PaperMage: A Unified Toolkit for Processing, Representing, and Manipulating Visually-Rich Scientific Documents

Kyle Lo[α*] **Zejiang Shen**[α,τ*] **Benjamin Newman**[α*] **Joseph Chee Chang**[α*]
Russell Authur[α] **Erin Bransom**[α] **Stefan Candra**[α] **Yoganand Chandrasekhar**[α]
Regan Huff[α] **Bailey Kuehl**[α] **Amanpreet Singh**[α] **Chris Wilhelm**[α] **Angele Zamarron**[α]
Marti A. Hearst[β] **Daniel S. Weld**[α,ω] **Doug Downey**[α,η] **Luca Soldaini**[α*]

[α]Allen Institute for AI [τ]Massachusetts Institute of Technology
[β]University of California Berkeley [ω]University of Washington [η]Northwestern University

`{kylel, lucas}@allenai.org`

Abstract

Despite growing interest in applying natural language processing models to the scholarly domain, scientific documents remain challenging to work with. They're often in difficult-to-use PDF formats, and the ecosystem of models to process them is fragmented and incomplete. We introduce `papermage`, an open-source Python toolkit for processing and analyzing the contents of visually-rich, structured scientific documents. `papermage` offers abstractions for seamlessly representing both textual and visual paper elements, integrates several disparate state-of-the-art models into a unified framework, and provides turn-key recipes for standard scientific NLP use-cases. `papermage` has powered multiple research prototypes along with a large-scale production system, processing millions of PDFs.

https://github.com/allenai/papermage

Tutorial: Video **License**: Apache 2.0

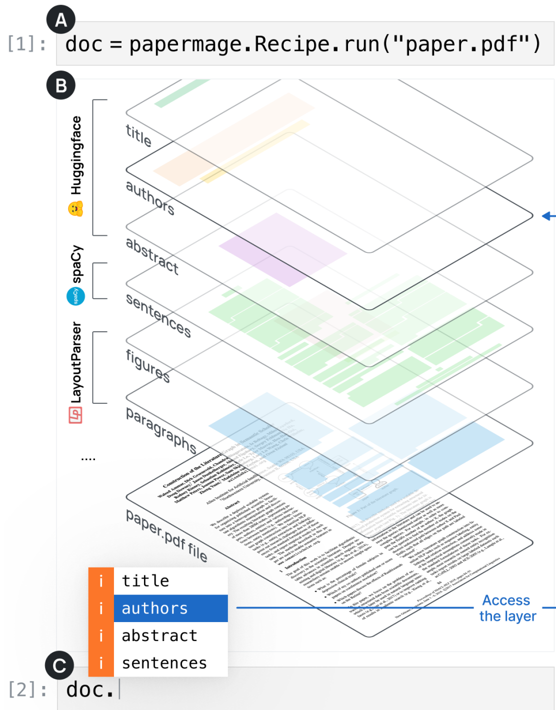

Figure 1: `papermage`'s document creation and representation. (A) Recipes are turn-key methods for processing a PDF. (B) They compose different models to extract document structure, which we conceptualize as layers of annotation that store textual and visual information. (C) Users can access the data in these layers in Python.

1 Introduction

Research papers and textbooks are central to the scientific enterprise, and there is increasing interest in developing new tools for extracting knowledge from these visually-rich documents. Recent research has explored, for example, AI-powered reading support for math symbol definitions (Head et al., 2021), in-situ passage explanations or summaries (August et al., 2023; Rachatasumrit et al., 2022), automatic span highlighting (Chang et al., 2023; Fok et al., 2023), interactive clipping and synthesis (Kang et al., 2022) and more. Further, extracting clean, properly-structured scientific text from PDF documents (Lo et al., 2020; Wang et al., 2020) forms a critical first step in pretraining language models of science (Beltagy et al., 2019; Lee et al., 2019; Gu et al., 2020; Luo et al., 2022; Taylor et al., 2022; Trewartha et al., 2022; Hong et al., 2023), automatic generation of more accessible paper formats (Wang et al., 2021), and developing datasets for scientific natural language processing (NLP) tasks over structured full text (Jain et al., 2020; Subramanian et al., 2020; Dasigi et al., 2021; Lee et al., 2023).

However, this type of NLP research on scientific corpora is difficult because the documents come in difficult-to-use formats like PDF,[1] and existing tools for working with the documents are limited.

[*]Core contributors.

[1]PDFs store text as character glyphs and their (x, y) positions on a page. Converting this data to usable text for NLP requires error-prone operations like inferring token boundaries, whitespacing, and reading order using visual positioning.

Typically, the first step in scientific document processing is to invoke a parser on a document file to convert it into a sequence of tokens and bounding boxes in inferred reading order. Parsers extract only the raw document content, and obtaining richer document structure (e.g., titles, authors, figures) or linguistic structure and semantics (e.g., sentences, discourse units, scientific claims) requires sending the token sequence through downstream models. Unlike more mature parsers (§2.1), these downstream models are often research prototypes (§2.2) that are limited to extracting only a subset of the structures needed for one's research (e.g., the same model may not provide both sentence splits and figure detection). As a result, users must write extensive custom code that strings pipelines of multiple models together. Research projects using models of different modalities (e.g., combining an image-based formula detector with a text-based definition extractor) can require hundreds of lines of code.

We introduce `papermage`, an open-source Python toolkit for processing scientific documents. Its contributions include (1) `magelib`, a library of data structures and built-in methods for representing and manipulating visually-rich documents as multi-modal constructs, (2) `Predictors`, a set of implementations that integrate different state-of-the-art scientific document analysis models into a unified interface, even if individual models are written in different frameworks or operate on different modalities, and (3) `Recipes`, which provide turn-key access to well-tested combinations of individual (often single-modality) modules to form sophisticated, extensible multi-modal pipelines.

2 Related Work

2.1 Turn-key software for scientific documents

Processing visually-rich documents like scientific papers requires a joint understanding of both visual and textual information. In practice, this often requires combining different models into complex processing pipelines. For example, GROBID (GRO, 2008–2023), a widely-adopted software tool for scientific document processing, uses twelve interdependent sequence labeling models[2] to perform its full text extraction. Other similar tools inlude CERMINE (Tkaczyk et al., 2015) and ParsCit (Councill et al., 2008). While such software is often an ideal choice for off-the-shelf processing, it is not necessarily designed for easy extension and/or integration with newer research models.[3]

2.2 Models for scientific document processing

While aforementioned software tools use CRF or BiLSTM-based models, Transformer-based models have seen wide adoption among NLP researchers for their powerful processing capabilities. Recent years have seen the rise of layout-infused Transformers (Xu et al., 2019; Shen et al., 2022; Xu et al., 2021; Huang et al., 2022b; Chen et al., 2023) for processing visually-rich documents, including recovering logical structure (e.g., titles, abstracts) of scientific papers (Huang et al., 2022a). Similarly, computer vision (CV) researchers have also shown impressive capabilities of CNN-based object detection models (Ren et al., 2015; Tan et al., 2020) for segmenting visually-rich documents based on their layout. While these research models are powerful and extensible for research purposes, it often requires significant "glue" code and combination with different software tools to combine into robust processing pipelines. One example is that of Lincker et al. (2023) who bootstrap a sophisticated processing pipeline around a research model for processing children's textbooks.

2.3 Combining models and pipelines

`papermage`'s use case lies between that of turn-key software and a framework for supporting research. Similar to `Transformers` (Wolfe et al., 2022)'s integration of different research models into standard interfaces, others have done similarly for the visually-rich document domain. `LayoutParser` (Shen et al., 2021) provides models for visually-rich documents and supports the creation of document processing pipelines. `papermage`, in fact, depends on `LayoutParser` for access to vision models, but is designed to also integrate text models which are omitted from `LayoutParser`. To allow models of different modalities to work well together, we also developed the `magelib` library (§3.1).

[2] https://grobid.readthedocs.io/en/latest/Training-the-models-of-Grobid/#models

[3] Most research in NLP requires a research have the ability to manipulate models within Python. Grobid, for example, requires users to manage a service process and send PDFs through a client. We also found in performing our evaluation in §3.3 that it is difficult to run just the model components isolated from PDF utilities, which makes comparison with research models challenging without significant "glue" code.

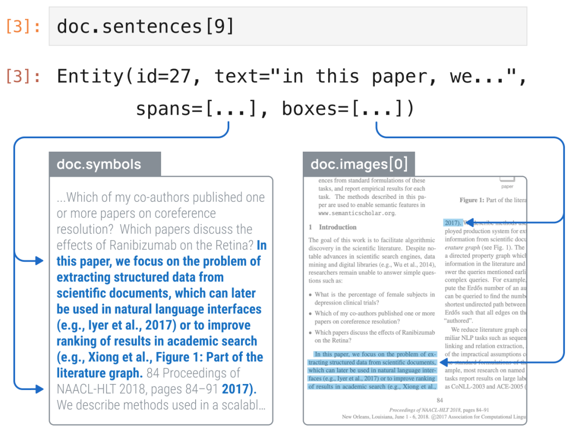

Figure 2: `Entities` are multimodal content units. Here, `spans` of a sentence are used to identify its text among all symbols, while `boxes` map its visual coordinates on a page. `spans` and `boxes` can include non-contiguous units, allowing great flexibility in `Entities`.

3 Design of `papermage`

`papermage` is comprised of three parts: (1) `magelib`, a library for intuitively representing and manipulating visually-rich documents, (2) `Predictors`, implementations of models for analyzing scientific papers that unify disparate machine learning frameworks under a common interface, and (3) `Recipes`, combinations of `Predictors` that form multi-modal pipelines.

3.1 Representing and manipulating visually-rich documents with `magelib`

In this section, we use code snippets to show how our library's abstractions and syntax are tailored for the visually-rich document problem domain.[4]

Data Classes. `magelib` provides three base data classes for representing fundamental elements of visually-rich, structured documents: `Document`, `Layers` and `Entities`. First, a `Document` minimally stores text as a string of symbols:

```
from papermage import Document
doc.symbols
>>> "Revolt: Collaborative Crowdsourcing..."
```

But visually-rich documents are more than a linearized string. For example, analyzing a scientific paper requires access to its visuospatial layout (e.g., pages, blocks, lines), logical structure (e.g., title, abstract, figures, tables, footnotes, sections), semantic units (e.g., paragraphs, sentences, tokens), and more (e.g., citations, terms). In practice, this means different parts of `doc.symbols` can correspond to different paragraphs, sentences, tokens, etc. in the `Document`, each with its own set of corresponding coordinates representing its visual position on a page.

`magelib` represents structure using `Layers` that can be accessed as attributes of a `Document` (e.g., `doc.sentences`, `doc.figures`, `doc.tokens`) (Figure 1). Each `Layer` is a list of content units, called `Entities`, which store both textual (e.g., spans, strings) and visuospatial (e.g., bounding boxes, pixel arrays) information. For example, let's consider representing "sentences" in a scientific paper. As seen in Figure 2, a sentence split across columns/pages and interrupted by floating figures/footnotes would require multiple spans and bounding boxes to represent.

Methods. `magelib` also provides a set of functions for building and interacting with the data classes: Annotating a `Document` with additional `Layers`, traversing and spatially searching for matching `Entities` in one `Layer`, and cross-referencing between `Layers` (see Figure 3).

For example, a `Document` that initially only contains the `doc.symbols` `Layer` can be annotated with additional `Layers` through:

```
paragraphs: List[Entity] = [...]
sentences: List[Entity] = [...]
tokens: List[Entity] = [...]

doc.annotate_entity("paragraphs", paragraphs)
doc.annotate_entity("sentences", sentences)
doc.annotate_entity("tokens", tokens)
```

Section §3.2 explains in more detail how a user can generate these `Entities`.

With the three new layers, users automatically gain the ability to iterate through the `Document` by each paragraph `Entity` in the `paragraphs` `Layer`, and cross-reference with its corresponding `Entities` in the `sentences` and `tokens` `Layers`:

```
for paragraph in doc.paragraphs:
    for sentence in paragraph.sentences:
        for token in sentence.tokens:
            ...
```

Finally, `magelib` also supports cross-modality operations. For example, searching for all `tokens` in a visual region on the PDF (See Figure 3 F):

```
query = Box(l=423, t=71, w=159, h=87, page=0)
selected_tokens = doc.find(query, "tokens")
[t.text for t in selected_tokens]
>>> ["Techniques", "for", "collecting", ...]
```

[4] Exact syntax shown here may differ from that in the code. Given software can evolve beyond the paper, we've opted to simplify syntax here, prioritizing legibility and design clarity.

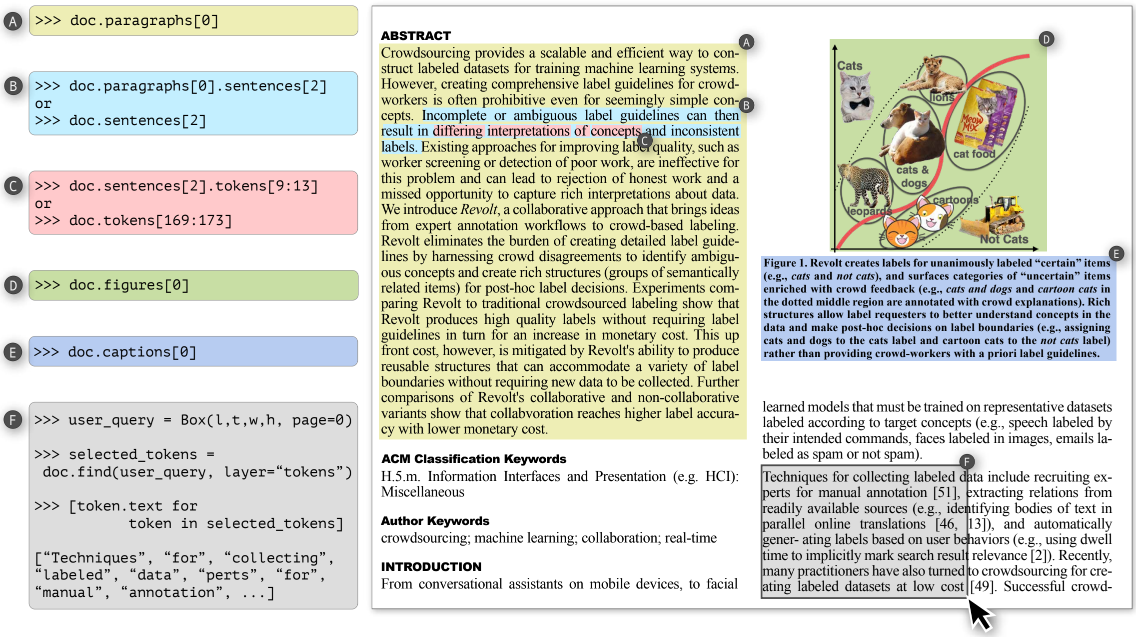

Figure 3: Illustrates how `Entities` can be accessed flexibly in different ways: (A) Accessing the `Entity` of the first paragraph in the `Document` via its own `Layer` (B) Accessing a sentence via the paragraph `Entity` or directly via the sentences `Layer` (C) Similarly, the same tokens can be accessed via the overlapping sentence `Entity` or directly via the tokens `Layer` of the `Document` (where the first tokens are the title of the paper.) (D, E) Figures, captions, tables and keywords can be accessed in similar ways (F) Additionally, given a bounding box (e.g., of a user selected region), `papermage` can find the corresponding `Entities` for a given `Layer`, in this case finding the tokens under the region. Excerpt from Chang et al. (2017).

Protocols and Utilities. For `Document` instantiation, `magelib` provides hooks into existing PDF processing tools called `Parsers` and `Rasterizers`:

```
import papermage as pm
parser = pm.PDF2TextParser()
doc = parser.parse("...pdf")
[token.text for token in doc.tokens]
>>> ["Revolt", ":", "Collaborative", ...]
doc.page[0].image
>>> None

rasterizer = pm.PDF2ImageRasterizer()
images = rasterizer.rasterize("...pdf")
rasterizer.attach_images(images, doc)
doc.page[0].image
>>> Image(np.array(...))
```

In the above example, `papermage` first runs the `PDF2TextParser` (using `pdfplumber`) to extract the textual information from a PDF file, and then runs the `PDF2ImageRasterizer` (using `pdf2image`) to attach an image to each of the corresponding `Entities` in the `Document` (i.e., pages). While `magelib` provides wrappers for these existing PDF processing tools out of the box, it can also be extended to support other PDF-processing tools.[5]

3.2 Interfacing with models for scientific document analysis through `Predictors`

In §3.1, we described how users could create custom `Layers` by annotating a set of `Entities` on a `Document`. But how would they make `Entities` in the first place?

For example, to identify multimodal structures in visually-rich documents, researchers might want to build complex pipelines that run and combine output from many different models (e.g., computer vision models for extracting figures, NLP models for classifying body text). `papermage` provides a unified interface, called `Predictors`, to ensure models produce `Entities` that are compatible with the `Document`.

`papermage` comes with numerous ready-to-use `Predictors` that leverage state-of-the-art models to extract specific document structures. While `magelib`'s abstractions are general for visually-rich documents, `Predictors` are optimized for parsing of scientific documents. They are designed to (1) be compatible with models from many different machine learning frameworks, (2) support inference

[5] PDFs are not the only way of representing visually-rich documents. For example, many scientific papers are distributed in XML format. We also provide `Parsers` that can instantiate a `Document` from XML input. See Appendix A.1.

Use-case	Description	Examples
Linguistic/ Semantic	Segments doc into text units often used for downstream models.	SentencePredictor wraps sciSpaCy (Neumann et al., 2019) and PySBD (Sadvilkar and Neumann, 2020) to segment sentences. WordPredictor is a custom scikit-learn model to identify broken words split across PDF lines or columns. ParagraphPredictor is a set of heuristics on top of both layout and logical structure models to extract paragraphs.
Layout Structure	Segments doc into visual block regions.	BoxPredictor wraps models from LayoutParser (Shen et al., 2021), which provides vision models like EfficientDet (Tan et al., 2020) pretrained on scientific layouts (Zhong et al., 2019).
Logical Structure	Segments doc into organizational units like title, abstract, body, footnotes, caption, and more.	SpanPredictor is compatible with Transformers (Wolfe et al., 2022) token classifiers. We support both pretrained weights from VILA (Shen et al., 2022), as well as RoBERTa (Liu et al., 2019), SciBERT (Beltagy et al., 2019) weights that we've finetuned on similar data and compiled using ONNX for production.
Task-specific	User-defined models for a given task can be used with papermage if wrapped as a Predictor following common patterns. See example in §4.	Many NLP researchers depend on prompting a model through an API call. We implement APIPredictor which interfaces external APIs, such as GPT-3 (Brown et al., 2020). E.g., we wrap the system in Newman et al. (2023) which takes a question *about* a paper snippet, retrieves within-paper evidence using Contriever (Izacard et al., 2022), and prompts different language models for answers.

Table 1: Types of Predictors implemented in papermage.

Model	Full			Grobid Subset		
	P	R	F1	P	R	F1
Grobid$_{CRF}$	40.6	38.3	39.1	81.2	76.7	78.9
Grobid$_{NN}$	42.0	36.5	37.6	84.1	73.0	78.2
RoBERTa	75.9	80.0	76.8	82.6	83.9	83.2
I-VILA	**92.0**	**94.1**	**92.7**	**92.2**	**95.2**	**93.7**

Table 2: Evaluating CoreRecipe for logical structure recovery on S2-VL (Shen et al., 2022). papermage supports interchangeable research models (e.g., text-only RoBERTa, layout-infused I-VILA(Shen et al., 2022)), as well as external APIs, such as calling Grobid. Metrics are computed for token-level classification, macro-averaged over categories. The "Grobid Subset" limits evaluation to only categories for which Grobid returns bounding box information, which was necessary for evaluation on S2-VL. See Appendix A.3 for details.

with text-only, vision-only, and multimodal models, and (3) support both adaptation of off-the-shelf, pretrained models as well as development of new ones from scratch. Similarly to the Transformers library, a Predictor's implementation is typically independent from its configuration, allowing users to customize each Predictor by tweaking hyperparameters or loading a different set of weights.

Below, we showcase how a vision model and two text models (both neural and symbolic) can be applied in succession to a single Document. See Table 1 for a summary of supported Predictors.

```
import papermage as pm
cv_predictor = pm.BoxPredictor(...)
output = cv_predictor.predict(doc)
tables, figures = pm.group_by(output, "label")
doc.annotate(tables, figures)

nlp_neu_predictor = pm.SpanPredictor(...)
output = nlp_neu_predictor.predict(doc)
titles, authors = pm.group_by(output, "label")
doc.annotate(titles, authors)

nlp_sym_predictor = pm.SentencePredictor(...)
output = nlp_sym_predictor.predict(doc)
sentences = pm.group_by(output, "label")
doc.annotate(sentences)
```

Predictors return a list of Entities, which can be group_by() to organize them based on predicted label value (e.g., tokens classified as "title" or "authors"). Finally, these predictions are passed to doc.annotate() to be added to Document.

3.3 End-to-end processing with Recipes

Finally, papermage provides predefined combinations of Predictors, called Recipes, for users seeking a high-quality option for turn-key processing of visually-rich documents:

```
from papermage import CoreRecipe
recipe = CoreRecipe()
doc = recipe.run("...pdf")
doc.captions[0].text
>>> "Figure 1. ..."
```

A brief performance comparison of potential modules for papermage main Recipe is reported in Table 2. We expect Recipes to appeal to two groups of users—end-to-end consumers, and developers of high-level applications. The former is comprised of developers and researchers who are looking for a one-step solution to multimodal scientific document analysis. The latter are likely developers and researchers looking to combine document structure primitives to build a complex application (see example in §4).

4 Vignette: Building an Attributed QA System for Scientific Papers

How could researchers leverage papermage for their research? Here, we walk through a user scenario in which a researcher (Lucy) is prototyping an attributed QA system for science.

System Design. Drawing inspiration from Ko et al. (2020) and Lee et al. (2023), Lucy is studying how language models can be used to resolve questions that arise while reading a paper (e.g. *What does this mean?* or *What does this refer to?*). In her prototype reading interface design, a user can highlight a passage that they are reading and ask a question about it. A retrieval model then finds relevant passages from the rest of the paper. The prototype then uses the *text* of the retrieved passages along with the user question to prompt a language model to generate an answer. When presenting the answer to the user, the prototype also *visually* highlights the retrieved passages as supporting evidence to the generated answer.

Getting started quickly. As a researcher proficient in Python, it only takes Lucy minutes to install papermage using pip and successfully process a local PDF file by following the example code snippet for CoreRecipe in §3.2. In an interactive session, she familiarizes herself with the provided Layers by following the traversal, cross-referencing and querying examples in §3.1. She makes sure she can serialize and re-instantiate her Document (§A.2).

Formatting input. Before using papermage, Lucy has prior experience building QA pipelines, but only dealt with text-only data (e.g., <str>) and document context (e.g., <List[str]>). Lucy realizes that she can reuse her prior code with papermage by implementing a couple of wrappers to gain additional capabilities: First, she converts a user's highlighted passage from a visual selection to text following the example in Figure 3 F. Next, she converts Document to her required text format by following the traversal examples in §3.1 (e.g., using `[s.text for s in doc.sentences]`). Within a few lines of code, Lucy has everything she needs for text-only input to her QA pipeline.

Formatting output. Lucy runs her QA system on her newly acquired text data and now has (1) a model-generated answer and (2) several retrieved evidence passages. She realizes that she already has access to the evidences' bounding boxes via a similar call to how she defined the model input context (e.g., `[s.boxes for s in doc.sentences]`). She can easily pass this to the user interface to enable linking to and highlighting of those passages.

Defining a `Predictor`. The pattern Lucy has followed is used in our many Predictor implementations: (1) gain access to text by traversing Layers (e.g., sentences), (2) perform all usual NLP computation on that text, and (3) format model output as Entities. This simple pattern allows users to reuse familiar models in existing frameworks and eschews lengthy onboarding to papermage. Lucy wraps her code in a new APISpanQAPredictor class. We've included an implementation of this Predictor as part of papermage (see Table 1).

Fast iterations. Leveraging the bounding box data from papermage to visually highlight the retrieved passages, Lucy suspects the retrieval component is likely underperforming. She makes a simple edit from `doc.sentences` to `doc.paragraphs` and observes how her system performs using different input granularity. She also realizes the system often retrieves content outside the main body text. She restricts her traversal to filter out paragraphs that overlap with footnotes—`[p.text for p in doc.paragraphs if not p.footnotes]`—making clever use of the cross-referencing functionality to detect when a paragraph is actually coming from a footnote. This example demonstrates the versatility of the affordances provided by magelib.

5 Conclusion

In this work, we've introduced papermage, an open-source Python toolkit for processing scientific documents. papermage was developed to supply high-quality data and reduce friction for research prototype development at Semantic Scholar. Today, it is being used in the production PDF processing pipeline to provide data for both the literature graph (Ammar et al., 2018; Kinney et al., 2023) and the paper-reading interface (Lo et al., 2023). It has also been used in working research prototypes which have since contributed to research publications (Fok et al., 2023; Kim et al., 2023).[6] We open-source papermage in hopes it will simplify research workflows that depend on scientific documents and promote extensions to other visually-rich documents like textbooks (Lincker et al., 2023) and digitized print media (Lee et al., 2020).

[6]See a demo of such a prototype papeo.app/demo.

Ethical Considerations

As a toolkit primarily designed to process scientific documents, there are two areas where `papermage` could cause harms or have unintended effects.

Extraction of bibliographic information `papermage` could be use to parse author names, affiliation, emails from scientific papers. Like any software, this extraction can be noisy, leading to incorrect parsing and thus misattribution of manuscript. Further, since `papermage` relies on static PDF documents, rather than metadata dynamically retrieved from publishers, it might extract and present names that should no longer be associated with authors, a harmful practice called deadnaming (Queer in AI et al., 2023). We recommend `papermage` users to exercise caution when using our toolkit to extract metadata, and cross reference extracted content with other sources when possible.

Misrepresentation or fabrication of information in documents In §3, we discussed how `papermage` can be easily extended to support high-level applications. Such applications might include question answering chatbots, or AI summarizers that perform information synthesis over one or more `papermage` documents. Such applications typically rely generative models to produce their output, which might fabricate incorrect information, or misstate claims. Developers should be vigilant when integrating `papermage` output into any downstream application.

References

2008–2023. Grobid. `https://github.com/kermitt2/grobid`.

Waleed Ammar, Dirk Groeneveld, Chandra Bhagavatula, Iz Beltagy, Miles Crawford, Doug Downey, Jason Dunkelberger, Ahmed Elgohary, Sergey Feldman, Vu Ha, Rodney Kinney, Sebastian Kohlmeier, Kyle Lo, Tyler Murray, Hsu-Han Ooi, Matthew Peters, Joanna Power, Sam Skjonsberg, Lucy Lu Wang, Chris Wilhelm, Zheng Yuan, Madeleine van Zuylen, and Oren Etzioni. 2018. Construction of the literature graph in semantic scholar. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, pages 84–91, New Orleans - Louisiana. Association for Computational Linguistics.

Tal August, Lucy Lu Wang, Jonathan Bragg, Marti A. Hearst, Andrew Head, and Kyle Lo. 2023. Paper plain: Making medical research papers approachable to healthcare consumers with natural language processing. *ACM Transactions on Computer-Human Interaction*. To appear.

Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. SciBERT: A pretrained language model for scientific text. 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)*, pages 3615–3620, Hong Kong, China. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Joseph Chee Chang, Saleema Amershi, and Ece Kamar. 2017. Revolt: Collaborative crowdsourcing for labeling machine learning datasets. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 2334–2346.

Joseph Chee Chang, Amy X Zhang, Jonathan Bragg, Andrew Head, Kyle Lo, Doug Downey, and Daniel S Weld. 2023. Citesee: Augmenting citations in scientific papers with persistent and personalized historical context. *ArXiv*, abs/2022.99999.

Catherine Chen, Zejiang Shen, Dan Klein, Gabriel Stanovsky, Doug Downey, and Kyle Lo. 2023. Are layout-infused language models robust to layout distribution shifts? a case study with scientific documents. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13345–13360, Toronto, Canada. Association for Computational Linguistics.

Isaac Councill, C. Lee Giles, and Min-Yen Kan. 2008. ParsCit: an open-source CRF reference string parsing package. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco. European Language Resources Association (ELRA).

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. 2021. A dataset of information-seeking questions and answers anchored in research papers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4599–4610, Online. Association for Computational Linguistics.

Raymond Fok, Hita Kambhamettu, Luca Soldaini, Jonathan Bragg, Kyle Lo, Marti Hearst, Andrew Head, and Daniel S Weld. 2023. Scim: Intelligent skimming support for scientific papers. In *IUI*.

Yu Gu, Robert Tinn, Hao Cheng, Michael R. Lucas, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. 2020. Domain-specific language model pretraining for biomedical natural language processing. *ACM Transactions on Computing for Healthcare (HEALTH)*, 3:1 – 23.

Andrew Head, Kyle Lo, Dongyeop Kang, Raymond Fok, Sam Skjonsberg, Daniel S. Weld, and Marti A. Hearst. 2021. Augmenting scientific papers with just-in-time, position-sensitive definitions of terms and symbols. *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*.

Zhi Hong, Aswathy Ajith, James Pauloski, Eamon Duede, Kyle Chard, and Ian Foster. 2023. The diminishing returns of masked language models to science. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 1270–1283, Toronto, Canada. Association for Computational Linguistics.

Po-Wei Huang, Abhinav Ramesh Kashyap, Yanxia Qin, Yajing Yang, and Min-Yen Kan. 2022a. Lightweight contextual logical structure recovery. In *Proceedings of the Third Workshop on Scholarly Document Processing*, pages 37–48, Gyeongju, Republic of Korea. Association for Computational Linguistics.

Yupan Huang, Tengchao Lv, Lei Cui, Yutong Lu, and Furu Wei. 2022b. Layoutlmv3: Pre-training for document ai with unified text and image masking. *Proceedings of the 30th ACM International Conference on Multimedia*.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised dense information retrieval with contrastive learning. *Transactions on Machine Learning Research*.

Sarthak Jain, Madeleine van Zuylen, Hannaneh Hajishirzi, and Iz Beltagy. 2020. SciREX: A challenge dataset for document-level information extraction. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7506–7516, Online. Association for Computational Linguistics.

Hyeonsu B Kang, Joseph Chee Chang, Yongsung Kim, and Aniket Kittur. 2022. Threddy: An interactive system for personalized thread-based exploration and organization of scientific literature. *arXiv preprint arXiv:2208.03455*.

Tae Soo Kim, Matt Latzke, Jonathan Bragg, Amy X. Zhang, and Joseph Chee Chang. 2023. Papeos: Augmenting research papers with talk videos. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*.

Rodney Michael Kinney, Chloe Anastasiades, Russell Authur, Iz Beltagy, Jonathan Bragg, Alexandra Buraczynski, Isabel Cachola, Stefan Candra, Yoganand Chandrasekhar, Arman Cohan, Miles Crawford, Doug Downey, Jason Dunkelberger, Oren Etzioni, Rob Evans, Sergey Feldman, Joseph Gorney, David W. Graham, F.Q. Hu, Regan Huff, Daniel King, Sebastian Kohlmeier, Bailey Kuehl, Michael Langan, Daniel Lin, Haokun Liu, Kyle Lo, Jaron Lochner, Kelsey MacMillan, Tyler Murray, Christopher Newell, Smita R Rao, Shaurya Rohatgi, Paul L Sayre, Zejiang Shen, Amanpreet Singh, Luca Soldaini, Shivashankar Subramanian, A. Tanaka, Alex D Wade, Linda M. Wagner, Lucy Lu Wang, Christopher Wilhelm, Caroline Wu, Jiangjiang Yang, Angele Zamarron, Madeleine van Zuylen, and Daniel S. Weld. 2023. The semantic scholar open data platform. *ArXiv*, abs/2301.10140.

Wei-Jen Ko, Te-yuan Chen, Yiyan Huang, Greg Durrett, and Junyi Jessy Li. 2020. Inquisitive question generation for high level text comprehension. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6544–6555, Online. Association for Computational Linguistics.

B. Lee, Jaime Mears, Eileen Jakeway, Meghan Ferriter, Chris Adams, Nathan Yarasavage, Deborah Thomas, Kate Zwaard, and Daniel S. Weld. 2020. The newspaper navigator dataset: Extracting headlines and visual content from 16 million historic newspaper pages in chronicling america. *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*.

Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. 2019. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240.

Yoonjoo Lee, Kyungjae Lee, Sunghyun Park, Dasol Hwang, Jaehyeon Kim, Hong-in Lee, and Moontae Lee. 2023. Qasa: Advanced question answering on scientific articles. In *Proceedings of the 40th International Conference on Machine Learning*. PMLR.

Élise Lincker, Olivier Pons, Camille Guinaudeau, Isabelle Barbet, Jérôme Dupire, Céline Hudelot, Vincent Mousseau, and Caroline Huron. 2023. Layout and activity-based textbook modeling for automatic pdf textbook extraction. In *iTextbooks@AIED*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

Kyle Lo, Joseph Chee Chang, Andrew Head, Jonathan Bragg, Amy X. Zhang, Cassidy Trier, Chloe Anastasiades, Tal August, Russell Authur, Danielle Bragg, Erin Bransom, Isabel Cachola, Stefan Candra, Yoganand Chandrasekhar, Yen-Sung Chen, Evie

(Yu-Yen) Cheng, Yvonne Chou, Doug Downey, Rob Evans, Raymond Fok, F.Q. Hu, Regan Huff, Dongyeop Kang, Tae Soo Kim, Rodney Michael Kinney, Aniket Kittur, Hyeonsu B Kang, Egor Klevak, Bailey Kuehl, Michael Langan, Matt Latzke, Jaron Lochner, Kelsey MacMillan, Eric Marsh, Tyler Murray, Aakanksha Naik, Ngoc-Uyen Nguyen, Srishti Palani, Soya Park, Caroline Paulic, Napol Rachatasumrit, Smita R Rao, Paul L Sayre, Zejiang Shen, Pao Siangliulue, Luca Soldaini, Huy Tran, Madeleine van Zuylen, Lucy Lu Wang, Christopher Wilhelm, Caroline M Wu, Jiangjiang Yang, Angele Zamarron, Marti A. Hearst, and Daniel S. Weld. 2023. The semantic reader project: Augmenting scholarly documents through ai-powered interactive reading interfaces. *ArXiv*, abs/2303.14334.

Kyle Lo, Lucy Lu Wang, Mark Neumann, Rodney Kinney, and Daniel Weld. 2020. S2ORC: The semantic scholar open research corpus. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4969–4983, Online. Association for Computational Linguistics.

Renqian Luo, Liai Sun, Yingce Xia, Tao Qin, Sheng Zhang, Hoifung Poon, and Tie-Yan Liu. 2022. Biogpt: Generative pre-trained transformer for biomedical text generation and mining. *Briefings in bioinformatics*.

Mark Neumann, Daniel King, Iz Beltagy, and Waleed Ammar. 2019. ScispaCy: Fast and robust models for biomedical natural language processing. In *Proceedings of the 18th BioNLP Workshop and Shared Task*, pages 319–327, Florence, Italy. Association for Computational Linguistics.

Benjamin Newman, Luca Soldaini, Raymond Fok, Arman Cohan, and Kyle Lo. 2023. A controllable qa-based framework for decontextualization. *ArXiv*, abs/2305.14772.

Organizers of Queer in AI, Anaelia Ovalle, Arjun Subramonian, Ashwin Singh, Claas Voelcker, Danica J. Sutherland, Davide Locatelli, Eva Breznik, Filip Klubicka, Hang Yuan, Hetvi J, Huan Zhang, Jaidev Shriram, Kruno Lehman, Luca Soldaini, Maarten Sap, Marc Peter Deisenroth, Maria Leonor Pacheco, Maria Ryskina, Martin Mundt, Milind Agarwal, Nyx Mclean, Pan Xu, A Pranav, Raj Korpan, Ruchira Ray, Sarah Mathew, Sarthak Arora, St John, Tanvi Anand, Vishakha Agrawal, William Agnew, Yanan Long, Zijie J. Wang, Zeerak Talat, Avijit Ghosh, Nathaniel Dennler, Michael Noseworthy, Sharvani Jha, Emi Baylor, Aditya Joshi, Natalia Y. Bilenko, Andrew Mcnamara, Raphael Gontijo-Lopes, Alex Markham, Evyn Dong, Jackie Kay, Manu Saraswat, Nikhil Vytla, and Luke Stark. 2023. Queer In AI: A Case Study in Community-Led Participatory AI. In *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '23, page 1882–1895, New York, NY, USA. Association for Computing Machinery.

Napol Rachatasumrit, Jonathan Bragg, Amy X. Zhang, and Daniel S. Weld. 2022. Citeread: Integrating localized citation contexts into scientific paper reading. *27th International Conference on Intelligent User Interfaces*.

Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1137–1149.

Nipun Sadvilkar and Mark Neumann. 2020. PySBD: Pragmatic sentence boundary disambiguation. In *Proceedings of Second Workshop for NLP Open Source Software (NLP-OSS)*, pages 110–114, Online. Association for Computational Linguistics.

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.

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. *arXiv preprint arXiv:2103.15348*.

Sanjay Subramanian, Lucy Lu Wang, Ben Bogin, Sachin Mehta, Madeleine van Zuylen, Sravanthi Parasa, Sameer Singh, Matt Gardner, and Hannaneh Hajishirzi. 2020. MedICaT: A dataset of medical images, captions, and textual references. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2112–2120, Online. Association for Computational Linguistics.

M. Tan, R. Pang, and Q. V. Le. 2020. Efficientdet: Scalable and efficient object detection. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10778–10787, Los Alamitos, CA, USA. IEEE Computer Society.

Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony S. Hartshorn, Elvis Saravia, Andrew Poulton, Viktor Kerkez, and Robert Stojnic. 2022. Galactica: A large language model for science. *ArXiv*, abs/2211.09085.

`ONNX`. 2023. `ONNX`. `https://github.com/onnx/onnx`.

`pdf2image`. 2023. `pdf2image`. `https://github.com/Belval/pdf2image`.

`pdfplumber`. 2023. `pdfplumber`. `https://github.com/jsvine/pdfplumber`.

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(4):317–335.

Amalie Trewartha, Nicholas Walker, Haoyan Huo, Sanghoon Lee, Kevin Cruse, John Dagdelen, Alex Dunn, Kristin Aslaug Persson, Gerbrand Ceder, and Anubhav Jain. 2022. Quantifying the advantage of domain-specific pre-training on named entity recognition tasks in materials science. *Patterns*, 3.

Lucy Lu Wang, Isabel Cachola, Jonathan Bragg, Evie (Yu-Yen) Cheng, Chelsea Hess Haupt, Matt Latzke, Bailey Kuehl, Madeleine van Zuylen, Linda M. Wagner, and Daniel S. Weld. 2021. Improving the accessibility of scientific documents: Current state, user needs, and a system solution to enhance scientific pdf accessibility for blind and low vision users. *ArXiv*, abs/2105.00076.

Lucy Lu Wang, Kyle Lo, Yoganand Chandrasekhar, Russell Reas, Jiangjiang Yang, Doug Burdick, Darrin Eide, Kathryn Funk, Yannis Katsis, Rodney Michael Kinney, Yunyao Li, Ziyang Liu, William Merrill, Paul Mooney, Dewey A. Murdick, Devvret Rishi, Jerry Sheehan, Zhihong Shen, Brandon Stilson, Alex D. Wade, Kuansan Wang, Nancy Xin Ru Wang, Christopher Wilhelm, Boya Xie, Douglas M. Raymond, Daniel S. Weld, Oren Etzioni, and Sebastian Kohlmeier. 2020. CORD-19: The COVID-19 open research dataset. In *Proceedings of the 1st Workshop on NLP for COVID-19 at ACL 2020*, Online. Association for Computational Linguistics.

Rosalee Wolfe, John McDonald, Ronan Johnson, Ben Sturr, Syd Klinghoffer, Anthony Bonzani, Andrew Alexander, and Nicole Barnekow. 2022. Supporting mouthing in signed languages: New innovations and a proposal for future corpus building. In *Proceedings of the 7th International Workshop on Sign Language Translation and Avatar Technology: The Junction of the Visual and the Textual: Challenges and Perspectives*, pages 125–130, Marseille, France. European Language Resources Association.

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. 2021. 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. 2019. Layoutlm: Pre-training of text and layout for document image understanding. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

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 Appendix

A.1 Comparison and Compatibility with XML

One can view `Layers` as capturing content hierarchy (e.g., tokens vs sentences) similar to that of other structured document representations, like TEI XML trees. We note that `Layers` are stored as unordered attributes and don't require nesting. This allows for specific cross-layer referencing operations that don't adhere to strict nesting relationships. For example:

```
for sentence in doc.sentences:
    for line in sentence.lines:
        ...
```

Recall that a sentence can begin or end midway through a line and cross multiple lines (§3.1). Similarly, not all lines are exactly contained within the boundaries of a sentence. As such, sentences and lines are not strictly nested within each other. This relationship would be difficult to encode in an XML format adhering to document tree structure.

Regardless, the way we represent structure in documents is highly versatile. We demonstrate this by also implementing `GrobidParser` as an alternative to the `PDF2TextParser` in §3.1. `GrobidParser` invokes Grobid to process PDFs, and reads the resulting TEI XML file generated by Grobid by converting each XML tag of a common level into an `Entity` of its own `Layer`. We use this to perform the evaluation in Table 2.

A.2 Additional `magelib` Protocols and Utilities

Serialization. Any `Document` and all of its `Layers` can be exported to a JSON format, and perfectly reconstructed:

```
import json
with open("....json", "w") as f_out:
    json.dump(doc.to_json(), f_out)

with open("...json", "r") as f_in:
    doc = json.load(f_in)
```

A.3 Evaluating `papermage`'s `CoreRecipe` against Grobid

Here, we detail how we performed the evaluation reported in §3.3 (Table 2). We also provide a full breakdown by category in Table 3.

As described earlier in the paper, Grobid is quite difficult to evaluate as it is developed with tight coupling between the PDF parser (`pdfalto`) and the models it employs to perform logical structure recovery over the resulting token stream. As such, there is no straightforward way to run just the model components of Grobid on an alternative token stream like that provided in the S2-VL (Shen et al., 2022) dataset.

To perform this baseline evaluation, we ran the original PDFs that were annotated for S2-VL through our `GrobidParser`, based on Grobid v0.7.3 (the latest version at the time of writing). Grobid also supports returning of bounding box coordinates of many of its produced categories (e.g., authors, abstract, paragraphs). We use these bounding boxes to create `Entities` that we annotate on a `Document` constructed manually from from S2-VL data. Using `magelib` cross-layer referencing, we were able to match Grobid predictions to S2-VL data to perform this evaluation.

Though we found there are certain categories for which bounding box information was either not available (e.g., Titles) or Grobid simply did not return that output (e.g., Figure text extraction). These are represented by zeros in Table 3, which contributes to the lower scores in Table 2 after macro averaging. For a more apples-to-apples comparison, we also included a "Grobid Subset" evaluation which restricted to just categories in S2-VL for which Grobid produced bounding box information.

In addition to Grobid, we evaluate two of our provided Transformer-based models. The RoBERTa-large (Liu et al., 2019) model is a `Transformers` token classification model that we finetuned on the S2-VL training set. The I-VILA model is a layout-infused Transformer model pretrained by Shen et al. (2022) on the S2-VL training set. Like we did with Grobid, we ran our `CoreRecipe` using these two models on the original PDFs in S2-VL, and performed a similar token mapping operation since our `PDF2TextParser` also produces a different token stream than that provided in S2-VL.

At the end of the day, the Transformer-based models performed better at this task than Grobid. This is unsurprising given expected performance improvements using a Transformer model compared to a CRF or BiLSTM. As well, the Transformer models were finetuned on S2-VL data, which gave them an advantage over Grobid, even accounting for both processes using token streams incompatible with S2-VL. Overall, this evaluation was intended to showcase how it is possible to use `papermage` to perform cross-system comparisons, even

Semantic Segment	GROBID$_{CRF}$			GROBID$_{NN}$			RoBERTa			I-VILA		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Abstract	81.9	89.1	85.3	85.3	89.8	87.5	89.2	93.7	91.4	97.4	98.3	97.8
Author	55.2	42.6	48.1	75.1	14.0	23.6	87.5	73.5	79.9	65.5	96.9	78.2
Bibliography	96.5	98.6	97.5	95.5	97.6	96.5	93.6	93.3	93.5	99.7	98.2	99.0
Caption	70.3	70.0	70.2	70.2	69.7	70.0	80.0	77.3	78.6	93.1	89.6	91.3
Equation	71.1	85.3	77.6	71.1	85.3	77.6	55.0	85.7	67.0	90.7	94.2	92.4
Figure	0.0	0.0	0.0	0.0	0.0	0.0	88.9	82.3	85.4	99.8	96.8	98.3
Footer	0.0	0.0	0.0	0.0	0.0	0.0	56.1	59.9	57.9	96.8	78.1	86.5
Footnote	0.0	0.0	0.0	0.0	0.0	0.0	59.8	44.3	50.9	80.2	93.5	86.3
Header	0.0	0.0	0.0	0.0	0.0	0.0	40.5	84.3	54.7	92.9	99.1	95.9
Keywords	0.0	0.0	0.0	0.0	0.0	0.0	93.8	97.1	95.4	96.9	99.4	98.1
List	0.0	0.0	0.0	0.0	0.0	0.0	61.9	63.8	62.9	76.7	82.4	79.4
Paragraph	94.5	89.8	92.1	94.4	89.9	92.1	93.5	93.0	93.3	98.7	97.9	98.3
Section	83.0	79.4	81.1	83.0	79.4	81.1	67.7	82.7	74.4	96.2	91.6	93.9
Table	97.3	58.6	73.2	97.9	58.6	73.3	94.7	71.8	81.7	96.1	94.9	95.5
Title	0.0	0.0	0.0	0.0	0.0	0.0	76.3	96.7	85.3	98.7	99.9	99.3
Macro Avg (Full S2-VL)	40.6	38.3	39.1	42.0	36.5	37.6	75.9	80.0	76.8	92.0	94.1	92.7
Macro Avg (Grobid Subset)	81.2	76.7	78.9	84.1	73.0	78.2	82.6	83.9	83.2	92.2	95.2	93.7

Table 3: Evaluating `CoreRecipe` for logical structure recovery on S2-VL (Shen et al., 2022). These are per-category metrics for Table 2. Metrics are computed for token-level classification, macro-averaged over categories. The "Grobid Subset" limits evaluation to only categories for which Grobid returns bounding box information, which was necessary for evaluation on S2-VL.