




---

# Docling Technical Report

---

Version 1.0

**Christoph Auer   Maksym Lysak   Ahmed Nassar   Michele Dolfi   Nikolaos Livathinos  
Panos Vagenas   Cesar Berrospi Ramis   Matteo Omenetti   Fabian Lindlbauer  
Kasper Dinkla   Lokesh Mishra   Yusik Kim   Shubham Gupta   Rafael Teixeira de Lima  
Valery Weber   Lucas Morin   Ingmar Meijer   Viktor Kuropiatnyk   Peter W. J. Staar**

AI4K Group, IBM Research  
Rüschlikon, Switzerland

## Abstract

This technical report introduces *Docling*, an easy to use, self-contained, MIT-licensed open-source package for PDF document conversion. It is powered by state-of-the-art specialized AI models for layout analysis (DocLayNet) and table structure recognition (TableFormer), and runs efficiently on commodity hardware in a small resource budget. The code interface allows for easy extensibility and addition of new features and models.

## 1 Introduction

Converting PDF documents back into a machine-processable format has been a major challenge for decades due to their huge variability in formats, weak standardization and printing-optimized characteristic, which discards most structural features and metadata. With the advent of LLMs and popular application patterns such as retrieval-augmented generation (RAG), leveraging the rich content embedded in PDFs has become ever more relevant. In the past decade, several powerful document understanding solutions have emerged on the market, most of which are commercial software, cloud offerings [3] and most recently, multi-modal vision-language models. As of today, only a handful of open-source tools cover PDF conversion, leaving a significant feature and quality gap to proprietary solutions.

With *Docling*, we open-source a very capable and efficient document conversion tool which builds on the powerful, specialized AI models and datasets for layout analysis and table structure recognition we developed and presented in the recent past [12, 13, 9]. *Docling* is designed as a simple, self-contained python library with permissive license, running entirely locally on commodity hardware. Its code architecture allows for easy extensibility and addition of new features and models.

Here is what Docling delivers today:

- Converts PDF documents to JSON or Markdown format, stable and lightning fast
- Understands detailed page layout, reading order, locates figures and recovers table structures
- Extracts metadata from the document, such as title, authors, references and language
- Optionally applies OCR, e.g. for scanned PDFs
- Can be configured to be optimal for batch-mode (i.e high throughput, low time-to-solution) or interactive mode (compromise on efficiency, low time-to-solution)
- Can leverage different accelerators (GPU, MPS, etc).

## 2 Getting Started

To use Docling, you can simply install the [docling](#) package from PyPI. Documentation and examples are available in our [GitHub](#) repository at [github.com/DS4SD/docling](#). All required model assets<sup>1</sup> are downloaded to a local huggingface datasets cache on first use, unless you choose to pre-install the model assets in advance.

Docling provides an easy code interface to convert PDF documents from file system, URLs or binary streams, and retrieve the output in either JSON or Markdown format. For convenience, separate methods are offered to convert single documents or batches of documents. A basic usage example is illustrated below. Further examples are available in the Docling code repository.

```
from docling.document_converter import DocumentConverter

source = "https://arxiv.org/pdf/2206.01062" # PDF path or URL
converter = DocumentConverter()
result = converter.convert_single(source)
print(result.render_as_markdown()) # output: "## DocLayNet: A Large
    Human-Annotated Dataset for Document-Layout Analysis [...]"
```

Optionally, you can configure custom pipeline features and runtime options, such as turning on or off features (e.g. OCR, table structure recognition), enforcing limits on the input document size, and defining the budget of CPU threads. Advanced usage examples and options are documented in the README file. Docling also provides a *Dockerfile* to demonstrate how to install and run it inside a container.

## 3 Processing pipeline

Docling implements a linear pipeline of operations, which execute sequentially on each given document (see Fig. 1). Each document is first parsed by a PDF backend, which retrieves the programmatic text tokens, consisting of string content and its coordinates on the page, and also renders a bitmap image of each page to support downstream operations. Then, the standard model pipeline applies a sequence of AI models independently on every page in the document to extract features and content, such as layout and table structures. Finally, the results from all pages are aggregated and passed through a post-processing stage, which augments metadata, detects the document language, infers reading-order and eventually assembles a typed document object which can be serialized to JSON or Markdown.

### 3.1 PDF backends

Two basic requirements to process PDF documents in our pipeline are a) to retrieve all text content and their geometric coordinates on each page and b) to render the visual representation of each page as it would appear in a PDF viewer. Both these requirements are encapsulated in Docling's PDF backend interface. While there are several open-source PDF parsing libraries available for python, we faced major obstacles with all of them for different reasons, among which were restrictive

---

<sup>1</sup>see [huggingface.co/ds4sd/docling-models/](https://huggingface.co/ds4sd/docling-models/)

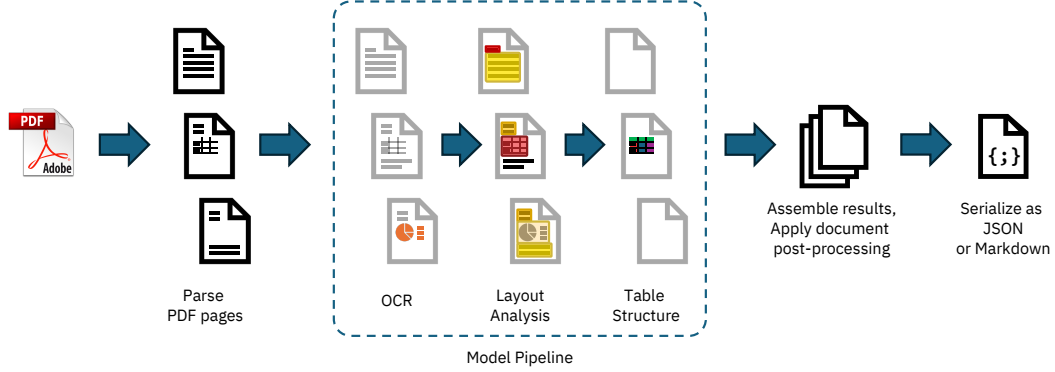


Figure 1: Sketch of Docling’s default processing pipeline. The inner part of the model pipeline is easily customizable and extensible.

licensing (e.g. `pymupdf` [7]), poor speed or unrecoverable quality issues, such as merged text cells across far-apart text tokens or table columns (`pypdfium`, `PyPDF`) [15, 14].

We therefore decided to provide multiple backend choices, and additionally open-source a custom-built PDF parser, which is based on the low-level `qpdf`[4] library. It is made available in a separate package named `docling-parse` and powers the default PDF backend in Docling. As an alternative, we provide a PDF backend relying on `pypdfium`, which may be a safe backup choice in certain cases, e.g. if issues are seen with particular font encodings.

### 3.2 AI models

As part of Docling, we initially release two highly capable AI models to the open-source community, which have been developed and published recently by our team. The first model is a layout analysis model, an accurate object-detector for page elements [13]. The second model is TableFormer [12, 9], a state-of-the-art table structure recognition model. We provide the pre-trained weights (hosted on [huggingface](#)) and a separate package for the inference code as `docling-ibm-models`. Both models are also powering the open-access `deepsearch-experience`, our cloud-native service for knowledge exploration tasks.

#### Layout Analysis Model

Our layout analysis model is an object-detector which predicts the bounding-boxes and classes of various elements on the image of a given page. Its architecture is derived from RT-DETR [16] and re-trained on DocLayNet [13], our popular human-annotated dataset for document-layout analysis, among other proprietary datasets. For inference, our implementation relies on the `onnxruntime` [5].

The Docling pipeline feeds page images at 72 dpi resolution, which can be processed on a single CPU with sub-second latency. All predicted bounding-box proposals for document elements are post-processed to remove overlapping proposals based on confidence and size, and then intersected with the text tokens in the PDF to group them into meaningful and complete units such as paragraphs, section titles, list items, captions, figures or tables.

#### Table Structure Recognition

The TableFormer model [12], first published in 2022 and since refined with a custom structure token language [9], is a vision-transformer model for table structure recovery. It can predict the logical row and column structure of a given table based on an input image, and determine which table cells belong to column headers, row headers or the table body. Compared to earlier approaches, TableFormer handles many characteristics of tables, such as partial or no borderlines, empty cells, rows or columns, cell spans and hierarchy both on column-heading or row-heading level, tables with inconsistent indentation or alignment and other complexities. For inference, our implementation relies on `PyTorch` [2].