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

Kyle Lo, Zejiang Shen, Benjamin Newman, Joseph Chang, Russell Authur, Erin Bransom, Stefan Candra, Yoganand Chandrasekhar, Regan Huff, Bailey Kuehl, Amanpreet Singh, Chris Wilhelm, Angele Zamarron, Marti A. Hearst, Daniel Weld, Doug Downey, Luca Soldaini

Allen Institute for AI, Massachusetts Institute of Technology, University of California Berkeley, University of Washington, Northwestern University

2023 / EMNLP

발제자 : 정현우 (junghw3333@gmail.com)

랩 : HUMANE Lab

2024-02-06

Motivation

- They're often in difficult-to-use PDF formats, and the ecosystem of models to process them is fragmented and incomplete.

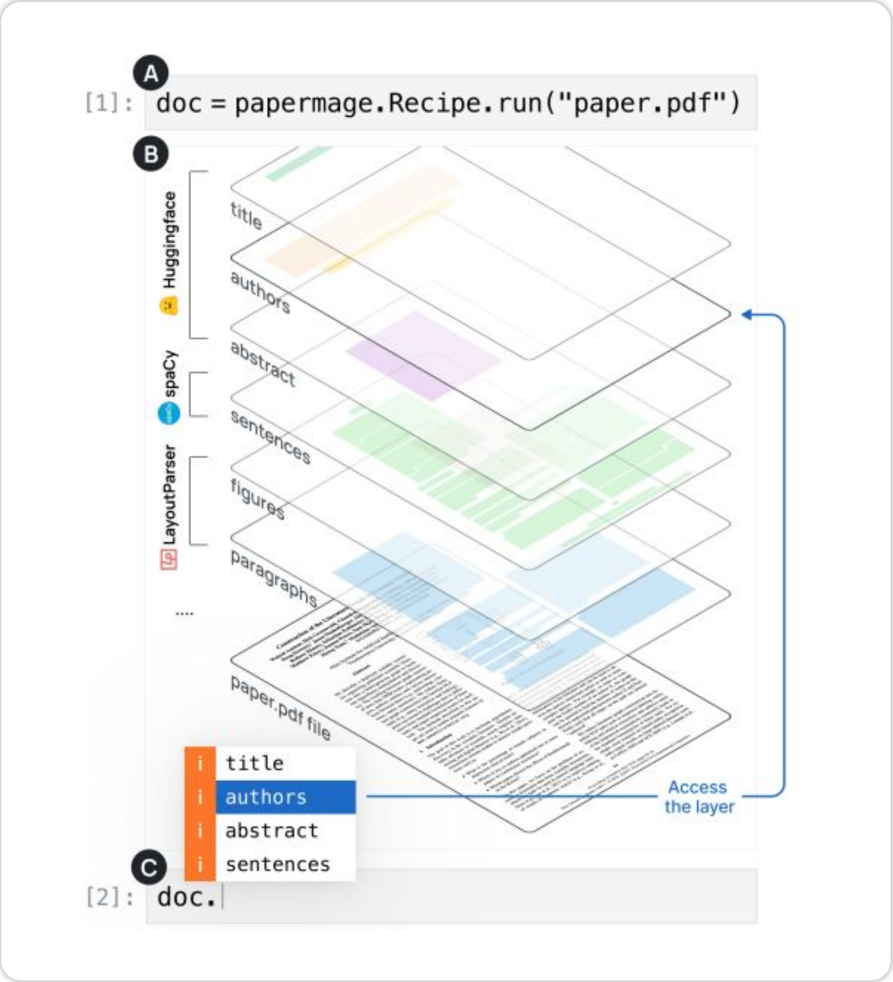
- As a result, users must write extensive custom code that strings pipelines of multiple models together. Research projects using models of different modalities can require hundreds of lines of code.

Contribution

- Magelib: a library of primitives and methods for representing and manipulating visually-rich documents as multimodal constructs. a library for intuitively representing and manipulating visuallyrich documents
- 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. it implementations of models for analyzing scientific papers that unify disparate machine learning frameworks under a common interface
- Recipes: which provide turn-key(installed complete and ready to operate) access to well-tested combinations of individual (often single-modality) modules to form sophisticated, extensible multimodal pipelines. It combinations of Predictors that form multimodal pipelines.

```
from papermage.recipes import CoreRecipe

recipe = CoreRecipe()
doc = recipe.run("tests/fixtures/papermage.pdf")
```

Papermage library



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

Kyle Lo Zejiang Shen Benjamin Newman Joseph Chee Chang
Russell Authur Erin Bransom Stefan Candra Yoganand Chandrasekhar
Regan Huff Bailey Kuehl Amanpreet Singh Chris Wilhelm Angele Zamarron
Marti A. Hearst Daniel S. Weld Doug Downey Luca Soldaini
Allen Institute for AI Massachusetts Institute of Technology
University of California Berkeley University of Washington 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

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.

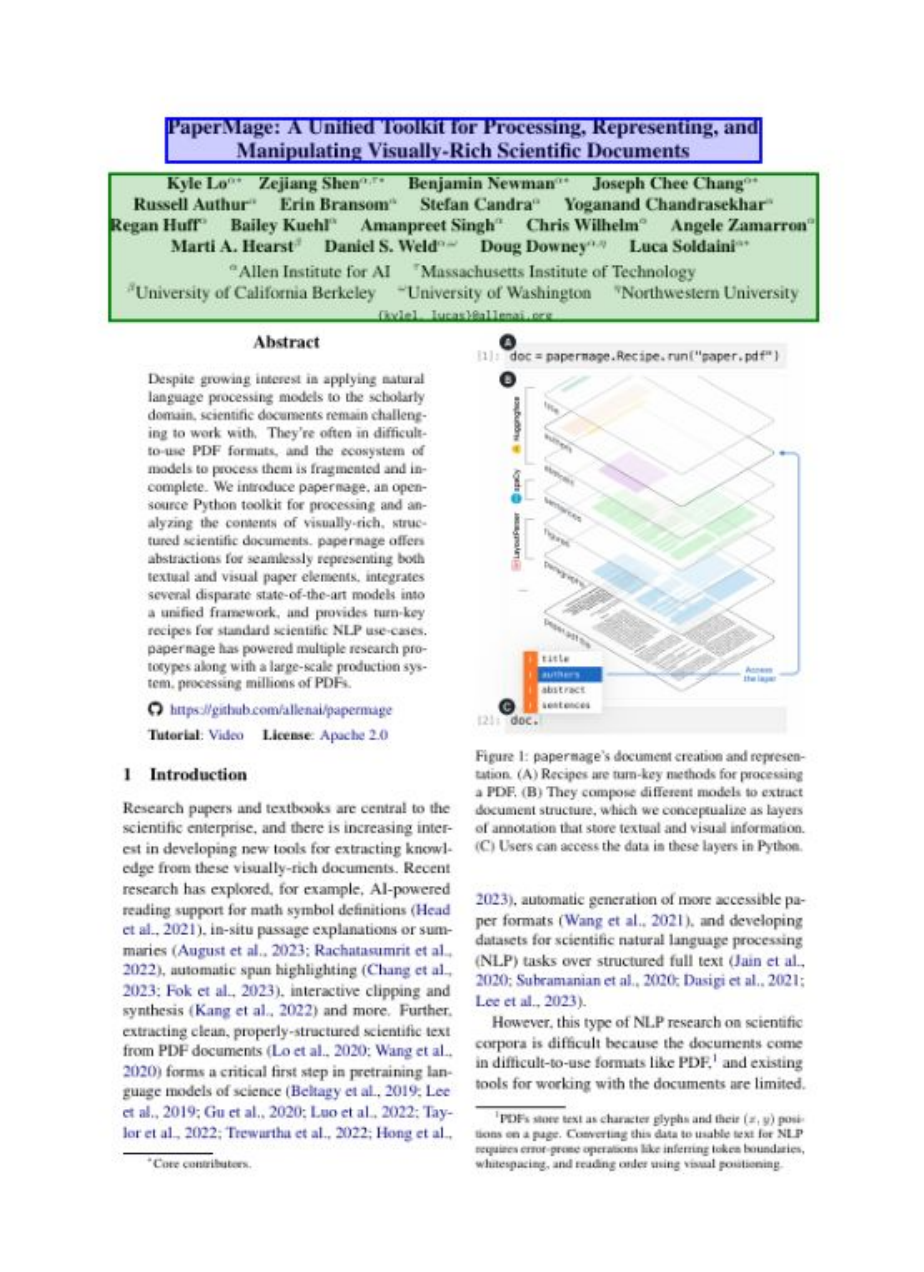
Figure 1: papermage's document creation and representation. (A) Recipes are turn-key methods for processing a PDF. (B) They compose different models to extract document structure, which we conceptualize as layers of annotation that store textual and visual information. (C) Users can access the data in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

*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.

Papermage library

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


AUTHORS:
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
```



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

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

Allen Institute for AI Massachusetts Institute of Technology University of California Berkeley University of Washington 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

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.

```
[1]: doc = papermage.Recipe.run("paper.pdf")
[2]: doc.
```

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.

Papermage library

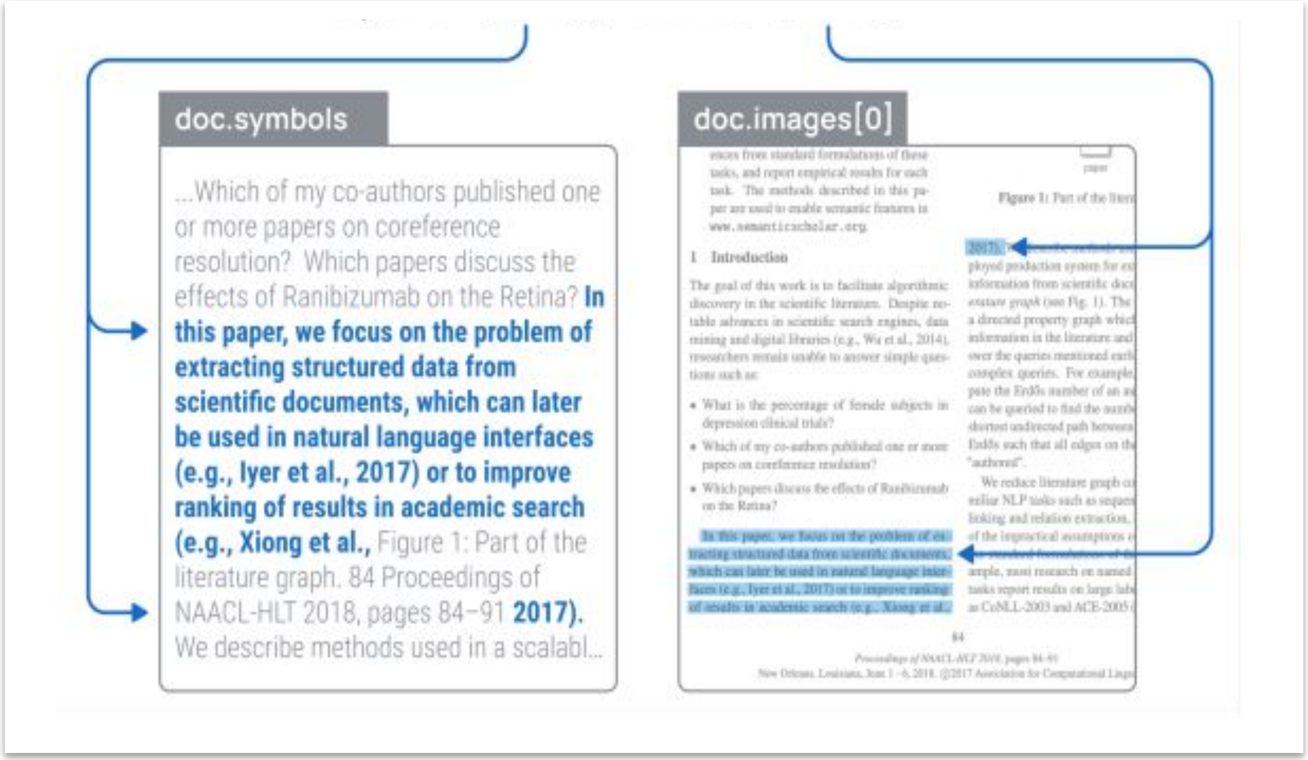
```
>>> sentences = Layer(entities=[
        Entity(...), Entity(...), ...
    ])
```

- Layers that can be accessed as attributes of a Document (e.g., doc.sentences, doc.figures, doc.tokens)
- Layer is a sequence of content units, called Entities, which store both textual (e.g., spans, strings) and visuospatial (e.g., bounding boxes, pixel arrays) information

```
>>> paragraphs = Layer(...)
>>> sentences = Layer(...)
>>> tokens = Layer(...)

>>> doc.add(paragraphs, sentences, tokens)
```

Papermage library

```
[3]: doc.sentences[9]

[3]: Entity(id=27, text="in this paper, we...",
            spans=[...], boxes=[...])
```

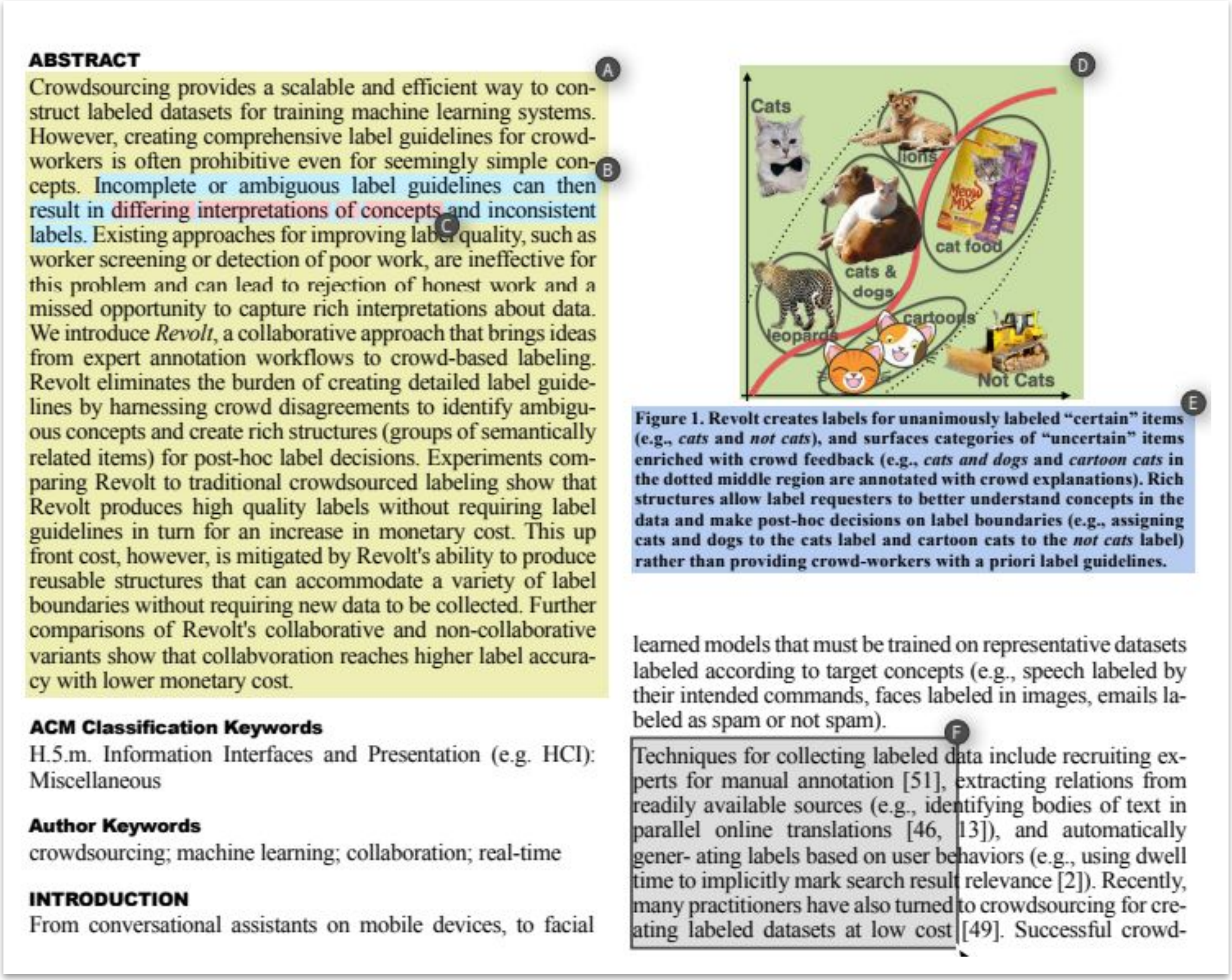


- It shows how “sentences” in a scientific document are represented as Entities
- boxes map its visual coordinates on a page.
- spans of a sentence are used to identify its text among all symbols.
- spans and boxes can include <u>non-contiguous units</u>, allowing great flexibility in Entities to <u>handle layout nuances</u>

Papermage library

이미지-캡션 pair를 만들기도 좋다

```
A  >>> doc.paragraphs[0]

B  >>> doc.paragraphs[0].sentences[2]
   or
   >>> doc.sentences[2]

C  >>> doc.sentences[2].tokens[9:13]
   or
   >>> doc.tokens[169:173]

D  >>> doc.figures[0]

E  >>> doc.captions[0]

F  >>> user_query = Box(l,t,w,h, page=0)

   >>> selected_tokens =
    doc.find(user_query, layer="tokens")

   >>> [token.text for
            token in selected_tokens]

   ["Techniques", "for", "collecting",
   "labeled", "data", "perts", "for",
   "manual", "annotation", ...]
```



ABSTRACT

Crowdsourcing provides a scalable and efficient way to construct labeled datasets for training machine learning systems. However, creating comprehensive label guidelines for crowdworkers is often prohibitive even for seemingly simple concepts. Incomplete or ambiguous label guidelines can then result in differing interpretations of concepts and inconsistent labels. Existing approaches for improving label quality, such as worker screening or detection of poor work, are ineffective for this problem and can lead to rejection of honest work and a missed opportunity to capture rich interpretations about data. We introduce *Revolt*, a collaborative approach that brings ideas from expert annotation workflows to crowd-based labeling. Revolt eliminates the burden of creating detailed label guidelines by harnessing crowd disagreements to identify ambiguous concepts and create rich structures (groups of semantically related items) for post-hoc label decisions. Experiments comparing Revolt to traditional crowdsourced labeling show that Revolt produces high quality labels without requiring label guidelines in turn for an increase in monetary cost. This up front cost, however, is mitigated by Revolt's ability to produce reusable structures that can accommodate a variety of label boundaries without requiring new data to be collected. Further comparisons of Revolt's collaborative and non-collaborative variants show that collabvoration reaches higher label accuracy with lower monetary cost.

ACM Classification Keywords

H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous

Author Keywords

crowdsourcing; machine learning; collaboration; real-time

INTRODUCTION

From conversational assistants on mobile devices, to facial

Figure 1. Revolt creates labels for unanimously labeled "certain" items (e.g., *cats* and *not cats*), and surfaces categories of "uncertain" items enriched with crowd feedback (e.g., *cats and dogs* and *cartoon cats* in the dotted middle region are annotated with crowd explanations). Rich structures allow label requesters to better understand concepts in the data and make post-hoc decisions on label boundaries (e.g., assigning cats and dogs to the cats label and cartoon cats to the *not cats* label) rather than providing crowd-workers with a priori label guidelines.

learned models that must be trained on representative datasets labeled according to target concepts (e.g., speech labeled by their intended commands, faces labeled in images, emails labeled as spam or not spam).

Techniques for collecting labeled data include recruiting experts for manual annotation [51], extracting relations from readily available sources (e.g., identifying bodies of text in parallel online translations [46, 13]), and automatically gener- ating labels based on user behaviors (e.g., using dwell time to implicitly mark search result relevance [2]). Recently, many practitioners have also turned to crowdsourcing for creating labeled datasets at low cost [49]. Successful crowd-

Predictors

```
>>> paragraphs = Layer(...)
>>> sentences = Layer(...)
>>> tokens = Layer(...)

>>> doc.add(paragraphs, sentences, tokens)
```

papermage provides a unified interface, called Predictors, to ensure models produce Entities that are compatible with the Document.

papermage includes several ready-to-use Predictors that leverage state-of-the-art models to extract specific document structures

Predictors

Semantic (paragraphs, setences, tokens)
Layout (pages, blocks, lines) -> Box 만들어준다.
Logical (title, abstract) -> Span 만들어준다.

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` wraps Token Classifiers from `Transformers` (Wolfe et al., 2022), which provides 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.
Task-specific	Models for a given scientific document processing task can be used with `papermage` if wrapped as a `Predictor` following common patterns.	As many practitioners depend on prompting a model through an API call, we implement `APIPredictor` which interfaces external APIs, such as GPT-3 (Brown et al., 2020), to perform tasks like question answering over a structured `Document`. We also implement `SnippetRetrievalPredictor` which wraps models like Contriever (Izacard et al., 2022) to perform top-k within-document snippet retrieval. See §4 for how these two can be combined.

Table 1: Types of `Predictors` implemented in `papermage`.

Predictors

papermage includes several ready-to-use Predictors that leverage state-of-the-art models to extract specific document structures

Use case	Description	Examples
Linguistic/ Semantic	Segments doc into text units often used for down-stream 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` wraps Token Classifiers from `Transformers` (Wolfe et al., 2022), which provides 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.
Task-specific	Models for a given scientific document processing task can be used with `papermage` if wrapped as a `Predictor` following common patterns.	As many practitioners depend on prompting a model through an API call, we implement `APIPredictor` which interfaces external APIs, such as GPT-3 (Brown et al., 2020), to perform tasks like question answering over a structured `Document`. We also implement `SnippetRetrievalPredictor` which wraps models like Contriever (Izacard et al., 2022) to perform top-k within-document snippet retrieval. See §4 for how these two can be combined.

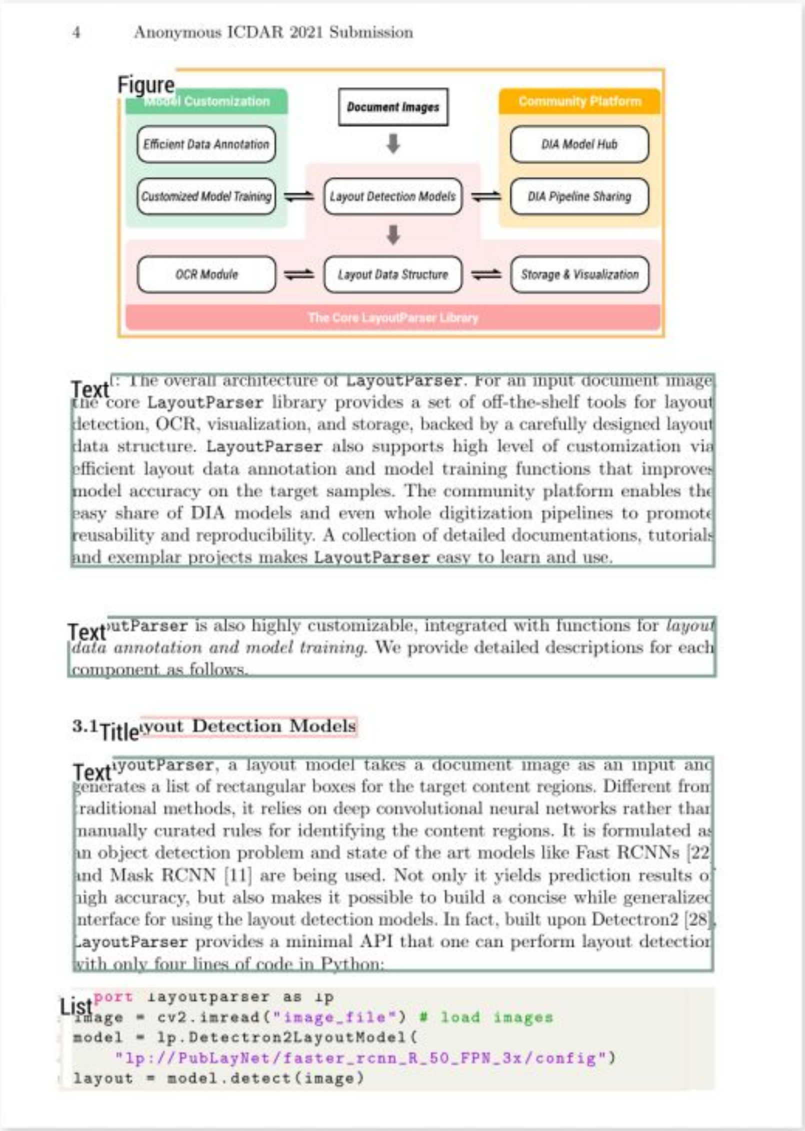
Table 1: Types of `Predictors` implemented in `papermage`.

```
>>> import papermage as pm
>>> cv = pm.BoxPredictor(...)
>>> tables, figures = cv.predict(doc)
>>> doc.add(tables, figures)

>>> nlp_neu = pm.SpanPredictor(...)
>>> titles, authors = nlp_neu.predict(doc)
>>> doc.add(titles, authors)

>>> nlp_sym = pm.SentencePredictor(...)
>>> sentences = nlp_sym.predict(doc)
>>> doc.add(sentences)
```

Predictors

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` wraps Token Classifiers from `Transformers` (Wolfe et al., 2022), which provides 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.
Task-specific	Models for a given scientific document processing task can be used with `papermage` if wrapped as a `Predictor` following common patterns.	As many practitioners depend on prompting a model through an API call, we implement `APIPredictor` which interfaces external APIs, such as GPT-3 (Brown et al., 2020), to perform tasks like question answering over a structured `Document`. We also implement `SnippetRetrievalPredictor` which wraps models like Contriever (Izacard et al., 2022) to perform top-k within-document snippet retrieval. See §4 for how these two can be combined.

Table 1: Types of `Predictors` implemented in `papermage`.



Mode I: Showing Layout on the Original Image

Predictors

Examples

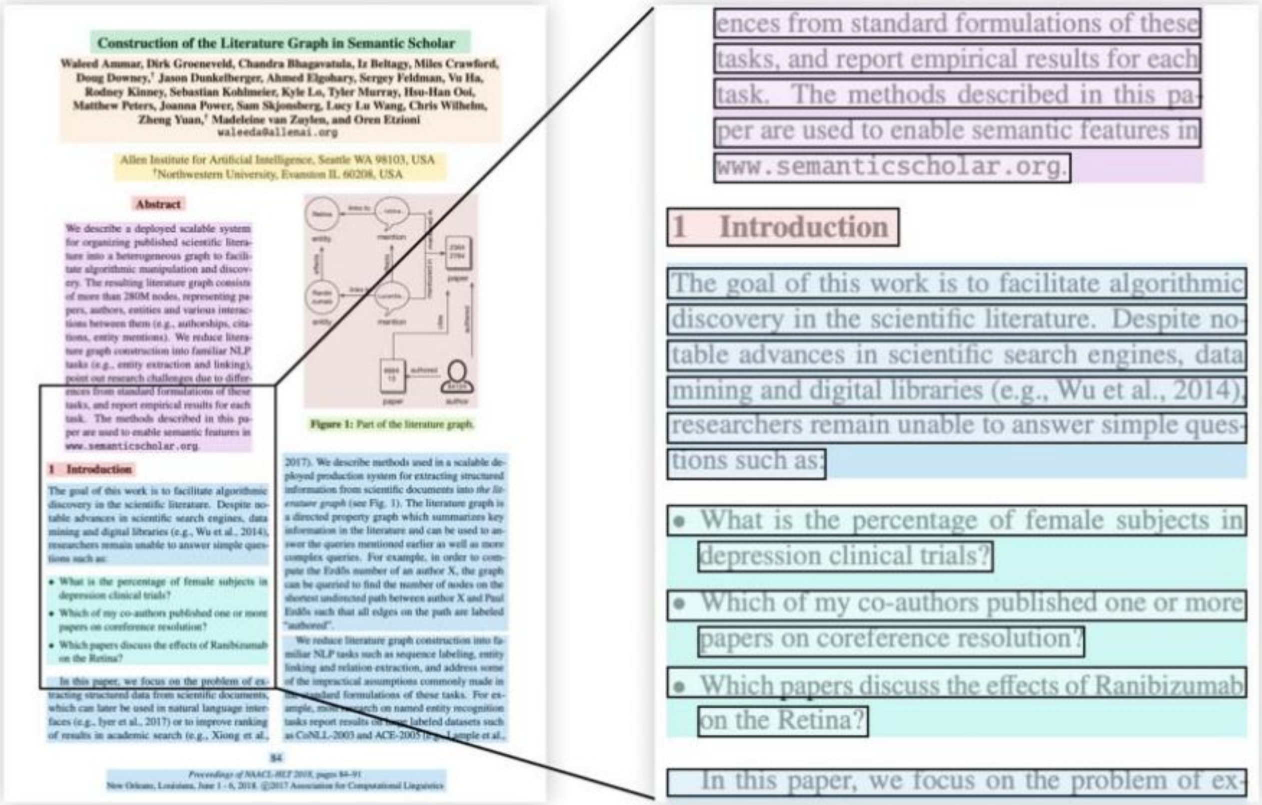
SentencePredictor wraps sciSpaCy (Neumann et al., 2019) and PySBD (Sadvilkar and Neumann, 2020) to segment sentences. WordPredictor is a custom scikit-learn model to identify broken words split across PDF lines or columns. ParagraphPredictor is a set of heuristics on top of both layout and logical structure models to extract paragraphs.

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).

SpanPredictor wraps Token Classifiers from Transformers (Wolfe et al., 2022), which provides 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.

As many practitioners depend on prompting a model through an API call, we implement APIPredictor which interfaces external APIs, such as GPT-3 (Brown et al., 2020), to perform tasks like question answering over a structured Document. We also implement SnippetRetrievalPredictor which wraps models like Contriever (Izacard et al., 2022) to perform top-k within-document snippet retrieval. See §4 for how these two can be combined.

es of Predictors implemented in papermage.



(a) Paper PDF Screenshot

The text blocks are highlighted in colored boxes.

(b) Examples of VIsual LAyout Groups

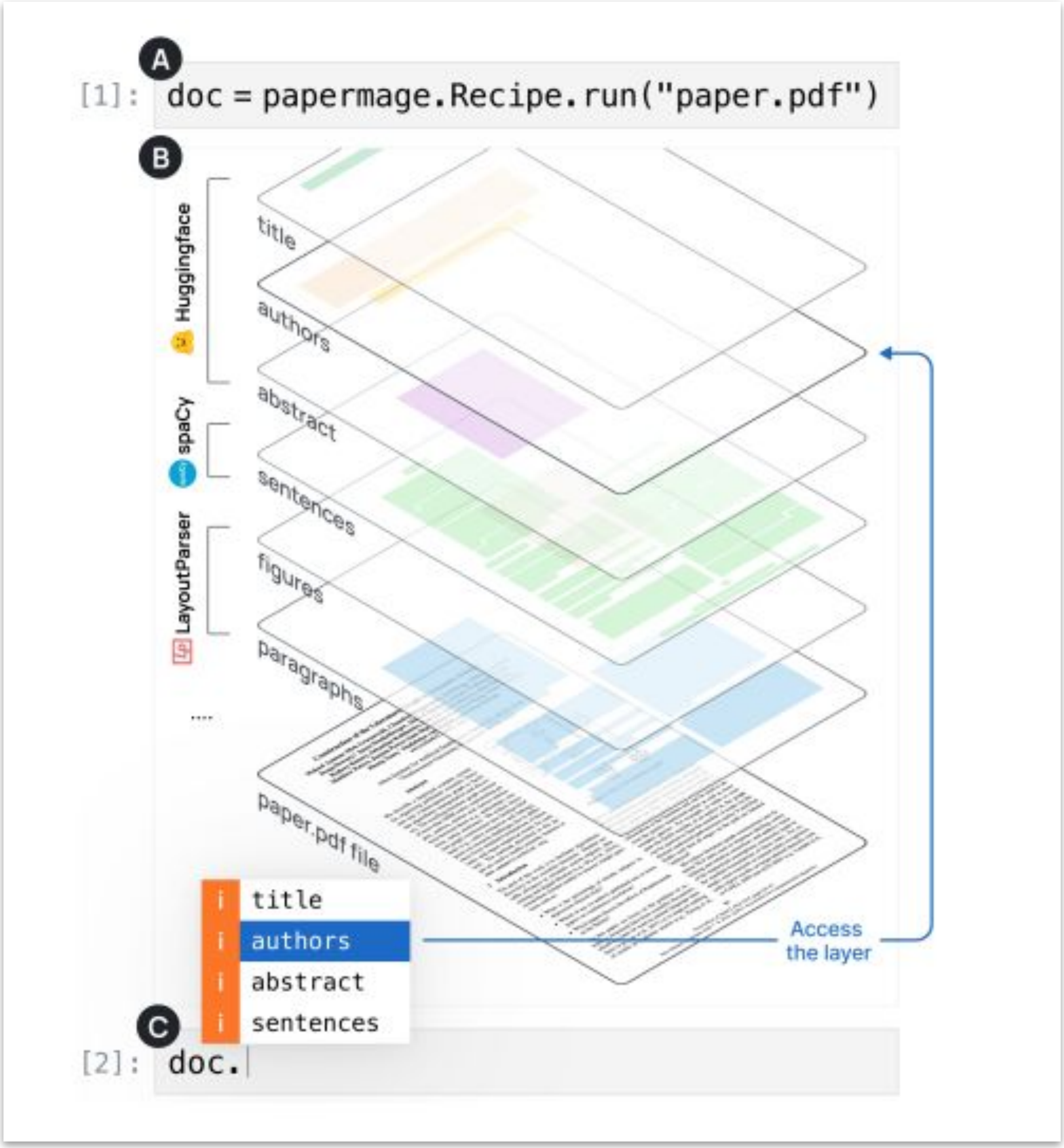
Tokens in the same text lines or blocks usually have the same semantic category.

- VILA 모델

Recipe

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

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



QA system

a researcher (Lucy) is prototyping an attributed QA system for science

System Design: In her prototype interface, a user can highlight a passage in a PDF and ask a question about it. The prototype then uses the text of the retrieved passages along with the user question to prompt a language model to generate an answer. the prototype also visually highlights the retrieved passages as supporting evidence to the generated answer.

Task-specific	Models for a given scientific document processing task can be used with papermage if wrapped as a `Predictor` following common patterns.	As many practitioners depend on prompting a model through an API call, we implement `APIPredictor` which interfaces external APIs, such as GPT-3 (Brown et al., 2020), to perform tasks like question answering over a structured `Document`. We also implement `SnippetRetrievalPredictor` which wraps models like Contriever (Izacard et al., 2022) to perform top-k within-document snippet retrieval. See §4 for how these two can be combined.

Conclusion

- Papermage 툴킷 제공을 통해 과학적인 pdf(논문) 분석하기 편해졌다.
- 각각의 파편화된 연구들을 잘 통합하였다.
- 이를 통해 복잡한 코드 없이 연구 workflow를 간단화 시킬 수 있다.
- Visually rich documents들을 더 많이 만들 수 있다.

각자 생각하는 활용방안?