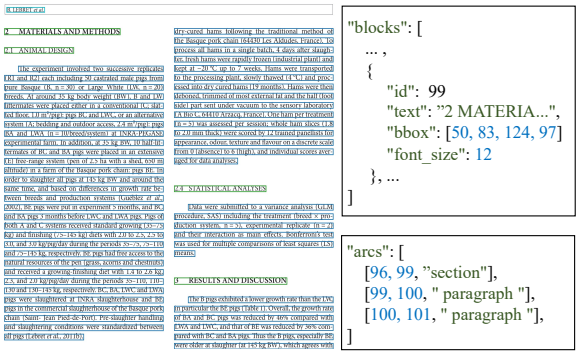


Figure 4: An annotation example of the PDF-to-Tree Dataset. We use code to automatically extract text blocks from PDFs and then manually annotate the relationships between these blocks to create a tree structure.

All the files in the PDF-to-Tree dataset are digital-born(rendered) PDFs from public domain, including product manuals, public technical reports, white papers, and so on. We use the open-source tool PDF Miner to automatically extract text blocks from PDF documents. Then, we manually label

the relationships between the blocks to create a tree structure. For each document, we have at least two crowdsourced annotators working on it. All of our annotators are college students. We pay $0.20 per page for the annotation. Please refer to the Appendix A for the details of our annotation tool.

In total, we gather 1290 documents comprising 9310 pages. There are 18 categories of labels. We allocate 80% of the documents for training, 10% for testing, and retain 10% as a development set. Specific data proportions are detailed in Table 1.

Additionally, in previous document layout datasets, like DocBank(Li et al., 2020), PubLayout(Zhong et al., 2019), most commonly, academic papers were used as document sources(such as LaTeX or XML), and layout annotations were automatically generated by analyzing the source files. While this covered a larger number of documents, the layouts were relatively uniform. On the other hand, the PDF-to-Tree dataset employs a combination of machine and human annotations. This means that even when the source files of the documents are not available, complete structural information of the document can still be obtained. As a result, the entire collection of documents encompasses a wider range of layouts, including elements like product manuals and public technical reports, which are not typically found in academic papers. Additionally, due to this diversity of document types, the distribution of document lengths also varies considerably, ranging from a minimum of 1 page to a maximum of 85 pages.

3.2 Baseline

In current Document AI work, entity linking is used to predict the relationships between text blocks, such as arcs in the PDF-to-Tree task. Therefore, we select three methods of entity linking to build our baseline, including SPADE (Hwang et al., 2021) and StrucTextV1 (Li et al., 2021b), and StrucTextV2(Yu et al., 2023). These models outperform other methods on entity linking tasks of the FUNSD dataset. We use the code provided by the original authors. However, the code of StrucText doesn't include the fine-tuning code, we implement that part ourselves according to the description mentioned in the original paper. After the anonymous period ends, we plan to open-source this training code along with our own model code.

Since PDF-to-Tree has documents up to 85 pages in length, it's not realistic to fit all the text blocks of a document into the input window size of aforementioned baseline models. To tackle this, we preprocess the dataset, dividing documents into blocks of no more than 500 tokens each for training and prediction and ignore the arcs between blocks. This simplification will only affect approximately 5% arcs in test set.

3.3 Training

For our PDF-to-Tree model, we opt for both text-only and text-image multi-modal pre-trained models as encoders, comparing how different modalities affect the outcomes. Specifically, we choose BERT for text-only pre-training models, and LayoutLM for the multi-modal pre-training model. We utilized both the base and large versions of the aforementioned pre-trained models in our training code. We use PyTorch to implement our model and the pre-trained weights are provided by Hugging Face.

Throughout training, our model employs the AdamW optimizer and a linear warm-up scheduler for the initial 10% of steps. Cross entropy loss is used for label and position prediction. We conduct hyperparameter searches for learning rate, batch size, and dropout using the dev dataset. For the base version of BERT and ReBERTa, we use a learning rate of 4×10^{-5}, while for the large version, we use a learning rate of 2×10^{-5}. Comparatively, for LayoutLMv1/2/3, a smaller learning rate is needed to make the model converge. We ultimately chose 2×10^{-5} as the learning rate for the base version and 1×10^{-5} for the large version. In all cases, the dropout is set to 0.1, the batch size is 32, and the number of epochs is 6. Our code and dataset will be open-sourced after the anonymous period ends. For the baseline models, we follow the hyperparameters provided in the original paper.

3.4 Metrics

To assess the accuracy of the model in reconstructing document structure, we utilize the attachment score, which is widely used metrics in dependency parsing. Unlabeled attachment score(UAS) is the percentage of tokens with correctly assigned heads, while labeled attachment score (LAS) is the percentage of tokens with correctly assigned heads and dependency relation labels. We define UAS and LAS in the PDF-to-Tree task by replacing tokens with text blocks, as Equation 6 and 7 illustrates. The primary emphasis of UAS lies in

Model†	Modality‡	Parameters	UAS	LAS
SPADE(Hwang et al., 2021)	T+L+V	-	0.5562	0.3445
StrucTexTv1$_{base}$(Li et al., 2021b)	T+L+V	110M	**0.8171**	**0.6402**
StrucTexTv2$_{small}$(Yu et al., 2023)*	V	28M	0.7681	0.5454
P2T$_{bert}$	T+L	110M	0.9158	0.7900
P2T$_{layoutlm}$	T+L	160M	0.9229	0.7551
P2T$_{layoutlmv2}$	T+L+V	220M	0.9338	0.7994
P2T$_{layoutlmv3}$	T+L+V	133M	**0.9385**	**0.8020**
P2T$_{bert\text{-}large}$	T+L	340M	0.9189	0.7757
P2T$_{layoutlm\text{-}large}$	T+L	390M	0.9233	0.7836
P2T$_{layoutlmv2\text{-}large}$	T+L+V	426M	0.9363	0.8070
P2T$_{layoutlmv3\text{-}large}$	T+L+V	368M	**0.9393**	**0.8166**

† P2T stands for PDF-to-Tree.
‡ "T" refers to text, "L" refers to layout and "V" refers to visual.
* We use the entity linking code of StructTexTv1 for StructTexTv2.

Table 2: Accuracy on PDF-to-Tree Dataset. All the scores are the average of 5 runs.

the precision associated with the construction of the document's structure. LAS takes into account both the labels and the links of text blocks.

$$\text{UAS} = \frac{\text{\# of blocks with correct } \textit{link}}{\text{\# of all blocks}} \tag{6}$$

$$\text{LAS} = \frac{\text{\# of blocks with correct } \textit{link} \text{ and } \textit{label}}{\text{\# of all blocks}} \tag{7}$$

4 Results

In this section, we compare PDF-to-Tree with the baseline models. We also evaluate the effects of various modality encoders on structure parsing for the PDF-to-Tree dataset. Additionally, we perform error analysis, ablation experiments, and inference speed analysis.

4.1 Accuracy

In general, the transition-based parser module shows a significant improvement in the task of document-level structure parsing. Our method, PDF-to-Tree shows substantial increases to the baseline in both the UAS and LAS . Specifically, PDF-to-Tree outperforms the baseline models by 12.22% in the UAS, and 17.64% in the LAS. Even taking into consideration the 5% of arcs omitted by baseline models due to the window size, this still represents a notable improvement. This indicates that incorporating the transition-based parser module effectively filters out many irrelevant pairs and enhances the precision of link prediction, compared to the pairwise linking strategy in the baseline. This approach effectively leverages the inherent characteristics of the document structure.

Furthermore, the modality of the pretrained weights also plays an important role in the results. By leverage the text-image multi-modal pretrained weights, LayoutLMv3, the UAS and LAS are improved, comparing to BERT, which are pre-trained solely on text. Among the LayoutLM series, LayoutLMv1 only uses image embeddings during pre-training. On the other hand, LayoutLMv2/3 utilize image embeddings in both pre-training and fine-tuning. Consequently, when employing LayoutLMv3, the PDF-to-Tree model achieves the highest UAS and LAS scores.

4.2 Ablation Study

Despite of visual modality, we also want to know how the other types of modalities affect the outcomes. As shown in Table 3, by removing layout as input, the model's UAS and LAS decrease by 3.25% and 2.52%. It shows the layout input is helpful for document structure parsing. It's also worth mentioning that when we only use layout information, the model surprisingly manages to achieve decent results. UAS reaches 83.15%, while LAS reaches 78.07%. It indicates that the model can still achieve a certain level of document structure parsing ability solely relying on layout information.

Model	Modality	UAS	LAS
$P2T_{layoutlmv3}$	T+L+V	0.9393	0.8166
$P2T_{bert}$	T+L	0.9158	0.7900
$P2T_{bert\text{-}wo\text{-}layout}$	T	0.8833	0.7280
$P2T_{bert\text{-}wo\text{-}text}$	L*	0.8315	0.5161

* P2T can still achieve a certain level of accuracy solely relying on layout information.

Table 3: Ablation Study on PDF-to-Tree Dataset

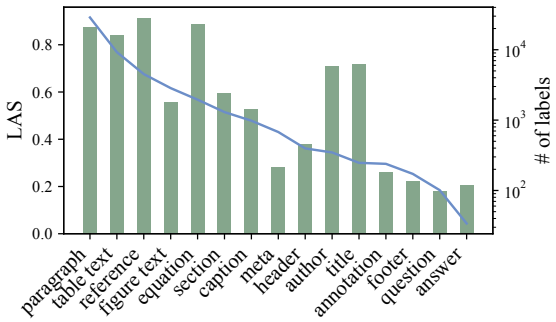


Figure 5: Labeled Attachment Score of Different Labels. The bars illustrate the LAS of labels. The line shows the number of labels. Labels with lower occurrence rates exhibit much lower scores. Some document components, such as meta and header, have a sufficient number of annotations, but their scores are not high due to their varied forms.

4.3 Score of Different Labels

Besides overall accuracy, we also analyze the accuracy for each label. As shown in the figure, the model performs well in predicting more common labels, such as paragraph, table, and reference. It also does well with simpler labels like authors and titles. The model finds labels like meta and header, which vary a lot in form, the most challenging. Compared to the least common labels, meta and header have a decent number of samples, but their scores are still not good. This is because document headers and meta information are more varied than fixed elements like titles and paragraphs. Overall, these results are as expected. Handling these rare and variably formatted tags is still a challenging task.

4.4 Inference Speed

We choose PDF-to-Tree$_{layoutlmv3}$ to compare the inference speed with the baseline models because these models share the similar parameter size and modality. As shown in Table 4, due to PDF-to-Tree encoding each state during the parsing process, it's slower in speed compared to the baselines.

Model	Sec / Page
SPADE(Hwang et al., 2021)	0.511
StrucTexT$_{base}$(Li et al., 2021b)	0.262
$P2T_{layoutlmv3}$	1.138

Table 4: The inference speed of P2T is slightly slower than the baseline. However, considering that all cross-page links are ignored in the baseline models, this speed difference is acceptable.

Specifically, on PDF-to-Tree$_{layoutlmv3}$, the average time to complete predictions for one page is 1.138 seconds, whereas the baselines only needs 0.511 and 0.262 seconds. Considering that the baseline models ignore all arcs cross pages due to not being able to fit the entire document into memory, this difference is acceptable. Moreover, the time cost of PDF-to-Tree only depends on document length. As depicted in Figure 6, with an increase in pages, the time cost of PDF-to-Tree grows linearly. In the future, we can further improve the inference speed of PDF-to-Tree by optimizing the encoding process.

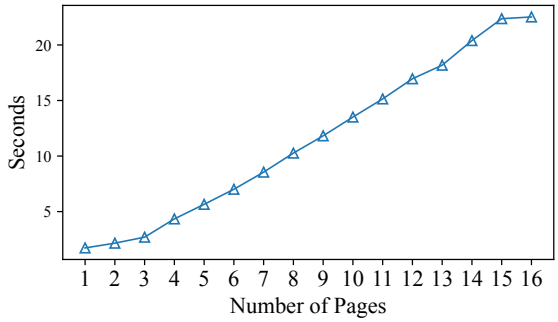


Figure 6: The inference cost of PDF-to-Tree increases linearly with the length of the document.

5 Related Work

Document AI is a research area that has gained attention in recent years. There's a lot of valuable information stored in the form of digital documents. The goal is to extract and convert digital documents into structured data. Jaume et al. divides Document AI into two subtasks: One involves categorizing blocks within the document to obtain labels for these blocks, which is called entity labeling. The other involves establishing connections between blocks to identify the relationships between them, which is known as entity linking.

5.1 Entity Labeling

For entity labeling, many studies use the NER framework to label sequences at the token level. These studies include BERTgrid(Denk and Reisswig, 2019), Post-OCR-Parsing(Hwang et al., 2019). Additionally, there are some studies that aim to combine the spatial information of text blocks for labeling sequences at the block level. Examples of such studies are GraphIE(Qian et al., 2018), TRIE(Zhang et al., 2020), LayoutParser(Shen et al., 2021), LayoutLM serials(Xu et al., 2020), (Xu et al., 2021a), (Xu et al., 2021b). Also Wang et al. finds that formatting can disrupt sequence labeling. In addition to providing semantic labels for text blocks, predicting the reading order is also necessary. Research in this field includes works like LayoutReader(Wang et al., 2021), ERNIE-Layout(Peng et al., 2022).

5.2 Entity Linking

Studies, such as dhSegment(Ares Oliveira et al., 2018), DocStruct(Wang et al., 2020), StrucTexT serials(Li et al., 2021b), (Yu et al., 2023) combine these two tasks and perform Entity Labeling and Linking at the block Level simultaneously. Those studies mainly deal with blocks of individual pages. SPADE(Hwang et al., 2021) formulates entity linking as a spatial dependency parsing problem. However the linking strategy is pair-wise. What sets this paper apart is the utilization of a transition-based parser for constructing entity links.

Numerous datasets have been introduced to support research in this direction, such as FUNSD(Jaume et al., 2019a), EPHOIE(Jaume et al., 2019b), SROIE(Huang et al., 2019). Unlike our work, these datasets usually focus only on information extraction. The annotated text blocks and relationships often cover only part of the information on a single page. In contrast, the PDF-to-Tree dataset and task aim to organize the information from an entire document into a tree structure.

5.3 Multi-Modal Feature Representation

Early works typically involved using a single mode for predictions, either text or images. LayoutLM(Xu et al., 2020) find that utilizing multi-modal data can significantly enhance the model's performance in understanding structured text. Similar works include StructuralLM(Li et al., 2021a), StrucTexT serials(Li et al., 2021b), (Yu et al., 2023). Meanwhile, datasets like DocBank(Li et al., 2020), PubLayout(Zhong et al., 2019), and RVL-CDIP(Harley et al., 2015) are introduced to support pretraining for layout understanding. These datasets share the common characteristic of being annotated directly from the source code of electronic documents. The advantage of this approach is the rapid acquisition of a substantial amount of labeled data. However, the downside is that it might result in relatively uniform document layouts, mostly sourced from academic papers. In this paper, we employed a combination of automated and manual annotation. This enables support for a broader range of document types, including product manuals, public technical reports, white papers, and so on.

6 Discusion

In this paper, we discuss the task of document structure parsing. This task is more intricate compared to the traditional reading order prediction. This complexity arises because we need to consider connecting paragraphs across pages and linking paragraphs into sections. To address these challenges, we introduce a transition-based parser as a solution. Alongside this, we introduce a new dataset called PDF-to-Tree to support this task. Experimental results demonstrate the effectiveness of our approach. However, there is still room for improvement in identifying less common labels. Moreover, there are areas where the efficiency of inference can be enhanced.

7 Limitation

All the PDFs used in the PDF-to-Tree dataset are born digital(rendered). In theory, our method could also be applied to scanned documents. However, due to resource constraints, it has not been used on the PDF-to-Tree dataset yet. In future work, we plan to include scanned documents in our dataset as well. Additionally, the high cost of manual labeling limits the amount of annotated data we can obtain. In future work, we believe it's worth discussing how to automatically label the tree structure of a document.

References

Sofia Ares Oliveira, Benoit Seguin, and Frederic Kaplan. 2018. dhsegment: A generic deep-learning approach for document segmentation. In

2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR).

Timo Denk and Christian Reisswig. 2019. Bertgrid: Contextualized embedding for 2d document representation and understanding.

Leipeng Hao, Liangcai Gao, Xiaohan Yi, and Zhi Tang. 2016. A table detection method for pdf documents based on convolutional neural networks. In *2016 12th IAPR Workshop on Document Analysis Systems (DAS)*.

AdamW. Harley, Alex Ufkes, and KonstantinosG. Derpanis. 2015. Evaluation of deep convolutional nets for document image classification and retrieval. *Cornell University - arXiv,Cornell University - arXiv*.

Zheng Huang, Kai Chen, Jianhua He, Xiang Bai, Dimosthenis Karatzas, Shijian Lu, and C. V. Jawahar. 2019. Icdar2019 competition on scanned receipt ocr and information extraction. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*.

Wonseok Hwang, Seonghyeon Kim, Minjoon Seo, Jinyeong Yim, Seunghyun Park, Sungrae Park, Junyeop Lee, Bado Lee, and Hwalsuk Lee. 2019. Post-ocr parsing: building simple and robust parser via bio tagging.

Wonseok Hwang, Jinyeong Yim, Seunghyun Park, Sohee Yang, and Minjoon Seo. 2021. Spatial dependency parsing for semi-structured document information extraction. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*.

Guillaume Jaume, Hazim Kemal Ekenel, and Jean-Philippe Thiran. 2019a. Funsd: A dataset for form understanding in noisy scanned documents. In *2019 International Conference on Document Analysis and Recognition Workshops (ICDARW)*.

Guillaume Jaume, Hazim Kemal Ekenel, and Jean-Philippe Thiran. 2019b. Funsd: A dataset for form understanding in noisy scanned documents. In *2019 International Conference on Document Analysis and Recognition Workshops (ICDARW)*.

Chenliang Li, Bin Bi, Ming Yan, Wei Wang, Songfang Huang, Fei Huang, and Luo Si. 2021a. Structurallm: Structural pre-training for form 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)*.

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*.

Yulin Li, Yuxi Qian, Yuechen Yu, Xiameng Qin, Chengquan Zhang, Yan Liu, Kun Yao, Junyu Han, Jingtuo Liu, and Errui Ding. 2021b. Structext: Structured text understanding with multi-modal transformers. In *Proceedings of the 29th ACM International Conference on Multimedia*.

Qiming Peng, Yinxu Pan, Wenjin Wang, Bin Luo, Zhenyu Zhang, Zhengjie Huang, Teng Hu, Weichong Yin, Yongfeng Chen, Yin Zhang, Shikun Feng, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. 2022. Ernie-layout: Layout knowledge enhanced pre-training for visually-rich document understanding.

Yujie Qian, Enrico Santus, Zhijing Jin, Jiang Guo, and Regina Barzilay. 2018. Graphie: A graph-based framework for information extraction. *arXiv: Computation and Language,arXiv: Computation and Language*.

Sebastian Schreiber, Stefan Agne, Ivo Wolf, Andreas Dengel, and Sheraz Ahmed. 2017. Deepdesrt: Deep learning for detection and structure recognition of tables in document images. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*.

Zejiang Shen, Ruochen Zhang, Melissa Dell, Benjamin Charles Germain Lee, Jacob Carlson, and Weining Li. 2021. *LayoutParser: A Unified Toolkit for Deep Learning Based Document Image Analysis.*, page 131–146.

Carlos Soto and Shinjae Yoo. 2019. Visual detection with context for document layout analysis. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

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*.

Zilong Wang, Mingjie Zhan, Xuebo Liu, and Ding Liang. 2020. Docstruct: A multimodal method to extract hierarchy structure in document for general form understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*.

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)*.

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.

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. *arXiv: Computation and Language,arXiv: Computation and Language*.

Yuechen Yu, Yulin Li, Chengquan Zhang, Xiaoqiang Zhang, Zengyuan Guo, Xiameng Qin, Kun Yao, Junyu Han, Errui Ding, and Jingdong Wang. 2023. Structextv2: Masked visual-textual prediction for document image pre-training.

Peng Zhang, Yunlu Xu, Zhanzhan Cheng, Shiliang Pu, Jing Lu, Liang Qiao, Yi Niu, and Fei Wu. 2020. Trie: End-to-end text reading and information extraction for document understanding. In *Proceedings of the 28th ACM International Conference on Multimedia*.

Xu Zhong, Jianbin Tang, and AntonioJimeno Yepes. 2019. Publaynet: largest dataset ever for document layout analysis. *Cornell University - arXiv,Cornell University - arXiv*.

A Dataset Annotation

The goal of the PDF-to-Tree annotation task is to extract text blocks from a PDF file and label their relationships in a tree structure. We start by using the open-source tool PDF Miner to extract text blocks from the PDF. Any incorrectly extracted blocks are manually corrected. Next, we use multi-level numbering to label the relationships between text blocks. Finally, we can add arcs between adjacent text blocks with serial numbers to form a tree structure.

Figure 7: An illustration of annotation tool used for the PDF-to-Tree dataset.

As Figure 7 illustrates, with two-level numbering, the first level represents the global order of document components, and the second level represents the order of the text block within the document component. For instance, the label "paragraph-3-2" means that this text block is is the second block within that paragraph and the paragraph is the third component in the entire document. Please note that the numbering is not continuous. We use number to represent relative order, making it easy to insert new labels anywhere in the sequence. For example, we can insert 15 between 10 and 20.

For more complex components like tables, we can extend to more levels of numbering, such as using the second level for row numbers and the third level for column numbers. For example, the label "table-5-3-1" indicates that it is the first cell in the third row of the table, which is the fifth element in the article.