

[🏠](#) [■ How-to guides ■](#)[How to load PDFs](#)

How to load PDFs

[🔗 Open in Colab](#)[🔄 Open on GitHub](#)

[Portable Document Format \(PDF\)](#), standardized as ISO 32000, is a file format developed by Adobe in 1992 to present documents, including text formatting and images, in a manner independent of application software, hardware, and operating systems.

This guide covers how to [load](#) [PDF](#) documents into the LangChain [Document](#) format that we use downstream.

Text in PDFs is typically represented via text boxes. They may also contain images. A PDF parser might do some combination of the following:

- Agglomerate text boxes into lines, paragraphs, and other structures

via heuristics or ML inference;

- Run **OCR** on images to detect text therein;
- Classify text as belonging to paragraphs, lists, tables, or other structures;
- Structure text into table rows and columns, or key-value pairs.

LangChain integrates with a host of PDF parsers. Some are simple and relatively low-level; others will support OCR and image-processing, or perform advanced document layout analysis. The right choice will depend on your needs. Below we enumerate the possibilities.

We will demonstrate these approaches on a **sample file**:

```
file_path = (  
    "../../docs/integrations/document_loaders/  
    parser-paper.pdf"  
)
```



A NOTE ON MULTIMODAL MODELS

Many modern LLMs support inference over multimodal inputs (e.g., images). In some applications -- such as question-answering over PDFs with complex layouts, diagrams, or scans -- it may be advantageous to skip the PDF parsing, instead casting a PDF page to an image and passing it to a model directly. We demonstrate an example of this in the [Use of multimodal models](#) section below.

Simple and fast text extraction

If you are looking for a simple string representation of text that is embedded in a PDF, the method below is appropriate. It will return a list of **Document** objects--one per page--containing a single string of the page's text in the Document's `page_content` attribute. It will not parse text in images or scanned PDF pages. Under the hood it uses the

PyPDF2 Python library.

LangChain **document loaders**

implement `load` and its async variant, `aload`, which return iterators of `Document` objects. We will use these below.

```
%pip install -qU pypdf
```

```
from langchain_community.document_loaders import PyPDFLoader

loader = PyPDFLoader(file_path)
pages = []
async for page in loader.aload():
    pages.append(page)
```

```
print(f"{pages[0].metadata}\n")
print(pages[0].page_content)
```

```
{'source':
'../../docs/integrations/document_loaders/pypdf/pypdf_parser-paper.pdf', 'page': 0}
```

LayoutParser : A Unified Toolkit for Deep Learning Based Document Image Analysis
Zejiang Shen1(✉), Ruochen Zhang1, Charles G. B. Zeller, Charles Germain

Lee⁴, Jacob Carlson³, and Weir¹
1Allen Institute for AI
shannons@allenai.org
2Brown University
ruochen_zhang@brown.edu
3Harvard University
{melissadell,jacob carlson }@t
4University of Washington
bcgl@cs.washington.edu
5University of Waterloo
w422li@uwaterloo.ca

Abstract. Recent advances in c
have been
primarily driven by the applic
Ideally, research
outcomes could be easily depl
for further
investigation. However, variou
codebases
and sophisticated model config
reuse of im-
portant innovations by a wide
on-going
efforts to improve reusability
model
development in disciplines lik
and computer
vision, none of them are optim
domain of DIA.
This represents a major gap in
central to
academic research across a wid
social sciences
and humanities. This paper int
source
library for streamlining the u
applica-
tions. The core LayoutParser
simple and
intuitive interfaces for apply
for layout de-

tection, character recognition processing tasks.

To promote extensibility, Layo community platform for sharing both pre-digitization pipelines. We demonstrate for both lightweight and large-scale document use cases.

The library is publicly available at [parser.github.io](https://github.com/layo/parser.github.io).

Keywords: Document Image Analysis · Character Recognition · Open Source

1 Introduction

Deep Learning(DL)-based approach for a wide range of document image analysis (DIA) tasks such as classification [11, arXiv:2103.01010].

Note that the metadata of each document stores the corresponding page number.

Vector search over PDFs

Once we have loaded PDFs into LangChain `Document` objects, we can index them (e.g., a RAG application) in the usual way. Below we use OpenAI embeddings, although any LangChain `embeddings` model will suffice.

```
%pip install -qU langchain-openai
```

```
import getpass
import os

if "OPENAI_API_KEY" not in os.environ:

os.environ["OPENAI_API_KEY"]
= getpass.getpass("OpenAI
API Key:")
```

```
from langchain_core.vectorstores import
InMemoryVectorStore
from langchain_openai import
OpenAIEmbeddings


vector_store =
InMemoryVectorStore.from_documents(
OpenAIEmbeddings(),
docs =
vector_store.similarity_search(
"LayoutParser?", k=2)
for doc in docs:
    print(f"Page {doc.metadata['page']}
{doc.page_content[:300]}\n")
```

API

Reference: [InMemoryVectorStore](#)

Page 13: 14 Z. Shen et al.
6 Conclusion
LayoutParser provides a
comprehensive toolkit for

deep learning-based document image analysis. The off-the-shelf library is easy to install, and can be used to build flexible and accurate pipelines for processing documents with complicated structures. It also supports hi

Page 0: LayoutParser : A Unified Toolkit for Deep Learning Based Document Image Analysis
Zejiang Shen1() , Ruochen Zhang2, Melissa Dell3, Benjamin Charles Germain Lee4, Jacob Carlson3, and Weining Li5
1Allen Institute for AI
shannons@allenai.org
2Brown University
ruochen_zhang@brown.edu
3Harvard University

Layout analysis and extraction of text from images

If you require a more granular segmentation of text (e.g., into distinct paragraphs, titles, tables, or

other structures) or require extraction of text from images, the method below is appropriate. It will return a list of [Document](#) objects, where each object represents a structure on the page. The Document's metadata stores the page number and other information related to the object (e.g., it might store table rows and columns in the case of a table object).

Under the hood it uses the

`langchain-unstructured` library.

See the [integration docs](#) for more information about using [Unstructured](#) with LangChain.

Unstructured supports multiple parameters for PDF parsing:

- `strategy` (e.g., `"fast"` or `"hi-res"`)
- API or local processing. You will need an API key to use the API.

The [hi-res](#) strategy provides support for document layout analysis and OCR. We demonstrate it below via the API. See [local parsing](#) section below

for considerations when running locally.

```
%pip install -qU langchain-unstructured
```

```
import getpass
import os

if "UNSTRUCTURED_API_KEY" not in os.environ:

    os.environ["UNSTRUCTURED_API_KEY"] = getpass.getpass("Unstructured API Key:")
```

```
Unstructured API Key:
.....
```

As before, we initialize a loader and load documents lazily:

```
from langchain_unstructured
import UnstructuredLoader

loader =
UnstructuredLoader(
    file_path=file_path,
    strategy="hi_res",
    partition_via_api=True,
    coordinates=True,
)
docs = []
for doc in
```

```
loader.lazy_load():  
docs.append(doc)
```

```
INFO: Preparing to split document  
INFO: Starting page number set to 1  
INFO: Allow failed set to 0  
INFO: Concurrency level set to 4  
INFO: Splitting pages 1 to 16  
INFO: Determined optimal split size of 4  
INFO: Partitioning 4 files with 4 partitions  
INFO: Partitioning set #1 (pages 1-4)  
INFO: Partitioning set #2 (pages 5-8)  
INFO: Partitioning set #3 (pages 9-12)  
INFO: Partitioning set #4 (pages 13-16)  
INFO: HTTP Request: POST https://api.unstructuredapp.io/api/v1/partition  
"HTTP/1.1 200 OK"  
INFO: HTTP Request: POST https://api.unstructuredapp.io/api/v1/partition  
"HTTP/1.1 200 OK"  
INFO: HTTP Request: POST https://api.unstructuredapp.io/api/v1/partition  
"HTTP/1.1 200 OK"  
INFO: HTTP Request: POST https://api.unstructuredapp.io/api/v1/partition  
"HTTP/1.1 200 OK"  
INFO: Successfully partitioned document and  
added to the final result.  
INFO: Successfully partitioned document and  
added to the final result.  
INFO: Successfully partitioned document and  
added to the final result.  
INFO: Successfully partitioned document and  
added to the final result.
```

Here we recover 171 distinct
structures over the 16 page document:

```
print(len(docs))
```

```
171
```

We can use the document metadata to recover content from a single page:

```
first_page_docs = [doc for doc in docs if doc.metadata.get("page_number") == 1]

for doc in first_page_docs:
    print(doc.page_content)
```

```
LayoutParser: A Unified
Toolkit for Deep Learning
Based Document Image
Analysis
1 2 0 2 n u J 1 2 ] V C . s
c [ 2 v 8 4 3 5 1 . 3 0 1 2
: v i X r a
Zejiang Shen® (<), Ruochen
Zhang?, Melissa Dell®,
Benjamin Charles Germain
Lee?, Jacob Carlson®, and
Weining Li®
1 Allen Institute for AI
shannons@allenai.org 2
Brown University ruochen
zhang@brown.edu 3 Harvard
University
{melissadell,jacob
carlson}@fas.harvard.edu 4
University of Washington
```

bcgl@cs.washington.edu 5
University of Waterloo
w422li@uwaterloo.ca
Abstract. Recent advances in document image analysis (DIA) have been primarily driven by the application of neural networks. Ideally, research outcomes could be easily deployed in production and extended for further investigation. However, various factors like loosely organized codebases and sophisticated model configurations complicate the easy reuse of important innovations by a wide audience. Though there have been on-going efforts to improve reusability and simplify deep learning (DL) model development in disciplines like natural language processing and computer vision, none of them are optimized for challenges in the domain of DIA. This represents a major gap in the existing toolkit, as DIA is central to academic research across a wide range of disciplines in the social sciences and humanities. This paper introduces LayoutParser, an open-source library for streamlining the usage of DL in DIA research and applications. The core

LayoutParser library comes with a set of simple and intuitive interfaces for applying and customizing DL models for layout detection, character recognition, and many other document processing tasks. To promote extensibility, LayoutParser also incorporates a community platform for sharing both pre-trained models and full document digitization pipelines. We demonstrate that LayoutParser is helpful for both lightweight and large-scale digitization pipelines in real-world use cases. The library is publicly available at <https://layout-parser.github.io>.

Keywords: Document Image Analysis · Deep Learning · Layout Analysis · Character Recognition · Open Source library · Toolkit.

1 Introduction

Deep Learning(DL)-based approaches are the state-of-the-art for a wide range of document image analysis (DIA) tasks including document image classification [11,

Extracting tables and other structures

Each `Document` we load represents a structure, like a title, paragraph, or table.

Some structures may be of special interest for indexing or question-answering tasks. These structures may be:

1. Classified for easy identification;
2. Parsed into a more structured representation.

Below, we identify and extract a table:

▶ Click to expand code for rendering pages

```
render_page(docs, 5)
```



Text
Title
Table

LayoutParser: A Unified Toolkit for DL-Based DIA ⓘ

Table 1: Current layout detection models in the LayoutParser model zoo

| Dataset | Base Model | Large Model | Notes |
|----------------|------------|-------------|--|
| PubLayNet [38] | F / M | M | Layouts of modern scientific documents |
| PRImA [3] | M | - | Layouts of scanned modern magazines and scientific reports |
| Newspaper [17] | F | - | Layouts of scanned US newspapers from the 20th century |
| TableBank [18] | F | F | Table region on modern scientific and business document |
| HJDataset [31] | F / M | - | Layouts of history Japanese documents |

For each dataset, we train several models of different sizes (the trade-off between accuracy vs. computational cost). For "base model" and "large model", we refer to using the ResNet 50 or ResNet 101 backbones [15], respectively. One can train models of different architectures, like Faster R-CNN [26] (F) and Mask R-CNN [32] (M). For example, an F in the Large Model column indicates it has a Faster R-CNN model trained using the ResNet 101 backbone. The platform is maintained and a number of additions will be made to the model zoo in coming months.

layout data structures, which are optimized for efficiency and versatility. 3) When necessary, users can employ existing or customized OCR models via the unified API provided in the *OCR module*. 4) LayoutParser comes with a set of utility functions for the visualization and storage of the layout data. 5) LayoutParser is also highly customizable, via its integration with functions for *layout data annotation and model training*. We now provide detailed descriptions for each component.

3.1 Layout Detection Models

In LayoutParser, a layout model takes a document image as an input and generates a list of rectangular boxes for the target content regions. Different from traditional methods, it relies on deep convolutional neural networks rather than manually curated rules to identify content regions. It is formulated as an object detection problem and state-of-the-art models like Faster R-CNN [26] and Mask R-CNN [12] are used. This yields prediction results of high accuracy and makes it possible to build a concise, generalized interface for layout detection. LayoutParser, built upon Detectron2 [35], provides a minimal API that can perform layout detection with only four lines of code in Python:

```
import layoutparser as lp
image = cv2.imread('image_file') # load images
model = lp.Detectron2LayoutModel(
    'lp://PubLayNet/faster_rcnn_R_50_FPN_3x/config')
layout = model.detect(image)
```

LayoutParser provides a wealth of pre-trained model weights using various datasets covering different languages, time periods, and document types. Due to domain shift [7], the prediction performance can notably drop when models are applied to target samples that are significantly different from the training dataset. As document structures and layouts vary greatly in different domains, it is important to select models trained on a dataset similar to the test samples. A semantic syntax is used for initializing the model weights in LayoutParser, using both the dataset name and model name `lp://<dataset-name>/<model-architecture-name>`.

LayoutParser: A Unified Toolkit for DL-Based DIA

5

Table 1: Current layout detection models in the LayoutParser model zoo

| Dataset | Base Model | Large Model | Notes |
|----------------|------------|-------------|--|
| PubLayNet [38] | F / M | M | Layouts of modern scientific documents |
| PRImA [3] | M | - | Layouts of scanned modern magazines and scientific reports |
| Newspaper [17] | F | - | Layouts of scanned US newspapers from the 20th century |
| TableBank [18] | F | F | Table region on modern scientific and |
| HJDataset [31] | F / M | - | Layouts of history Japanese documents |

business document Layouts
of history Japanese
documents

1 For each dataset, we train several models of different sizes for different needs (the trade-off between accuracy vs. computational cost). For “base model” and “large model”, we refer to using the ResNet 50 or ResNet 101 backbones [13], respectively. One can train models of different architectures, like Faster R-CNN [28] (F) and Mask R-CNN [12] (M). For example, an F in the Large Model column indicates it has a Faster R-CNN model trained using the ResNet 101 backbone. The platform is maintained and a number of additions will be made to the model zoo in coming months.

layout data structures, which are optimized for efficiency and versatility.

3) When necessary, users can employ existing or customized OCR models via the unified API provided in the OCR module.

4) LayoutParser comes with a set of utility functions for the visualization and storage of the layout data.

5) LayoutParser is also

highly customizable, via its integration with functions for layout data annotation and model training. We now provide detailed descriptions for each component.

3.1 Layout Detection Models

In LayoutParser, a layout model takes a document image as an input and generates a list of rectangular boxes for the target content regions. Different from traditional methods, it relies on deep convolutional neural networks rather than manually curated rules to identify content regions. It is formulated as an object detection problem and state-of-the-art models like Faster R-CNN [28] and Mask R-CNN [12] are used. This yields prediction results of high accuracy and makes it possible to build a concise, generalized interface for layout detection. LayoutParser, built upon Detectron2 [35], provides a minimal API that can perform layout detection with only four lines of code in Python:

```
1 import layoutparser as lp
```

```
2 image = cv2 . imread ( "
image_file " ) # load
images 3 model = lp . De t
e c t r o n 2 Lay outM odel (
" lp :// PubLayNet / f a s t
er _ r c nn _ R _ 50 _ F P
N_ 3 x / config " ) 4 5
layout = model . detect (
image )
```

LayoutParser provides a wealth of pre-trained model weights using various datasets covering different languages, time periods, and document types. Due to domain shift [7], the prediction performance can notably drop when models are applied to target samples that are significantly different from the training dataset. As document structures and layouts vary greatly in different domains, it is important to select models trained on a dataset similar to the test samples. A semantic syntax is used for initializing the model weights in LayoutParser, using both the dataset name and model name `lp://<dataset-name>/<model-architecture-name>`.

Note that although the table text is collapsed into a single string in the

document's content, the metadata contains a representation of its rows and columns:

```
from IPython.display import HTML

segments = [
    doc.metadata
    for doc in docs
    if doc.metadata.get("page") == 5 and doc.metadata.get("category") == "Table"
]

display(HTML(segments[0]["text"]))
```

Table 1: Current layout detection models in the LayoutParser model zoo

| Dataset | Base Model1 | Large Model Notes |
|----------------|-------------|--|
| PubLayNet [38] | F/M | Layouts of modern scientific documents |
| | | Layouts of scanned |

| | | |
|-------------------|-----|---|
| PRImA | M | modern magazines and scientific reports |
| Newspaper | F | Layouts of scanned US newspaper from the 20th century |
| TableBank [18] | F | Table region on modern scientific and business document |
| HJDataset | F/M | Layouts of history Japanese documents |

Extracting text from specific sections

Structures may have parent-child relationships -- for example, a paragraph might belong to a section with a title. If a section is of particular interest (e.g., for indexing) we can isolate the corresponding `Document` objects.

Below, we extract all text associated with the document's "Conclusion" section:

```
render_page(docs, 14,  
print_text=False)
```

Text
Title

[1] Z. Shen et al.

6 Conclusion

LayoutParser provides a comprehensive toolkit for deep learning-based document image analysis. The off-the-shelf library is easy to install, and can be used to build flexible and accurate pipelines for processing documents with complicated structures. It also supports high-level customization and enables easy labeling and training of DL models on unique document image datasets. The **LayoutParser** community platform facilitates sharing DL models and DIA pipelines, inviting discussion and promoting code reproducibility and reusability. The **LayoutParser** team is committed to keeping the library updated continuously and bringing the most recent advances in DL-based DIA, such as multi-modal document modeling [37, 36, 9] (an upcoming priority), to a diverse audience of end-users.

Acknowledgements We thank the anonymous reviewers for their comments and suggestions. This project is supported in part by NSF Grant OIA-2033556 and funding from the Harvard Data Science Initiative and Harvard Catalyst. Zejiang Shen thanks Doug Downey for suggestions.

References

- [1] Alshadi, M., Agarwal, A., Barham, P., Bercks, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Joudewicz, R., Kaiser, L., Kudrhar, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olsh, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X. **TensorFlow: Large-scale machine learning on heterogeneous systems** (2015), <https://www.tensorflow.org/>, software available from tensorflow.org
- [2] Alberti, M., Pondekuschitz, V., Würrsch, M., Ingold, R., Lüscher, M. **DeepPipe: a highly-functional python framework for reproducible experiments**. In: 2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR). pp. 423-428. IEEE (2018)
- [3] Antonoglou, A., Reichen, D., Papadopoulos, C., Pletschacher, S. **A realistic dataset for performance evaluation of document layout analysis**. In: 2009 10th International Conference on Document Analysis and Recognition. pp. 296-300. IEEE (2009)
- [4] Bae, Y., Lee, B., Han, D., Yun, S., Lee, H. **Character region awareness for text detection**. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8365-8374 (2019)
- [5] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L. **ImageNet: A Large-Scale Hierarchical Image Database**. In: CVPR09 (2009)
- [6] Deng, Y., Kuznetsov, A., Ling, J., Kulkarni, A.M. **Image-to-marking generation with coarse-to-fine attention**. In: International Conference on Machine Learning. pp. 980-989. PMLR (2017)
- [7] Guo, Y., Lempitsky, V. **Unsupervised domain adaptation by backpropagation**. In: International conference on machine learning. pp. 1180-1189. PMLR (2015)

```
conclusion_docs = []
```

```
parent_id = -1
for doc in docs:
    if
doc.metadata["category"] ==
"Title" and "Conclusion" in
doc.page_content:
    parent_id =
doc.metadata["element_id"]
    if
doc.metadata.get("parent_id")
== parent_id:

conclusion_docs.append(doc)

for doc in conclusion_docs:
    print(doc.page_content)
```

LayoutParser provides a comprehensive toolkit for deep learning-based document image analysis. The off-the-shelf library is easy to install, and can be used to build flexible and accurate pipelines for processing documents with complicated structures. It also supports high-level customization and enables easy labeling and training of DL models on unique document image datasets. The LayoutParser community platform facilitates sharing DL models and DIA pipelines, inviting discussion and promoting code reproducibility and reusability. The

LayoutParser team is committed to keeping the library updated continuously and bringing the most recent advances in DL-based DIA, such as multi-modal document modeling [37, 36, 9] (an upcoming priority), to a diverse audience of end-users.

Acknowledgements We thank the anonymous reviewers for their comments and suggestions. This project is supported in part by NSF Grant OIA-2033558 and funding from the Harvard Data Science Initiative and Harvard Catalyst. Zejiang Shen thanks Doug Downey for suggestions.

Extracting text from images

OCR is run on images, enabling the extraction of text therein:

```
render_page(docs, 11)
```




LayoutParser: A Unified Toolkit for DL-Based DIA 21

focuses on precision, efficiency, and robustness. The target documents may have complicated structures, and may require training multiple layout detection models to achieve the optimal accuracy. Light-weight pipelines are built for relatively simple documents, with an emphasis on development ease, speed and flexibility. Ideally one only needs to use existing resources, and model training should be avoided. Through two exemplar projects, we show how practitioners in both academia and industry can easily build such pipelines using LayoutParser and extract high-quality structured document data for their downstream tasks. The source code for these projects will be publicly available in the LayoutParser community hub.

5.1 A Comprehensive Historical Document Digitization Pipeline

The digitization of historical documents can unlock valuable data that can shed light on many important social, economic, and historical questions. Yet due to scan noises, page wearing, and the prevalence of complicated layout structures, obtaining a structured representation of historical document scans is often extremely complicated.

In this example, LayoutParser was used to develop a comprehensive pipeline, shown in Figure 5, to generate high-quality structured data from historical Japanese firm financial tables with complicated layouts. The pipeline applies two layout models to identify different levels of document structures and two customized OCR engines for optimized character recognition accuracy.

As shown in Figure 4 (a), the document contains columns of text written vertically¹⁸, a common style in Japanese. Due to scanning noise and archaic printing technology, the columns can be skewed or have variable widths, and hence cannot be easily identified via rule-based methods. Within each column, words are separated by white spaces of variable size, and the vertical positions of objects can be an indicator of their layout type.



Fig. 5: Illustration of how LayoutParser helps with the historical document digitization pipeline.

¹⁸ A document page consists of eight rows like this. For simplicity we skip the row segmentation discussion and refer readers to the source code when available.

LayoutParser: A Unified Toolkit for DL-Based DIA

focuses on precision, efficiency, and robustness. The target documents may have complicated structures, and may require training multiple layout detection models to achieve the optimal accuracy. Light-weight pipelines are built for relatively simple documents, with an emphasis on development ease, speed and flexibility. Ideally one only needs to use existing resources, and model training should be avoided. Through two exemplar projects, we show how practitioners in both

academia and industry can easily build such pipelines using LayoutParser and extract high-quality structured document data for their downstream tasks. The source code for these projects will be publicly available in the LayoutParser community hub.

11

5.1 A Comprehensive Historical Document Digitization Pipeline

The digitization of historical documents can unlock valuable data that can shed light on many important social, economic, and historical questions. Yet due to scan noises, page wearing, and the prevalence of complicated layout structures, obtaining a structured representation of historical document scans is often extremely complicated. In this example, LayoutParser was used to develop a comprehensive pipeline, shown in Figure 5, to generate high-quality structured data from historical Japanese firm financial tables with complicated layouts. The

pipeline applies two layout models to identify different levels of document structures and two customized OCR engines for optimized character recognition accuracy.

'Active Learning Layout
Annotate Layout Dataset | +
— Annotation Toolkit A4
Deep Learning Layout Layout
Detection Model Training &
Inference, A Post-
processing – Handy Data
Structures & \ Lo orajport
7) Al Pls for Layout Data
A4 Default and Customized
Text Recognition OCR Models
¥ Visualization & Export
Layout Structure
Visualization & Storage The
Japanese Document Helpful
LayoutParser Modules
Digitization Pipeline

As shown in Figure 4 (a), the document contains columns of text written vertically 15, a common style in Japanese. Due to scanning noise and archaic printing technology, the columns can be skewed or have variable widths, and hence cannot be easily identified via rule-based methods. Within each column, words are separated by white spaces of variable size, and the

vertical positions of objects can be an indicator of their layout type.

Fig. 5: Illustration of how LayoutParser helps with the historical document digitization pipeline.

15 A document page consists of eight rows like this. For simplicity we skip the row segmentation discussion and refer readers to the source code when available.

Note that the text from the figure on the right is extracted and incorporated into the content of the `Document`.

Local parsing

Parsing locally requires the installation of additional dependencies.

Poppler (PDF analysis)

- Linux: `apt-get install poppler-utils`
- Mac: `brew install poppler`
- Windows:
<https://github.com/oschwartz10612/poppler-windows>

Tesseract (OCR)

- Linux: `apt-get install tesseract-ocr`
- Mac: `brew install tesseract`
- Windows: <https://github.com/UB-Mannheim/tesseract/wiki#tesseract-installer-for-windows>

We will also need to install the `unstructured` PDF extras:

```
%pip install -qU  
"unstructured[pdf]"
```

We can then use the `UnstructuredLoader` much the same way, forgoing the API key and `partition_via_api` setting:

```
loader_local =  
UnstructuredLoader(  
    file_path=file_path,  
    strategy="hi_res",  
)  
docs_local = []  
for doc in  
    loader_local.lazy_load():  
        docs_local.append(doc)
```

WARNING: This function will be

[illegible]

The list of documents can then be processed similarly to those obtained from the API.

Use of multimodal models

Many modern LLMs support inference over multimodal inputs (e.g., images). In some applications--such as question-answering over PDFs with complex layouts, diagrams, or scans--it may be advantageous to skip the PDF parsing, instead casting a PDF page to an image and passing it to a model directly. This allows a model to reason over the two dimensional content on the page, instead of a "one-dimensional" string representation.

In principle we can use any LangChain [chat model](#) that supports multimodal inputs. A list of these models is documented [here](#). Below we use OpenAI's `gpt-4o-mini`.

First we define a short utility function to convert a PDF page to a base64-encoded image:

```
%pip install -qU PyMuPDF
pillow langchain-openai
```

```
import base64
import io

import fitz
from PIL import Image

def pdf_page_to_base64(pdf_page: fitz.Page,
                        page_number: int):
    pdf_document = fitz.open(pdf_page.pdf)
    page = pdf_document.load_page(page_number - 1)
    # input is one-indexed
    pix = page.get_pixmap()
    img = Image.frombytes("RGB",
                          [pix.width, pix.height],
                          pix.samples)

    buffer = io.BytesIO()
    img.save(buffer, format="png")

    return
    base64.b64encode(buffer.getvalue())
```

```
from IPython.display import Image
from IPython.display import data_url

base64_image = pdf_page_to_base64(pdf_page, page_number)
display(Image(data=base64_image))
```


focuses on precision, efficiency, and robustness. The target documents may have complicated structures, and may require training multiple layout detection models to achieve the optimal accuracy. Light-weight pipelines are built for relatively simple documents, with an emphasis on development ease, speed and flexibility. Ideally one only needs to use existing resources, and model training should be avoided. Through two exemplar projects, we show how practitioners in both academia and industry can easily build such pipelines using LayoutParser and extract high-quality structured document data for their downstream tasks. The source code for these projects will be publicly available in the LayoutParser community hub.

5.1 A Comprehensive Historical Document Digitization Pipeline

The digitization of historical documents can unlock valuable data that can shed light on many important social, economic, and historical questions. Yet due to scan noises, page wearing, and the prevalence of complicated layout structures, obtaining a structured representation of historical document scans is often extremely complicated.

In this example, LayoutParser was used to develop a comprehensive pipeline, shown in Figure 5, to generate high-quality structured data from historical Japanese firm financial tables with complicated layouts. The pipeline applies two layout models to identify different levels of document structures and two customized OCR engines for optimized character recognition accuracy.

As shown in Figure 4 (a), the document contains columns of text written vertically¹⁵, a common style in Japanese. Due to scanning noise and archaic printing technology, the columns can be skewed or have variable widths, and hence cannot be easily identified via rule-based methods. Within each column, words are separated by white spaces of variable size, and the vertical positions of objects can be an indicator of their layout type.

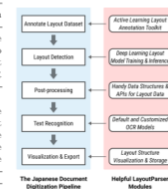


Fig. 5: Illustration of how LayoutParser helps with the historical document digitization pipeline.

¹⁵ A document page consists of eight rows like this. For simplicity we skip the row segmentation discussion and refer readers to the source code when available.

We can then query the model in the **usual way**. Below we ask it a question on related to the diagram on the page.

```
from langchain_openai
import ChatOpenAI

llm =
ChatOpenAI(model="gpt-4o-
mini")
```

```
from langchain_core.messages
HumanMessage

query = "What is the name of
step in the pipeline?"

message = HumanMessage(
    content=[
```

```
        {"type": "text", "text": "The first step in the pipeline is to\n        create a dataset of PDF documents."},\n        {\n            "type": "image_url",\n            "image_url": {"url": f"data:image/jpeg;base64,{base64_image}"},\n        },\n    ],\n)\n\nresponse = llm.invoke([message])\nprint(response.content)
```

API Reference: [HumanMessage](#)

```
INFO: HTTP Request: POST\nhttps://api.openai.com/v1/chat/completions\n"HTTP/1.1 200 OK"\n```\noutput\nThe first step in the pipeline is to\ncreate a dataset of PDF documents.
```

Other PDF loaders

For a list of available LangChain PDF loaders, please see [this table](#).

 [Edit this page](#)