DocParser: Hierarchical Document Structure Parsing from Renderings

Johannes Rausch, 1 Octavio Martínez, 1 Fabian Bissig, 1 Ce Zhang, 1 Stefan Feuerriegel 2

1 Department of Computer Science, ETH Zurich
2 Department of Management, Technology, and Economics, ETH Zurich
johannes.rausch@inf.ethz.ch, octaviom@student.ethz.ch, fbissig@student.ethz.ch
ce.zhang@inf.ethz.ch, sfeuerriegel@ethz.ch

Abstract

Translating renderings (e.g. PDFs, scans) into hierarchical document structures is extensively demanded in the daily routines of many real-world applications. However, a holistic, principled approach to inferring the complete hierarchical structure of documents is missing. As a remedy, we developed “DocParser”: an end-to-end system for parsing the complete document structure — including all text elements, nested figures, tables, and table cell structures. Our second contribution is to provide a dataset for evaluating hierarchical document structure parsing. Our third contribution is to propose a scalable learning framework for settings where domain-specific data are scarce, which we address by a novel approach to weak supervision that significantly improves the document structure parsing performance. Our experiments confirm the effectiveness of our proposed weak supervision: Compared to the baseline without weak supervision, it improves the mean average precision for detecting document entities by 39.1% and improves the F1 score of classifying hierarchical relations by 35.8%.

1 Introduction

The structural and layout information in a document can be a rich source of information that facilitates Natural Language Processing (NLP) tasks (e.g. information extraction). Over the years, the NLP community has developed a range of techniques to detect, understand, and take advantage of document structures (Hurst and Nasukawa 2000; Chen, Tsai, and Tsai 2000; Tengli, Yang, and Ma 2004; Luong, Nguyen, and Kan 2012; Govindaraju, Zhang, and Ré 2013; Katti et al. 2018; Schäfer et al. 2011; Schäfer and Weitz 2012; Garnarék et al. 2020).

However, structural information in documents is becoming increasingly challenging to obtain — many file formats that are prevalent today are being rendered without structural information. Prominent examples are PDF documents: this file format benefits from portability and immutability, yet it is flat in the sense that it stores all content as isolated entities (e.g., combinations of characters and positions) and, thus, hierarchical information is lacking. As such, the structure behind figures and especially tables is discarded and thus no longer available to computerized analyses in NLP.

In contrast, file formats such as XML or JSON naturally encode hierarchical document structures among textual entities. Hence, techniques are required in order to convert renderings into structured, textual document representations to enable joint inference between text, layout, and other document structures.

Earlier attempts for structure parsing on documents focused on a subset of simpler tasks such as segmentation of text regions (Antonacopoulos et al. 2009), locating tables (Zanibbi, Blostein, and Cordy 2004; Embley et al. 2006), or parsing them (Schreiber et al. 2018), but not parsing complete document structures. However, document structures are required as a representation of many downstream tasks in NLP. For instance, recent efforts in the NLP community (Katti et al. 2018; Apostolova and Tomuro 2014; Liu et al. 2019) have shown that utilizing 2D document information, e.g. character and word positions, can be an effective way to improve upon standard NLP tasks such as information extraction.

A holistic, principled approach for inferring the complete hierarchical structure from documents is missing. On the one hand, such a task is nontrivial due to the complexity of documents, particularly their deeply-nested structures. For instance, nested tables are fairly easy to recognize for human readers, yet detecting them is known to impose computational hurdles (cf. Schreiber et al. 2018). On the other hand, efficient learning is prevented as large-scale training sets are lacking (cf. Arif and Shafait 2018; Schreiber et al. 2018). Notably, prior datasets are limited to table structures (Gobel et al. 2013; Rice, Jenkins, and Nartker 1995) and not the complete document structures. Needless to say, complex structures also make the labeling process significantly more costly (Wang, Phillips, and Haralick 2004). Therefore, an effective implementation that makes only a scarce use of labeled data is demanded.

This work focuses on parsing the hierarchical document structure from renderings. We develop an end-to-end system for inferring the complete document structure (see Figure 1). This includes all entities (e.g., text, bibliography regions, figures, equations, headings, tables, and table cells), as well as the hierarchical relations among them. We specifically adapt to settings in practice that suffer from data scarcity. For this purpose, we propose a novel learning framework for scalable weak supervision. It is intentionally tailored to the
We contribute “DocParser”. This presents the first end-to-end system for parsing in the following directions: (i) it includes all entities that can appear in documents (i.e., not just tables) and (ii) it includes the hierarchical relations among them. The dataset is based on 127,472 scientific articles from the arXiv repository.

2. We contribute the first dataset (called “arXivdocs”) for evaluating document parsing. It extends existing datasets for parsing in two directions: (i) it includes all entities that can appear in documents (i.e., not just tables) and (ii) it includes the hierarchical relations among them. The dataset is based on 127,472 scientific articles from the arXiv repository.

3. We propose a novel weakly-supervised learning framework to foster efficient learning in practice where annotated documents are scarce. It is based on an automated and thus scalable labeling process, where annotations are retrieved by reverse rendering the source code of documents. Specifically, in our work, we utilize \( \LaTeX \) source files from arXiv together with \texttt{synctex} for this objective. This then yields weakly-supervised labels by reverse rendering of the \( \LaTeX \) code.

4. We conduct extensive evaluation of our proposed techniques, outperforming the state-of-the-art on the related task of table parsing.

Contributions:\footnote{\textsuperscript{1}Source codes and the arXivdocs dataset are available from https://github.com/DS3Lab/DocParser.} We extend prior literature on document parsing in the following directions:

1. We contribute “DocParser”. This presents the first end-to-end system for parsing renderings into hierarchical document structures. Prior literature has merely focused on simpler tasks such as table detection or table parsing but not on the parsing of complete documents. As a remedy, we present a system for inferring document structures in a holistic, principled manner.

2. We contribute the first dataset (called “arXivdocs”) for evaluating document parsing. It extends existing datasets for parsing in two directions: (i) it includes all entities that can appear in documents (i.e., not just tables) and (ii) it includes the hierarchical relations among them. The dataset is based on 127,472 scientific articles from the arXiv repository.

3. We propose a novel weakly-supervised learning framework to foster efficient learning in practice where annotated documents are scarce. It is based on an automated and thus scalable labeling process, where annotations are retrieved by reverse rendering the source code of documents. Specifically, in our work, we utilize \( \LaTeX \) source files from arXiv together with \texttt{synctex} for this objective. This then yields weakly-supervised labels by reverse rendering of the \( \LaTeX \) code.

4. We conduct extensive evaluation of our proposed techniques, outperforming the state-of-the-art on the related task of table parsing.

2.1 Problem Description

Given a set of document renderings \( D_1, \ldots, D_n \), the objective is to generate hierarchical structures \( T_1, \ldots, T_n \). A hierarchical structure \( T_i, i = 1, \ldots, n \), consists of both entities and relations as follows:\footnote{\textsuperscript{2}For consistency, we use the term “entity” throughout the article when referring to all elements in the document structure (e.g., a figure, table, or text) that need to be detected. While the term “object” is common in computer vision, we chose the term “entity” to highlight its semantic nature for NLP.}

Entities \( E_j, j = 1, \ldots, m \), refer to the various elements within a document, such as a figure, table, row, cell, etc. Each entity is described by three attributes: (1) its semantic category \( c_j \in C \) (i.e., which defines the underlying type) and (2) the coordinates given by rectangular bounding box \( B_j \) in the document rendering. There further is (3) a confidence score \( P_j \). This is not part of the ground truth labels; however, it comes from the predictions inside the DocParser system.

Relations \( R_j, j = 1, \ldots, k \) of type \( \Psi \) are given by triples \( (E_{subj}, E_{obj}, \Psi) \) consisting of a subject \( E_{subj} \), an object \( E_{obj} \), and a relation type \( \Psi \in \{parent_of, followed_by, null\} \). The latter, null, is reserved for entities with meta-information that do not have designated order (i.e., header, footer, keywords, date, page number). All other entities must have \( \Psi \neq \text{null} \).

The combination of entities and relations is sufficient to reconstruct the hierarchical structure \( T_i \) for a document. However, generating such a hierarchical structure from a document rendering \( D_i \) is subject to inherent challenges: the similar appearance of entities impedes detection and, further, the hierarchy can be nested arbitrarily, with substantial variation across different documents.

2.2 System Components

DocParser performs document structure parsing via 5 components (see overview in Figure 2): (1) image conversion, (2) entity detection, (3) relation classification, (4) structure-based refinement, and (5) scalable weak supervision. To store document structures, we developed a customized, JSON-based file format.

Component 1: Image Conversion

[Image of a diagram showing the system overview]
Document renderings are converted into images with a predefined resolution $\rho$. Furthermore, all images are resized to a fixed rectangular size $\phi$ (if necessary, with zero padding).

The document images are further pre-processed: the RGB channels of all document images are normalized analogous to the MS COCO dataset (i.e., by subtracting the mean RGB channel values from the inputs). The reason is that all neural models are later initialized with pre-trained weights from the MS COCO dataset (Lin et al. 2014).

**Component 2: Entity Detection**

To detect all document entities within a document image, we build upon a neural model for image segmentation, namely Mask R-CNN (He et al. 2017). Specifically, it takes the images from the previous component as input and then returns a flat list of entities $E_1, \ldots, E_m$ as output. For each entity Mask R-CNN determines (i) its rectangular bounding box, (ii) confidence score, (iii) a binary segmentation mask that distinguishes between the detected entity and background pixels within the bounding box, and (iv) a category label for the entity. Our implementation makes use of 23 categories $C$: CONTENT BLOCK, TABLE, TABLE ROW, TABLE COLUMN, TABLE CELL, TABULAR, FIGURE, HEADING, ABSTRACT, EQUATION, ITEMIZE, ITEM, BIBLIOGRAPHY BLOCK, TABLE CAPTION, FIGURE GRAPHIC, FIGURE CAPTION, HEADER, FOOTER, PAGE NUMBER, DATE, KEYWORDS, AUTHOR, AFFILIATION.

**Component 3: Relation Classification**

A set of heuristics is applied to translate the flat list of entities into hierarchical relations $R_1, \ldots, R_k$. Here, we distinguish the heuristics according to whether they generate (1) the nesting among entities or (2) the ordering for entities of the same nesting level. The former case corresponds to $\Psi = parent_of$, while the latter determines all relations with $\Psi = followed_by$. In this component, we ignore all entities with meta-information, e.g. footers, as these have no designated hierarchy (cf. document grammar in the supplements).

**Relations with Nesting (parent_of):** Four heuristics $h_1, \ldots, h_4$ determine parent-child relation according to the following:

- **(h1): Overlaps** A list of candidate parent-child relations is compiled based on the overlap of bounding boxes. That is, DocParser loops over all bounding boxes and, for each bounding box $B_{sub}$, it determines all other bounding boxes that are contained within $B_{sub}$. Formally, this is given by all tuples of bounding boxes $(B_{sub}, B_{obj})$ with subj $\in m$, obj $\in m$, and subj $\neq$ obj where $h_1(B_{sub}, B_{obj})$ is satisfied: Tuples for which the bounding box of $B_{obj}$ is fully or partially enclosed by the bounding box of $B_{sub}$ are added to the candidate list. Furthermore, we add tuples to the candidate list that satisfy $area(B_{obj} \cap B_{sub}) \geq \theta_1$ and $\frac{area(B_{obj})}{area(B_{sub})} > \theta_2$, i.e. they must have a certain overlap fraction $\theta_1$ and size ratio $\theta_2$. In DocParser, thresholds of $\theta_1 = 0.45$ and $\theta_2 = 1.2$ are used.

- **(h2): Grammar Check** This heuristic validates the candidate list against a predefined document grammar (see document grammar in the supplements). Concretely, all illegal candidates, e.g., a TABULAR nested inside a FIGURE, are removed.

- **(h3): Direct children** The candidate list is further pruned so that it contains only direct children of the parent and not sub-children. For this purpose, all sub-children are removed. As an example, this should remove $(E_{sub}^1, E_{obj}^3)$ from a candidate list $\{ (E_{sub}^1, E_{obj}^2), (E_{sub}^1, E_{obj}^3), (E_{sub}^2, E_{obj}^3) \}$, since it represents a sub-child and not a direct child of $E_{sub}^1$.

- **(h4): Unique Parents** The candidate list is altered so that each entity has only a single parent. Formally, if an entity $E_{obj}$ has multiple candidate parents, we first compare the Intersection over Union (IoU) of the bounding boxes of all candidate parents with $E_{obj}$: $\text{IoU} = \frac{area(B_{sub} \cap B_{obj})}{area(B_{sub} \cup B_{obj})}$. We then keep the parent with the maximal IoU, while all others are removed. If two parents have the same IoU, we select the element with the highest confidence score $P_i$ as parent. If that value is also equal, we choose the entity with the largest bounding box.

**Relations with Ordering (followed_by):** The entities are ordered according to the general reading flow (i.e., from left to right). Here care is needed so that multi-column pages are processed correctly. For this, two heuristics $o_1$ and $o_2$ are used. By default, all entities are processed by both heuristics. Children of floating entities are only processed by heuristic $o_2$, however.

- **(o1): Page Layout Entities** First, all entities are grouped according to their coordinates on the document page, namely, into groups belonging to the (a) left side $G_l$, (b) center $G_c$, or (c) right side $G_r$. Formally, this is achieved by computing the overlap for each entity $E_j, j = 1, \ldots, m$ with the left (and right) side of a document page, i.e. $\tau_{ovlp} = \text{overlap/width}(B)$. If the overlap with either the left (or the right) side is above a threshold (i.e. $\tau_{ovlp} > 0.7$), the entity $E_j$ is assigned to the left (or right) side.

  Otherwise, if such assignment is not possible with high confidence, the entity $E_j$ is assigned to center group $G_c$. In essence, the center group is an indicator whether the document is in single- or multi-column.

  If no entities have been assigned to the center group (i.e., $G_c = \emptyset$), then the entities are ordered first according to $G_l$ followed by $G_r$. Within each group, the entities are ordered top-to-bottom and then left-to-right by applying heuristic $o_2$. In sum, this approach should find an appropriate ordering for multi-column pages. If entities have been assigned to the center group (i.e., $G_c \neq \emptyset$), then grouping is further decomposed into additional subgroups: the entities $E \in G_c'$ from the center group are used to split $G_l, G_c, G_r$ into vertical subgroups $G_l', G_c', G_r'$, respectively. Afterward, we loop over all vertical subgroups $i$. For each, we order the entities according to the group (first $G_l'$, followed by $G_r'$, and then $G_c'$). Within each subgroup, we perform the ordering via heuristic $o_2$. This approach should correctly arrange entities in two cases: (1) in single-column pages and (2) when multi-column pages are split into different chunks by full-width figures or tables.

  For each subgroup, we perform the ordering via heuristic $o_2$.
(o2: Reading Flow) The entities $E_j$, $j = 1, \ldots, m$, are ordered top-to-bottom and, within lines, left-to-right, so that it matches the usual reading flow in documents. Formally, let the top-left corner of a document image refer to the coordinate $(0, 0)$. Furthermore, let us consider the top-left location of all bounding boxes $B_j$. The top-left location is then used to sort the entities first by their $y$-coordinate of $B_j$ and, if equal, by their $x$-coordinate (both ascending).

Component 4: Structure-Based Refinement We utilize the classified relations to iteratively refine entities and relations in four steps when parsing full document pages:

1. For each entity $E_{\text{parent}}$ with $l$ child entities $E_{1}^{\text{child}}, \ldots, E_{l}^{\text{child}}$, we update its bounding box such that $B_{\text{parent}} = \text{union}(B_{\text{parent}}, B_{1}^{\text{child}}, \ldots, B_{l}^{\text{child}})$.  
   2. For parent entities $E_{\text{parent}}$ with exactly one child entity of the same category, we remove the child entity and update $B_{\text{parent}}$ such that it is the union of parent and child bounding boxes. We also consider entity pairs of categories that do not conform to the document grammar. This allows us to discard duplicate entities of any category.  
   3. If an entity $E_{\text{child}}$ is sibling to other entities in a way that conflicts the document grammar, we generate a new entity that encloses $E_{\text{child}}$ to achieve conformity with the document grammar. Notably, nested FIGURE structures are defined such that one FIGURE should at most contain one FIGURE GRAPHIC entity child. If multiple FIGURE GRAPHIC are classified as children, we wrap each of them individually into new FIGURE entities.  
   4. If no parent is found for an entity $E_{\text{child}}$ that should only occur as a child entity, we identify a suitable parent entity by analyzing its neighboring siblings as follows: we consider all entities that jointly appear in an ordering relation with $E_{\text{child}}$ as candidates $E_{\text{cand}}$. We dismiss candidates of category $C$ that would not conform to the hierarchies defined in the document grammar. Finally, we dismiss any candidate for which $B_{\text{cand}} \cap B_{\text{child}} = \emptyset$. If exactly one candidate remains, we update its bounding box $B_{\text{cand}} = \text{union}(B_{\text{cand}}, B_{\text{child}})$.

The updates to the set of entities can lead to further changes to the classified relations. For this reason, whenever changes are made to entities in one of the four refinement steps, we update the relations via Component 3 and move back to refinement step (1). The refinement is completed once no change is applied in any of the steps or a limit of $r$ loop iterations has been reached.

Component 5: Scalable Weak Supervision

The system is further extended by scalable weak supervision. This aims at improving the performance of entity detection and, as a consequence, of end-to-end parsing.

Our weak supervision builds upon an additional dataset that consists of source codes (rather than document renderings). The source codes allow us to create a mapping between entities in the source code and their renderings. This process has three particular characteristics: first, the mapping is noisy and thus creates only weak labels. Despite that, the weak labels can aid efficient learning. Second, annotations are obtained only for some entities and relations. Third, if automated, this process circumvents human annotations and is thus highly scalable.

Let the unlabeled entities found in the source code be given by $S_{1}, \ldots, S_{k}$. For them, we generate weak labels $W_{1}, \ldots, W_{k}$ consisting of a semantic category and coordinates of the bounding box. However, both the semantic category and the bounding box can be subject to noise. Furthermore, weak labels are generated merely for a subset $C^{*} \subseteq C$ of the semantic categories.

In DocParser, the weak supervision is based on \TeX source files that are used to generate document renderings in the form of PDF files. The mapping between both formats is then obtained via synctex (Laurens 2008). synctex is a synchronization tool that performs a reverse rendering, so that PDF locations are mapped to \TeX code. For given coordinates in the document rendering, synctex returns a list of rectangular bounding boxes and the corresponding source code. Notably, the inference bounding boxes represent noisy labels, since the resulting entity annotations could be wrongly labeled, shifted, or entirely missing.

We proceed as follows. We iterate through the source code and retrieve bounding boxes for all \TeX commands. We then map the source code to our entities $E$. For instance, the bounding box for \TeX code $\begin{itemize} \item \text{figure} \item \end{itemize}$ environment is mapped onto a FIGURE GRAPHIC entity that is nested inside a FIGURE entity. Bounding boxes for all entities that act as inner children are created dynamically by computing the union bounding of all child bounding boxes.

We perform following processing steps to generate noisy labels for weak supervision:

1. Bounding boxes that are retrieved for simple text tokens inside the source code are mapped to CONTENT LINE entities.
2. If we encounter \TeX environments or \TeX commands (e. g., $\begin{itemize}$ or $\item$), we create corresponding candidate entities. All entities retrieved for tokens inside the scope of these environments are created as nested child entities. This approach is used to create the following entity types, namely FIGURE, FIGURE GRAPHIC, FIGURE CAPTION, TABLE, TABULAR, TABLE CAPTION, ITEMIZE, ITEM, ABSTRACT, and BIBLIOGRAPHY. Any other entities are mapped onto the CONTENT LINE category.
3. We utilize a special characteristic of synctex to identify EQUATION, EQUATION FORMULA and EQUATION LABEL entities: bounding boxes returned by synctex are highly uniform and typically consist of per-line bounding boxes of consistent width and $x$-coordinates. Equations and labels are an exception to this rule and typically only consist of vertically aligned bounding boxes of smaller width.
4. The sectioning structure of documents is considered: any type of \TeX command is mapped to a SECTION entity. The argument of the sectioning command, e. g. $\text{subsection}[\text{titlearg}]$, is mapped via synctex to a HEADER entity. Entities generated from code in the scope of a section are created as children to the section entity that corresponds to the current section scope.

---

\footnote{Details on our parameter choice and pseudocode are included in the supplements.}
5. Within sections, we sort entities based on a top-to-bottom, left-to-right reading order. Using these sorted lists of sibling entities, we form CONTENT BLOCK entities from subsequent groups of CONTENT LINE entities within page columns. If such block occurs within a BIBLIOGRAPHY environment, we instead map it to a BIBLIOGRAPHY BLOCK entity.

6. In TABLE environments, we consider all child entities (except captions) that do not span across a whole table width as CELL and the remainder as TABLE ROW. As we shall see later, this is effective at retrieving complex table structures.

7. We use the detected table cells to generate rows and columns as follows: We compute the centroids of all cells. To identify rows, we consider the sorted y-coordinates of the centroids and group them such that the pixel-wise distance between two consecutive y-coordinates in a group is smaller or equal to 5. If any identified group contains two or more centroid y-coordinates, we create a TABLE ROW entity from the union of the corresponding TABLE CELL entities. Analogously, using the x-coordinates of the cell centroids, we identify TABLE COLUMN entities.

8. Additional cleaning steps are performed for tables and figures: Child entities with width or height of 2 or fewer pixels are discarded. Caption bounding boxes that enclose other non-caption child entities are also discarded.

9. We make sure that entities contain at most one leaf node by moving excess leaves into newly generated CONTENT LINE entities.

10. We remove duplicate bounding boxes and entities without any leaf nodes in their respective sub-tree. Candidates are filtered such that only a group of entities and their respective sub-tree are preserved: ITEMIZE, FIGURE, TABLE, EQUATION, HEADING, CONTENT BLOCK, BIBLIOGRAPHY, ABSTRACT.

During training, entities with obvious errors are dismissed, i.e. leaf nodes or entities with bounding boxes that extend beyond page limits or with area of 0.

3 Datasets with Document Structures

We contribute the dataset “arXivdocs” that is tailored to the task of hierarchical structure parsing. It comes in two variants: arXivdocs-target and arXivdocs-weak.

(1) arXivdocs-target contains documents that have been manually checked and annotated. (2) arXivdocs-weak contains a large-scale set of documents that have no manual annotations but that can be used for weak supervision.

3.1 arXivdocs-target

arXivdocs-target provides a set of documents with manual annotations of the complete document structure. These documents were randomly selected from arXiv as an open repository of scientific articles, but in a way such that each has at most 30 pages and contains at least one TABLE within the source code. Altogether, it counts 362 documents. arXivdocs-target comes with predefined splits for training, validation, and eval that consist of 160, 79, 123 documents, respectively. The dataset comprises of 30 different entity categories. We ensure a fairly uniform distribution of entity categories across different splits by sampling one random page rendering for each of the 362 documents that contain an ABSTRACT, FIGURE, or TABLE. On average, each document contains 86.32 entities. The number of leaf nodes in the document graph as well as the frequency and average depth of the different entities are reported in the supplements.

Evidently, the most common category in the dataset is CONTENT LINE (34.33 %). This is because they typically represent leaf nodes in the graph and are children of larger entities such as ABSTRACT, CAPTION, or CONTENT BLOCK.

Annotators were instructed to follow the document grammar during labeling. Annotation of disallowed hierarchies is, however, possible to provide them the freedom to deal with the range of different document representations. Document annotations are automatically initialized by our scalable weak supervision mechanism to speed up the annotation process. The labelers were instructed to annotate entities only up to the coarseness that is used by DocParser, e.g. labeling content blocks, rather than individual lines.

3.2 arXivdocs-weak

arXivdocs-weak contains 127,472 documents with an average length of 12.84 pages that were retrieved from arXiv. We selected only documents that have a length of at most 30 pages and contain at least one TABLE within their source code. For reproducibility, we make our weak labels available.

4 Computational Setup

4.1 Mask R-CNN

Mask R-CNN extends the architecture of a convolution neural network with skip connections (He et al. 2016) so that it is highly effective for image segmentation and entity detection. Formally, it comprises of multiple stages with decreasing spatial resolution. The output of these stages is then fed into a so-called feature pyramid network (FPN) (Lin et al. 2017). The FPN then interconnects these inputs in multiple stages of increasing spatial resolution to produce multi-scale feature maps. Specifically, we use a ResNet-110 architecture (He et al. 2016) to extract features in 5 stages at different resolutions. The outputs of stages 2 to 5, denoted as $C_2, \ldots, C_5$, are passed to the FPN. The FPN outputs a total of 5 feature maps $P_2, \ldots, P_5$ at different resolutions. We refer the reader to (Lin et al. 2017) for a detailed description of the five feature maps. The multi-scale feature maps are then input to different prediction networks: first, a region proposal network (RPN) generates a list of candidate
bounding boxes that should contain an entity. Second, a Region of Interest (RoI) alignment layer filters out the multi-scale feature maps that correspond to the candidate regions. We note that all 5 feature maps are used by the RPN, but \( P_0 \) is not included in the inputs to the RoI alignment layer. Third, for each region proposal, a mask sub-network predicts the segmentation masks, based on the RoI aligned features. These segmentation masks are not used in subsequent steps of DocParser at prediction time; however, they are utilized in our loss function during the training process. Fourth, these bounding boxes are subsequently refined in a detection sub-network, thereby yielding the final bounding boxes \( B \). It also provides the label for classifying the entity category.

All of the above sub-networks were carefully adapted to the specific characteristics of our task: (1) We modified the region proposal network so that it uses a maximum base aspect ratio of 1:8 per entity. The reason for this modification is that document entities (as opposed to classical image segmentation) contain entities that have highly rectangular shapes. This is the case for most entities, e.g., single CONTENT LINE or TABLE ROW entities. (2) The output size of the classifier sub-network is modified so that it can produce predictions for entities across all semantic categories \( C \). (3) During training of the mask sub-network, we treat all pixels in ground truth bounding boxes as foreground. We do this to incorporate our understanding of the exact shape of many entities that span very wide rectangular regions. (4) We use a mask sub-network loss with a weighting factor of 0.5. This is to prioritize that features relevant for the correct prediction of bounding boxes and entity categories are learned. The Mask R-CNN stage of DocParser comprises 63,891,032 parameters and is built upon the implementation of Mask R-CNN provided by Abdulla (2017), yet which we carefully adapted as described above.

**Training Procedure:** All neural models are initialized with pre-trained weights based on the MS COCO dataset (Lin et al. 2014). We then train each model across three phases for a total of 80,000 iterations. This is split into three phases of 20,000, 40,000, and 20,000 iterations, respectively. During the first phase, we freeze all layers of the CNN that is used as the initial block in Mask R-CNN. In the second phase, stages four and five of the CNN are unfrozen. In the last phase, all network layers are trainable. Early stopping is applied based on the performance on the validation set for unrefined predictions. The performance is measured every 2000 iterations via the so-called intersection over union with a threshold of 0.8.

We train all models in a multi-GPU setting, using 8 GPUs with 8 VRAM of 12 GB. Each GPU was fed with one image per training iteration. Accordingly, the batch size per training iteration is set to 8. Furthermore, we use stochastic gradient descent with a learning rate of 0.001 and learning momentum of 0.9.

**Parameter Settings:** During training, we sampled randomly 100 entities from the ground truth per document image (i.e., up to 100 entities as some document images might have fewer). In Mask R-CNN, the maximum number of entity predictions per image is set to 200. During prediction, we only keep entities with a confidence score \( P_j \) of 0.7 or higher.

**Weak Supervision:** Training with weak supervision is as follows: all models are initialized with the weights of our pre-trained DocParser WS instead of default weights. We perform the training with learnable parameters analogous to phase 1 above but for 2000 steps with early stopping. In our experiments, we use only a subset of 80% of the annotated documents from arXivdocs-weak, while the other 20% remain unused. The intention is that we want to allow for additional annotations in the future while ensuring comparability to our results. We further ensure a fairly uniform distribution of entities by utilizing only document pages that contain at least an ABSTRACT, a FIGURE, or TABLE, while all others are discarded. This amounts to 593,583 pages.

### 4.2 System Variants

We compare the following variants of DocParser: DocParser Baseline is trained solely on the noise-free labels provided for the training dataset (here: arXivdocs-target); DocParser WS benefits from weak supervision (WS). It is trained based on a second dataset (here: arXivdocs-weak) with noisy labels for weak supervision. This is to test whether training systems on noisy labels can lead to higher performance, compared to training on small but noise-free training datasets; DocParser WS+FT is initialized with the weights from DocParser WS, but then fine-tuned (FT) on the target dataset.

### 4.3 Performance Metrics

We separately evaluate the performance of our system for (i) detection of entities \( E_j \) and (ii) classification of hierarchical relations \( R_j \). The former aims at a high detection rate (i.e., recognizing true positives out of all positives). Hence, we use the average precision as evaluation metric. The latter is based on the F1 score as it represents a typical classification task (i.e., recognizing one of the relations from \( \Psi \)).

**Entity Detection:** entity detection is commonly measured by the mean average precision (mAP) of a model (0: worst, 100: best). The inferred entities \( E_j = (c_j, B_j, P_j) \) are compared against the ground truth label consisting of the true category \( c_j \) with a bounding box \( B_j \). Here we follow common practice in computer vision (Everingham et al. 2010) and measure the overlap between bounding boxes from the same category. Specifically, we calculate the so-called intersection over union (IoU): \( \text{IoU} = \frac{\text{area}(B_j \cap B_i)}{\text{area}(B_j \cup B_i)} \). If the IoU is higher than a user-defined threshold, a predicted entity is considered a true positive. If multiple entities are matched with the same ground truth entity, we only consider the entity with the highest IoU as a true positive. Unmatched predictions and ground truth entities are considered false positives and false negatives, respectively. This is then used to calculate the average precision (AP) per semantic category \( C_k \in C \). The overall performance across all categories is given by the mean average precision. We compare IoU thresholds of 0.5 and 0.65.\(^8\)

\(^8\)Additional results for IoU=0.8 are in the supplements.
Prediction of Hierarchical Relations: Here we measure the classification performance for predicting the correct relations. A relation $R = (E_{\text{subj}}, E_{\text{obj}}, \Psi)$ is counted as correct only if the complete tuple is identical. However, the performance depends on the correct entity detection as input. Hence, we later vary the IoU thresholds for entity detection analogous to above and then report the corresponding F1 score for correctly predicting hierarchical relations. The F1 score is the harmonic average of precision and recall for predicting these triples (0: worst, 1: best).

Note that our performance measure is relatively strict. We show that, even if some F1 scores are in a lower range, we can recover the overall document structure successfully. In particular, we outperform state-of-the-art OCR results, as illustrated in the qualitative samples in our supplements.

### 4.4 Robustness Check: Table Structure Parsing

We additionally train our model for structure parsing so that it identifies table structures to demonstrate the robustness of our system and weak supervision.

We confirm the effectiveness of our weak supervision as follows: we draw upon the ICDAR 2013 dataset (Gobel et al. 2013) for table structure parsing and compare it with the state-of-the-art. The ICDAR 2013 dataset consists of a variety of real-world documents and is not limited to scientific articles. We proceed analogously to full document structure parsing and train the three system variants for the task of table structure recognition.

**DocParser Baseline** is trained solely on the samples provided in the ICDAR 2013 training dataset; **DocParser WS** is trained on table structures generated from arXivdocs-weak. **DocParser WS+FT** is generated by subsequent fine-tuning on the ICDAR training split.9

Both training and fine-tuning of all variants follow the 3 phase training scheme for a total of 80,000 iterations.10

### 5 Results

The key focus of our experiments is to confirm the effectiveness of DocParser for parsing the complete document structures. However, we emphasize again that both suitable baselines and datasets for this task are hitherto lacking. Hence, we proceed two-fold. On the one hand, we evaluate the performance based on arXivdocs as the first dataset for document structure parsing. On the other hand, we draw upon the table structure ICDAR 2013 dataset: it is limited to table structures and not complete holistic parsing of document structures. However, it allows to test the effectiveness of our weak supervision against state-of-the-art.

---

9Details about the setting and additional experiments are provided in the supplements.

10Due to the different domain of the target dataset, we experimented with other weak supervision strategies, e.g. randomly sampling images from arXivdocs-weak and ICDAR 2013 during the same training procedure. However, the performance of models trained by sequential fine-tuning could not be surpassed.

<table>
<thead>
<tr>
<th>IoU=0.5</th>
<th>IoU=0.65</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP</td>
<td>Baseline WS</td>
</tr>
<tr>
<td>mean AP</td>
<td>49.9</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>95.2</td>
</tr>
<tr>
<td>AFFILIATION</td>
<td>51.6</td>
</tr>
<tr>
<td>AUTHOR</td>
<td>18.0</td>
</tr>
<tr>
<td>BIB. BLOCK</td>
<td>42.4</td>
</tr>
<tr>
<td>CONT. BLOCK</td>
<td>89.3</td>
</tr>
<tr>
<td>DATE</td>
<td>0.0</td>
</tr>
<tr>
<td>EQUATION</td>
<td>65.8</td>
</tr>
<tr>
<td>FIG. CAPTION</td>
<td>47.8</td>
</tr>
<tr>
<td>FIG. GRAPHIC</td>
<td>22.3</td>
</tr>
<tr>
<td>FIGURE</td>
<td>47.8</td>
</tr>
<tr>
<td>FOOTER</td>
<td>55.7</td>
</tr>
<tr>
<td>HEADER</td>
<td>79.7</td>
</tr>
<tr>
<td>HEADING</td>
<td>53.7</td>
</tr>
<tr>
<td>ITEM</td>
<td>0.0</td>
</tr>
<tr>
<td>ITEMIZE</td>
<td>0.0</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>36.4</td>
</tr>
<tr>
<td>PAGE NR</td>
<td>74.7</td>
</tr>
<tr>
<td>TAB. CAPTION</td>
<td>55.2</td>
</tr>
<tr>
<td>TABLE</td>
<td>84.5</td>
</tr>
<tr>
<td>TABULAR</td>
<td>78.4</td>
</tr>
</tbody>
</table>

Table 1: Average precision (AP) of entity detection.

### 5.1 Document Structure Parsing

We compare the performance of document structure parsing based on our arXivdocs-target dataset across both performance metrics.

**Entity Detection** The overall performance for entity detection is detailed in Table 1 (first row). We discuss the performance for IoU = 0.5 in the following. DocParser Baseline achieves an mAP of 49.9. This is higher than DocParser WS with an mAP of 34.6. We attribute this to the fact that several entity categories from arXivdocs-target are not part of arXivdocs-weak. Notably, the fine-tuned system DocParser WS+FT results in significant performance improvements: it obtains a mAP of 69.4, which, in comparison to the baseline DocParser, is an improvement by 39.1%.

DocParser WS+FT consistently outperforms the baseline system, even for categories that are not annotated during weak supervision (e.g. AUTHOR, FOOTER, HEADER, PAGE NUMBER). We attribute this to the better model initialization due to the prior weakly supervised pre-training. There is a small number of entity categories for which the Baseline achieves higher AP values. We attribute this to our experimental protocol which yields the best model via early stopping, based on mAP and not on individual entity AP values. For a few entities a decrease can be observed after fine-tuning (e.g. TABLE at IoU=0.5). We attribute this to the high quality of weak annotations for this category and, consequently, a slight decrease of generalization due to fine-tuning. Some AP values (for both DocParser Baseline and DocParser WS) amount to 0.0, e.g. for DATE. This is caused by the absence of some categories in arXivdocs-weak in the case of DocParser WS. For DocParser Baseline, we attribute this to the limited amount of samples in arXivdocs-target for
Figure 3: Performance of entity detection (mAP for IoU = 0.5) during fine-tuning.

<table>
<thead>
<tr>
<th></th>
<th>IoU=0.5</th>
<th>IoU=0.65</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Baseline</td>
<td>WS</td>
</tr>
<tr>
<td>All</td>
<td>0.416</td>
<td>0.343</td>
</tr>
<tr>
<td>followed_by</td>
<td>0.413</td>
<td>0.387</td>
</tr>
<tr>
<td>parent_of</td>
<td>0.421</td>
<td>0.235</td>
</tr>
<tr>
<td>Refined:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>All</td>
<td>0.453</td>
<td>0.382</td>
</tr>
<tr>
<td>followed_by</td>
<td>0.455</td>
<td>0.410</td>
</tr>
<tr>
<td>parent_of</td>
<td>0.451</td>
<td>0.317</td>
</tr>
</tbody>
</table>

Table 2: Performance in predicting hierarchical relations (as measured by F1).

The affected categories, coupled with an inferior model initialization, compared to DocParser WS+FT. DocParser WS+FT outperforms the DocParser Baseline system across all measured IoU thresholds by a considerable margin. Using IoU thresholds above 0.5 leads to a performance decrease. Even though higher IoUs should generally correspond to better matches with the ground truth, they can penalize ambiguous cases and thus a correct detection. In sum, this confirms the effectiveness of our weak supervision in bolstering the overall performance.

Table 1 breaks down the performance by entity category. For DocParser WS+FT, we observe an especially good performance for detecting tabulars and figures. This is owed to the strong initialization of our system due to the high quality and large number of samples in our scalable weak supervision.

Figure 3 shows the fine-tuning. Only 20 fine-tuning samples are sufficient for DocParser WS+FT to surpass the baseline system DocParser (which is trained on 160 samples from the target dataset). It thus helps in reducing the labeling effort by a factor of around 8. Furthermore, we observe a steady increase in the performance of the fine-tuned networks with more samples. Notably, the highest performance increase is already achieved by the first 10 document images for fine-tuning.

For a few entities, the best performance is achieved a combination of the WS system together with a high IoU (e.g., BIBLIOGRAPHY BLOCK). A likely reason for this is the composition of arXivdocs-target. As BIBLIOGRAPHY entities were not specifically used as a criterion for the per-page sampling, fewer documents in the target dataset contained relevant entities, leading to decreased performance of the baseline and WS+FT systems.

5.2 Robustness Check: Table Structure Parsing

Results: Table 3 compares the state-of-the-art for table structure parsing with our weak supervision strategy. Altogether, our weak supervision outperforms the state-of-the-art (Schreiber et al. 2018) by a considerable margin.

Discussion: Our system shows significant improvement over the image-based state of the art. We also compare our approach to the state-of-the-art heuristic-based system that operates on raw PDF files, instead of images, as input (Nurminen 2013). Even though our system does not utilize the additional information provided by raw PDF files, DocParser achieves an F1 score of 0.9292, compared to 0.9221 for the PDF-based system. We refrain from directly comparing the aforementioned F1 score with that from earlier experiments as the underlying target domains differ.

6 Related Work

OCR: Extracting text from document images has been extensively studied as part of optical character recognition (OCR) within the NLP community (e.g., Schäfer et al. 2011; Schäfer and Weitz 2012). To this end, the work by Katti et al. (2018) argued that OCR should be seen as a preprocessing step for downstream NLP tasks. As such, the au-
There are works that recognize table structures from text or other syntactic tokens (Kieninger and Dengel 1998; Pivk et al. 2007) rather than directly from document renderings. As such, these works are tailored to tokens as input, and it is thus unclear how such an approach could theoretically be adapted to document renderings since our task inherently relies upon images as input. Because of the different input and thus the different datasets for benchmarking, the performance of the aforementioned works is not comparable to our approach. The works by Schreiber et al. (2018); Qasim, Mahmood, and Shafait (2019) draw upon deep neural networks to identify table structures for rendered inputs. However, they aim at a different purpose: parsing table structures, but not complete document hierarchies. As such, the authors do not attempt to identify text elements, nested figures, etc.

**Weak Supervision for Document Layout:** (Zhong, Tang, and Yepes 2019) use weak supervision for detection of page layout entities. The WS mechanism relies on matching external XML annotations with text extractions by a heuristic-based third-party tool. In contrast, our weak supervision directly builds on the LATEX compilation and can be readily extended to any new dataset of LATEX source files. Furthermore, the dataset features only 5 coarse categories and the system does not feature a relation classification component, thus being insufficient to acquire full document structures.\(^\text{12}\)

**Weak Supervision in NLP:** Annotations in NLP are oftentimes costly and, as a result, there has been a recent surge in weak supervision. Weak supervision has now been applied to various tasks, such as text classification (e.g., Hingmire and Chakraborti 2014; Lin, He, and Everson Richard 2011), information extraction (e.g., Hoffmann et al. 2011), and semantic parsing (e.g., Goldman et al. 2018). The methodological levers for obtaining weak labels are versatile and include, e.g., manual rules (e.g., Rabinovich et al. 2018), estimated models (e.g., Hoffmann et al. 2011), or reinforcement learning (Pröllchs, Feuerriegel, and Neumann 2019); however, not for document structure parsing.

7 Discussion and Conclusion

**Efficiency:** Our system requires only \(~340\) ms/document during entity detection (averaged over our validation set of 79 documents for DocParser WS+FT) on a single Titan Xp GPU with 12 GB VRAM and a batch size of 1. The relation detection in stage 2 only adds a minimal overhead of an average of 5.67 ms/document (10.81 ms/document with refinement) on a single CPU @ 2.1 GHz.

**Qualitative Assessment:** We performed a qualitative analysis on a subset of documents. We observe that, even for F1 scores below 0.5, the final document structure is often still very accurate. In fact, state-of-the-art OCR systems as natural baselines are outperformed significantly. This can be explained by our experiment design: we used very strict evaluation metrics. Hence, even small mismatches or ambiguities between the ground truth and predicted entities result in fairly large F1 penalties, despite high overall similarity. Details are in the supplements (including qualitative examples).

**Detection Model Choice:** Deep CNN models, including recent work (Tan and Le 2019; Duan et al. 2019), are heavily reliant on large training datasets. As such, we expect the impact of our technical contribution, as shown in our comparison of baseline and WS+FT models, to be the same across different modern CNN backbones. Our choice of Mask R-CNN as a tool for instance segmentation was also done in consideration of possible future extensions of DocParser to non-rectified documents. Here, the additional instance masks could guide the OCR or rectification process.

**Future Work:** In future work, we plan to explore approaches that can jointly learn entity and relation detection. Furthermore, we aim to further improve our system by enriching 2D inputs with textual features, e.g. high-dimensional word embeddings. The robustness of WS pre-training w.r.t. smaller subsets of arXivdocs-weak is another area of future investigation.

**Conclusion:** Despite the extensive interest of the NLP community in leveraging document structures (e.g., Apostolova and Tomuro 2014; Schäfer et al. 2011; Schäfer and Weitz 2012; Schreiber et al. 2018; Katti et al. 2018), the task of parsing complete document structures from renderings has been overlooked. To the best of our knowledge, we present the first system for this task. In particular, DocParser provides an effective alternative to state-of-the-art OCR which is still widespread in practice. In addition, DocParser allows to provide additional semantic input to downstream NLP tasks (e.g. information extraction).

Acknowledgments

Ce Zhang and the DS3Lab gratefully acknowledge the support from the Swiss National Science Foundation (Project Number 200021_184628), Innosuisse/SNF BRIDGE Discovery (Project Number 40B2-0_187132), European Union Horizon 2020 Research and Innovation Programme (DAPHNE, 957407), Botnar Research Centre for Child Health, Swiss Data Science Center, Alibaba, Cisco, eBay, Google Focused Research Awards, Oracle Labs, Swisscom, Zurich Insurance, Chinese Scholarship Council, and the Department of Computer Science at ETH Zurich.
References


Hingmire, S.; and Chakraborti, S. 2014. Sprinkling Topics For Weakly Supervised Text Classification. In Annual Meeting of the ACL.


Pivk, A.; Cimiano, P.; Sure, Y.; Gams, M.; Rajković, V.; and Studer, R. 2007. Transforming Arbitrary Tables into Logical Form with TARTAR. *Data and Knowledge Engineering* 567–595. ISSN 0169023X.


Yildiz, B.; Kaiser, K.; and Miksch, S. 2005. *pdf2table: A Method to Extract Table Information from PDF Files*. 2nd *Indian International Conference on Artificial Intelligence (IICAI)*.