Parsing Complex Documents for Free

A Deep Dive into Docling and its RAG Capabilities

5 min read · Just now

Yogesh Haribhau Kulkarni (PhD)

Photo by Annie Spratt on Unsplash

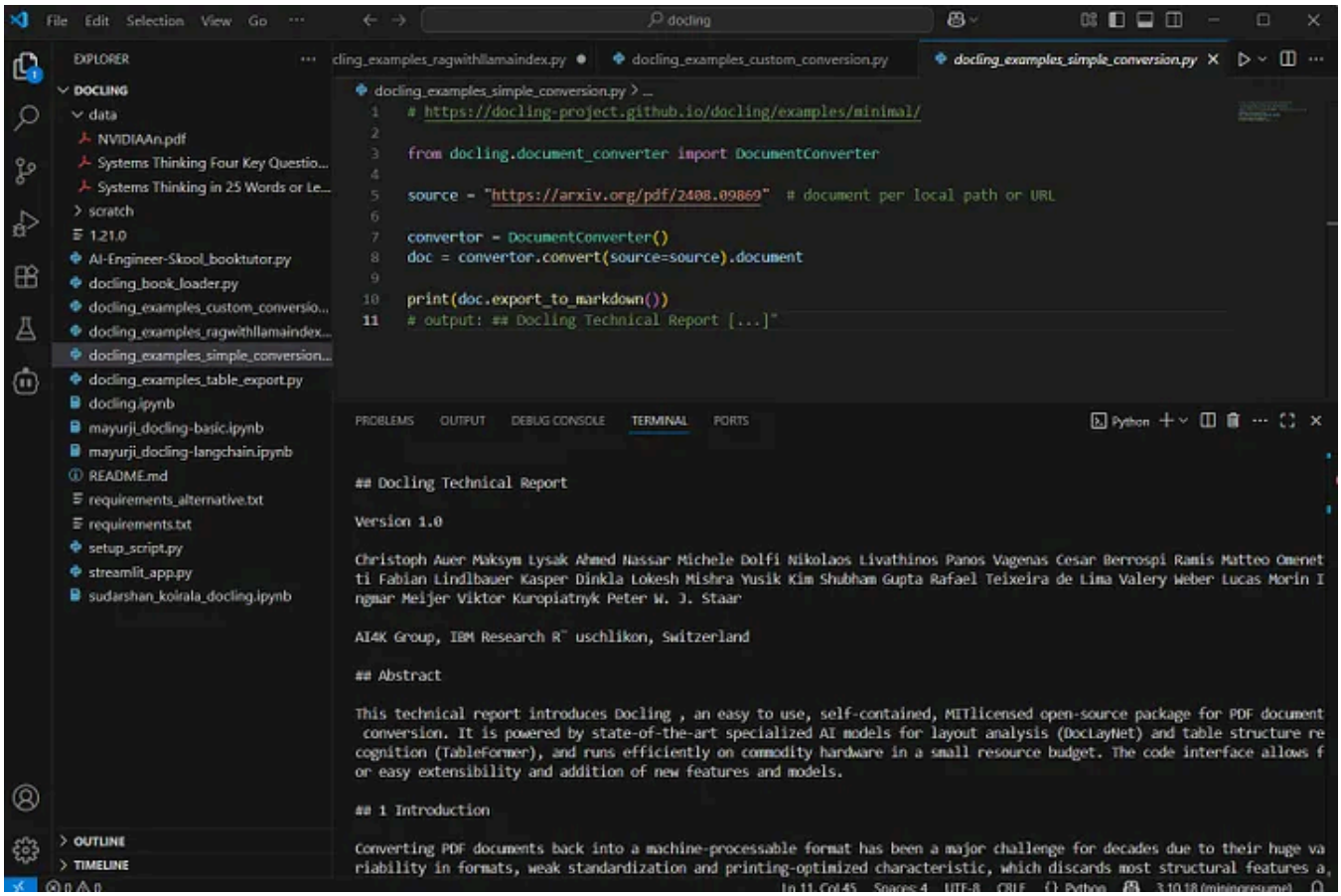
The ability to efficiently and accurately process complex documents is paramount in the age of large language models (LLMs) and AI-driven workflows. For non-trivial, enterprise-level LLM applications, Retrieval-Augmented Generation (RAG) on complex documents is a common scenario. The quality of the conversion from documents with complex layouts, tables, pictures, etc., to a processable text format is key to achieving good output. Docling is a powerful tool designed to simplify document processing across a multitude of formats, enabling seamless integration with the generative AI ecosystem. This blog post explores the architecture, key features, and practical applications of Docling, including a detailed look at how it can be used to build a sophisticated RAG system.

Core Features and Architecture

Docling boasts a rich feature set that makes it a standout choice for document AI tasks.

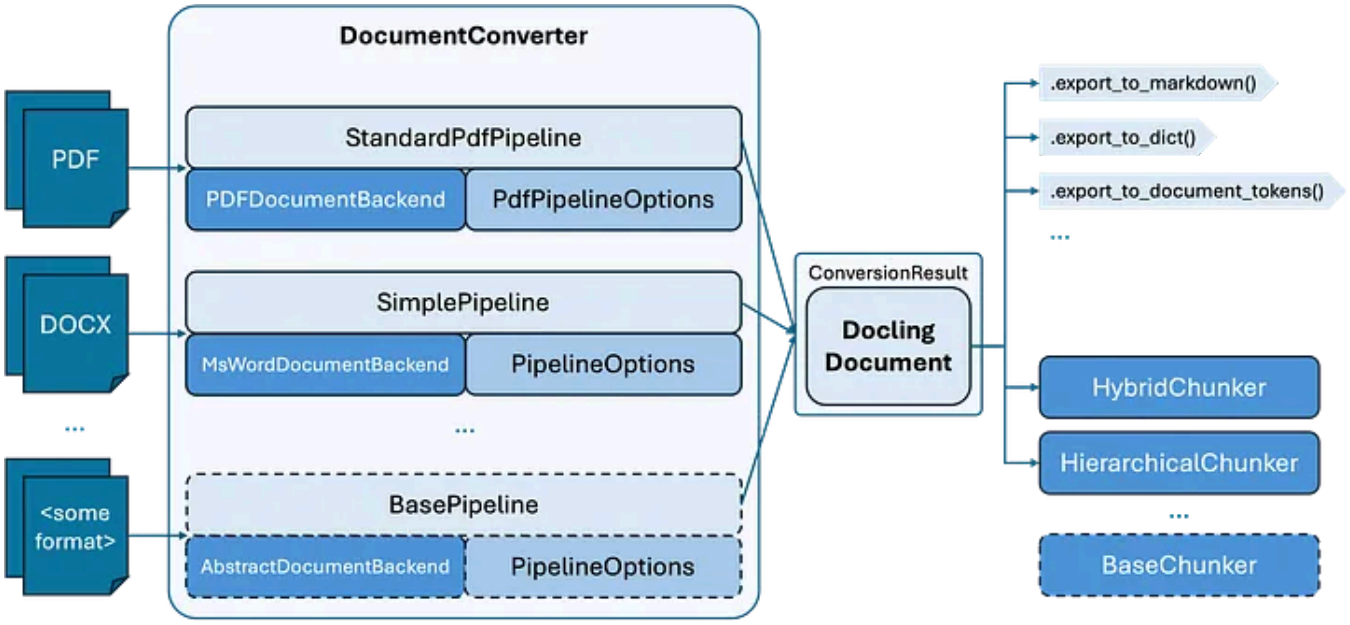
Key Features at a Glance

- **Multi-format Parsing**: Docling can parse a wide array of file types, including PDF, DOCX, PPTX, XLSX, HTML, and even audio and image files.
- **Advanced PDF Understanding**: It intelligently interprets PDF layouts, reading order, tables, and formulas, a critical feature for complex documents.
- **Unified Representation**: All processed documents are converted into a standardized `DoclingDocument` format, which simplifies downstream tasks.
- **Flexible Export Options**: Documents can be exported to Markdown, HTML, DocTags, and JSON, offering versatility for different use cases.
- **Local Execution**: For sensitive data or air-gapped environments, Docling can be run locally, ensuring data privacy.
- **Plug-and-Play Integrations**: It integrates smoothly with popular AI frameworks like LangChain, LlamaIndex, Crew AI, and Haystack.
- **Advanced OCR and ASR**: Docling includes extensive OCR support for scanned documents and Automatic Speech Recognition for audio files13.



(Author's own screenshot of Docling trials)

Architectural Overview



(Source: Docling documentation)

Docling's architecture is designed for modularity and efficiency. A document converter selects the appropriate backend for a given file format, and a pipeline orchestrates the conversion process. The result is a `DoclingDocument`, which can then be exported or segmented using serializers and chunkers, respectively.

At the heart of Docling is the `DoclingDocument`, a Pydantic datatype that provides a unified representation for all documents. This structure expresses text, tables, pictures, and hierarchical sections, distinguishing between the main body and "furniture" like headers and footers15. It maintains provenance information for traceability and includes layout details with bounding boxes, enabling structured analysis and processing.

The content is organized into categories:

- **Content Items**: These include text, tables, pictures, and key-value pairs.
- **Content Structure**: This comprises the main body, furniture, and groups for organizing content.

Reading order is preserved through a hierarchical tree structure, with items nested under parent elements and ordered sequentially.

Chunking and Serialization: Preparing Data for AI

For AI applications like RAG, documents need to be broken down into smaller, meaningful segments or “chunks.” Docling’s **Chunking Framework** abstracts this process, allowing for flexible segmentation strategies. It offers two primary implementations:

1. **Hierarchical Chunker**: This method uses the document’s inherent structure for element-based chunking.
2. **Hybrid Chunker**: This approach refines hierarchical chunks by splitting oversized ones based on token limits and merging undersized ones, making it highly effective for preparing data for LLMs.

Here’s a Python snippet demonstrating the `HybridChunker`:

```
from docling.document_converter import DocumentConverter
from docling.chunking import HybridChunker

doc = DocumentConverter().convert(source=DOC_SOURCE).document
chunker = HybridChunker()
chunk_iter = chunker.chunk(dl_doc=doc)
for i, chunk in enumerate(chunk_iter):
    print(f"=== {i} ===")
    print(f"chunk.text:\n{f'{chunk.text[:300]}'!r}")
    enriched_text = chunker.contextualize(chunk=chunk)
```

```
        print(f"chunker.contextualize(chunk):\n{f'{enriched_text[:300]}'!r}")
        print()
```

Similarly, the **Serialization Framework** converts `DoclingDocument` objects into textual formats like Markdown or HTML. It features a provider model that allows for custom serialization strategies, such as specifying a different table serializer.

Use Case: Building a RAG System with Docling and LlamaIndex

One of the most powerful applications of Docling is in building RAG systems. This example demonstrates how to create a RAG pipeline using Docling, LlamaIndex, ChromaDB, and HuggingFace models.

The pipeline performs the following steps:

- **Document Processing:** Docling intelligently parses a complex PDF, preserving its structure and metadata.
- **Node Parsing:** The `MarkdownNodeParser` from LlamaIndex converts the document into semantic chunks or nodes.
- **Vector Storage:** The text chunks are converted into vector embeddings using a HuggingFace model (`BAAI/bge-small-en-v1.5`) and stored in a ChromaDB vector store.
- **Indexing:** A `VectorStoreIndex` is created from the documents, making them searchable.
- **Querying:** A query engine, powered by a local language model (`microsoft/DialoGPT-small`), performs a similarity search to find relevant document chunks and generate a coherent response.

Here is the core code for building the searchable knowledge base:

```
# Setup the reader and parser
reader = DoclingReader()
node_parser = MarkdownNodeParser()
# Create the vector index
index = VectorStoreIndex.from_documents(
    documents=reader.load_data(SOURCE),
    transformations=[node_parser],
    storage_context=StorageContext.from_defaults(
```

```
            vector_store=vector_store
        ),
        embed_model=EMBED_MODEL,
    )

    result = index.as_query_engine(llm=GEN_MODEL).query(QUERY)
    print(f"Q: {QUERY}")
    print(f"A: {result.response.strip()}")
    print(f"\nSources:")
    print([(n.text, n.metadata) for n in result.source_nodes])
```

This architecture enables the system to answer specific questions by retrieving relevant passages from the source document and generating accurate, context-aware responses.

Extensibility and Confidence

Docling is also designed to be extensible through a **plugin system**, allowing developers to add new capabilities, such as different OCR engines. This is managed via the `pluggy` system with setuptools entry points. Furthermore, Docling provides **confidence scores** to assess the quality of document conversion. These scores, ranging from 0.0 to 1.0, help identify documents that may require manual review and allow for the adjustment of conversion pipelines. The scores are broken down into components like layout, OCR, and table extraction quality, providing a granular view of the conversion process.

Conclusion

Docling stands out as a robust and flexible solution for modern document processing challenges. Its ability to handle complex formats, preserve document structure, and seamlessly integrate with the generative AI ecosystem makes it an invaluable tool for developers building RAG systems, content management solutions, and other document-intensive applications36. By providing a unified representation, advanced chunking and serialization, and transparent quality assessment, Docling empowers organizations to unlock the full potential of their unstructured data.

References

Docling

Docling simplifies document processing, parsing diverse formats - including advanced PDF understanding - and providing...

docling-project.github.io

Docling | Document Parser | Artificial Intelligence | Retrieval Augmented Gen | Llamaindex

Following

Published in Technology Hits

3.6K followers · Last published just now

We cover important, high-impact, informative, and engaging stories on all aspects of technology. Subscribe to our content marketing strategy newsletter: https://drmehmetyildiz.substack.com/ Writer inquiries: https://digitialmehmet.com/contact

Edit profile

Written by Yogesh Haribhau Kulkarni (PhD)

1.8K followers · 2.1K following

PhD in Geometric Modeling | Google Developer Expert (Machine Learning) | Top Writer 3x (Medium) | More at https://www.linkedin.com/in/yogeshkulkarni/

No responses yet

Yogesh Haribhau Kulkarni (PhD)

What are your thoughts?

More from Yogesh Haribhau Kulkarni (PhD) and Technology Hits

In ILLUMINATION Videos and Podcasts by Yogesh Haribhau Kulkarni (PhD)

The Equation of True Happiness

Based on talks by Arthur C Brooks

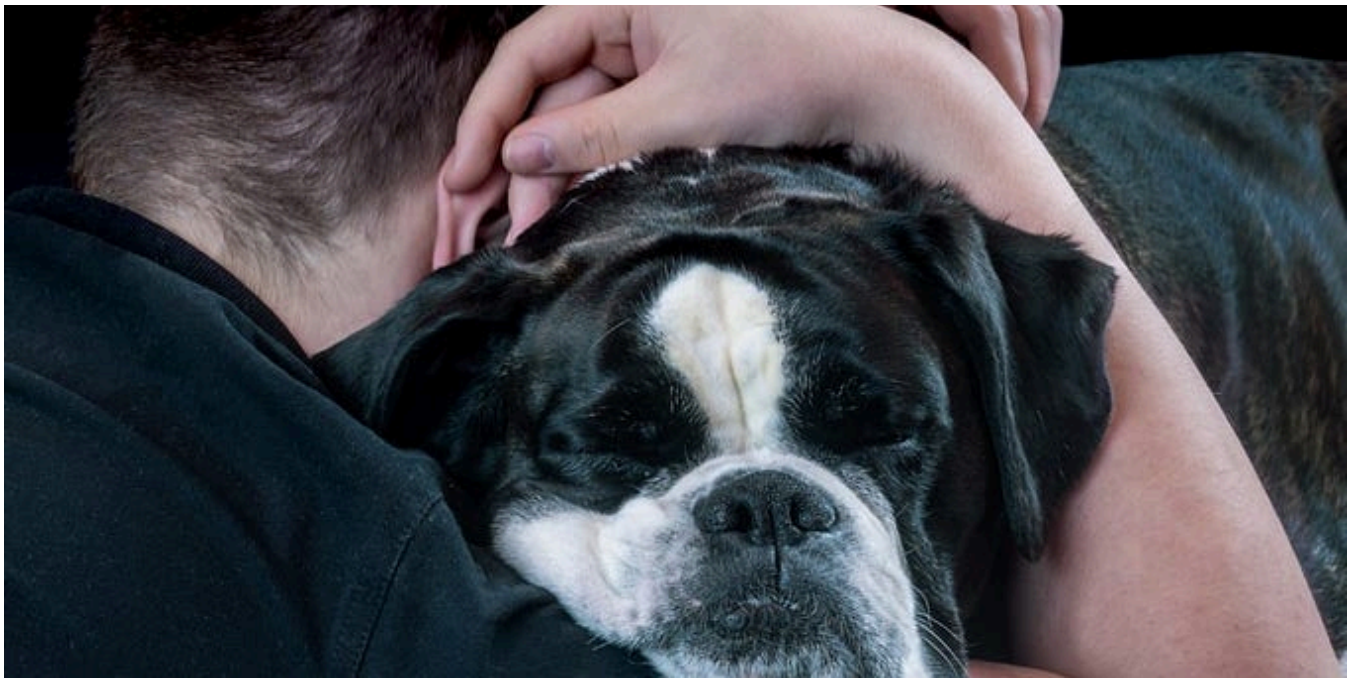
Sep 17, 2023 164 1

In ILLUMINATION Videos and Podcasts by Yogesh Haribhau Kulkarni (PhD)

Core concepts of Bhagavad Gita

Based on interview of Keshava Swami by Ranveer Allahbadia

Nov 26, 2023 116 1



In Technology Hits by Yogesh Haribhau Kulkarni (PhD)

Transformation by Hugging Face

Are you lost in the storm of these BERTs ie ALBERT, DistilBERT, RoBERTa etc? And these GPTs (1–2–3)? Don't understand how they work? What...

Nov 21, 2022 5

In Technology Hits by Yogesh Haribhau Kulkarni (PhD)

Specs for 'Chatbot on Knowledge Graph using Large Language Models'

Product Requirements Document with hints of Implementation

Dec 8, 2023 8

See all from Yogesh Haribhau Kulkarni (PhD)

See all from Technology Hits

Recommended from Medium

In Coding Nexus by Code Coup

Why JSON Prompts Make AI More Reliable (With Code & Real Examples)

Dealing with AI can be a rollercoaster.

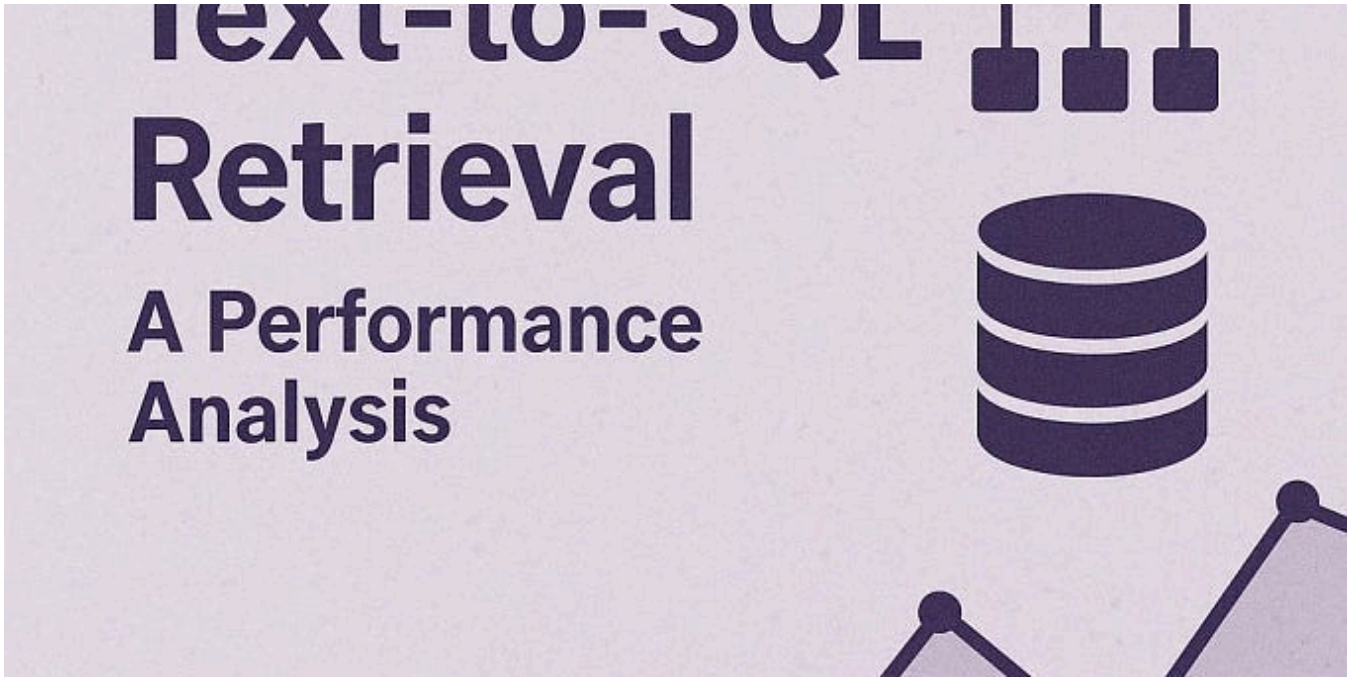
5d ago 136 1

Pankaj

Agentic-Doc: Extract Structured Data from Complex Documents with Ease

agestic-doc is a powerful Python library that extracts structured, visually grounded data from complex documents like PDFs and images...

6d ago 49



Tamanna

Optimizing Text-to-SQL Retrieval

Text-to-SQL is a game-changer in how we interact with databases. It lets people ask questions in plain English (or other natural languages)...

Jul 15 201 1

In Level Up Coding by Gaurav Shrivastav

RAG Without Embeddings? Here's how OpenAI is doing this...

OpenAI's new RAG method ditches embeddings—LLMs now read like humans, navigate docs smartly, and give verified, grounded answers!

Jun 11 231 3

In Towards AI by Youssef Hosni

Building Local OCR Application SmolDocling: A Step-by-Step Guide [Part 1]

If you want to build a Local OCR application such that you do not need to send your documents to APIs. You can use **SmolDocling**...

Mar 30 416 1