Parsing Complex Documents for RAG with Docling

Yogesh Haribhau Kulkarni

YHK

Outline

About Me

Yogesh Haribhau Kulkarni

Bio:

- 20+ years in CAD/Engineering software development
- Got Bachelors, Masters and Doctoral degrees in Mechanical Engineering (specialization: Geometric Modeling Algorithms).
- Currently doing Coaching in fields such as Data Science, Artificial Intelligence Machine-Deep Learning (ML/DL) and Natural Language Processing (NLP).
- Feel free to follow me at:
  - Github (github.com/yogeshhk)
  - LinkedIn (www.linkedin.com/in/yogeshkulkarni/)
  - Medium (yogeshharibhaukulkarni.medium.com)
  - Send email to yogeshkulkarni at yahoo dot com

Office Hours: Saturdays, 2 to 5pm (IST); Free-Open to all; email for appointment.

YHK

RAG Parsing Introduction

YHK

Executive Summary

- Document parsing transforms unstructured data into structured, machine-readable form.
- It is foundational for RAG (Retrieval-Augmented Generation) systems.
- Parsing quality directly affects downstream retrieval and generation accuracy.
- Poor parsing causes hallucinations and retrieval errors.
- Docling, by IBM Research, represents a modern open-source parsing framework.
- This report explores parsing theory, current methods, and Docling's architecture.

YHK

Background and Need for Document Parsing

- Evolved from OCR, which focused only on character recognition.
- Early OCR failed with complex layouts, images, and poor scans.
- Modern LLMs require semantically rich document structures.
- Traditional text extraction leads to “garbage in, garbage out”.
- Parsing is essential for contextualized, structured inputs to RAG systems.

YHK

RAG Pipeline and Parsing Role

- RAG pipeline steps: Ingestion → Chunking → Embedding → Indexing → Retrieval → Generation.
- Parsing defines quality of all downstream stages.
- Poor parsing leads to loss of information and degraded retrieval.
- Incorrect structure causes hallucinations during generation.
- Accurate parsing enhances semantic coherence and query relevance.

YHK

Document Parsing: Historical Evolution

- Early OCR (1970s-1990s): Character-level recognition only.
- Limited to simple, clean documents with uniform layouts.
- Struggled with: multi-column layouts, mixed content, variable quality scans.
- Modern era (2020s): LLM-driven demand for semantic understanding.
- Shift from text extraction to structural comprehension.
- Integration with AI pipelines requires context preservation.

YHK

Impact of Parsing Quality on RAG

- **Information Loss:** Tables and visual layouts become inaccessible.
- **Context Degradation:** Reading order confusion breaks semantic flow.
- **Downstream Hallucinations:** LLMs fabricate info from incomplete context.
- **Retrieval Failures:** Vector search fails on poorly extracted chunks.
- **Critical Finding:** Distracting documents degrade performance more than irrelevant ones.
- Parsing is the *first line of defense* against RAG failures.

YHK

Document Layers: Multi-Dimensional Understanding

- **Spatial Layer:** Physical coordinates and bounding boxes.
- **Logical Layer:** Semantic hierarchy (sections, relationships).
- **Visual Layer:** Styling information (fonts, colors, emphasis).
- **Textual Layer:** Raw character sequences.
- Modern parsers must integrate all layers for accurate representation.
- Single-layer extraction (text-only) loses critical context.

YHK

Reading Order: The Critical Challenge

- PDFs store visual representations, not logical flow.
- Reading order must be *inferred* from layout.
- Challenges: multi-column, floating elements, tables.
- Algorithms use spatial clustering and visual cues.
- Incorrect order produces incoherent chunks.

YHK

Chunking: Semantic Preservation Strategies

- **Fixed-size:** Simple, 512/1024 tokens; ignores semantic boundaries.
- **Sentence-based:** Respects grammar but may split cohesive ideas.
- **Semantic:** Uses embedding similarity to group related content.
- **Hierarchical:** Preserves document structure (sections, subsections).
- **Best Practice:** Structure-aware chunking maintains context.
- Poor chunking = disconnected information for retrieval.

YHK

Parsing Approaches: Modular vs End-to-End

- **Modular Pipelines:**
  - Sequential stages: OCR → Layout → Structure → Assembly.
  - Component-level optimization and testing.
  - Requires careful integration.
- **End-to-End VLMs:**
  - Single model processes entire page.
  - Eliminates integration complexity.
  - Higher computation, potential generality trade-off.
- **Hybrid (Docling):** Specialized models + flexible orchestration.

YHK

Technical Challenges: Context Windows

- Vision-Language Models have finite attention spans.
- High-resolution A4 pages exceed typical context limits.
- Models process small patches, losing global context.
- 50-page documents with cross-page relationships are problematic.
- Solutions: sophisticated chunking, multi-pass processing.
- Advanced platforms handle multi-page context automatically.

YHK

Performance Challenges: Speed vs Accuracy

- Complex documents are time-consuming (1-3 sec/page typical).
- Vision models require GPU acceleration (3-33x speedup).
- Resource requirements: 6-18 GB VRAM for high-accuracy models.
- Specialized models excel but struggle with edge cases.
- General models handle diversity but sacrifice accuracy.
- Trade-off: speed (egret-m) vs accuracy (heron-101).

YHK

Semantic Challenges: Fragmentation & Hallucination

- **Information Fragmentation:** Captions separated from figures.
- Retrieval fails when context is disconnected.
- **Downstream Hallucinations:** LLMs generate false info from incomplete input.
- Not an LLM failure—a parsing failure.
- **Ambiguity:** Handwritten notes, artistic layouts lack clear structure.
- Deterministic algorithms struggle with subjective interpretations.

YHK

Cloud vs Open-Source: Trade-offs

Aspect	Cloud-Based	Open-Source
Cost	Per-page fees ($0.03+)	Infrastructure only
Privacy	Data transmitted	Local processing
Customization	Limited	Full control
Scalability	Automatic	Manual setup
Vendor Lock-in	High	None
Setup	Easy	Moderate-High

- Cloud: prototyping, bursty workloads, no infrastructure.
- Open-Source: production, sensitive data, cost control.

YHK

When to Use Each Approach

- **Cloud-Based:**
  - Prototyping and proof-of-concepts.
  - Unpredictable, bursty document volumes.
  - No infrastructure expertise available.
- **Open-Source (Docling):**
  - Production systems with consistent volume.
  - Sensitive/proprietary documents (legal, medical, financial).
  - Air-gapped or offline requirements.
  - Cost-constrained at scale (millions of docs).
  - Full customization and auditability needed.

YHK

Document Layout Understanding

- Modern parsing identifies spatial, logical, visual, and textual layers.
- Layout analysis uses computer vision to classify elements.
- Elements: headers, body, tables, figures, captions, and footers.
- Enables structured, semantically meaningful document representations.

YHK

Information Structure Types

- Linear: sequential paragraphs and text.
- Tabular: structured grid data.
- Visual: charts, diagrams, images.
- Mathematical: equations and formulas.
- Hierarchical: sections and subsections.
- Effective parsing must handle all these content forms.

YHK

Reading Order Reconstruction

- PDFs lack explicit reading order; it must be inferred.
- Algorithms analyze spatial proximity, alignment, and indentation.
- Handle multi-column, nested, or cross-page content.
- Ensure logical, coherent flow for retrieval and chunking.

YHK

Chunking Strategies

- Fixed-size: simple but ignores semantics.
- Sentence-based: respects grammar but splits ideas.
- Semantic: uses embeddings for cohesive grouping.
- Hierarchical: preserves document structure.
- Docling enables native, structure-aware chunking.

YHK

Approaches to Document Parsing

- Modular pipelines: sequential OCR, layout, structure, post-processing.
- End-to-end VLMs: unified model predicting full structure.
- Hybrid systems: mix specialized and orchestration models.
- Docling employs modular yet flexible hybrid pipelines.

YHK

Challenges in Document Parsing

- Context window limits in vision-language models.
- Complex, dense, and multi-language layouts.
- Cross-page relationships and multimodal integration.
- Speed, GPU demand, and accuracy trade-offs.
- Hallucinations from fragmented or incorrect parsing.

YHK

Parsing Solutions: Cloud vs Open-Source

- Cloud-based: easy to scale, but has privacy and cost issues.
- Open-source: full control, no vendor lock-in, local execution.
- Hybrid: combines flexibility with enterprise reliability.
- Docling fits open-source/enterprise hybrid model.

YHK

RAG Pipeline: Complete Workflow

- **Stage 1 - Document Ingestion:** Parse raw documents into structured format.
- **Stage 2 - Chunking:** Segment into meaningful pieces.
- **Stage 3 - Embedding:** Convert chunks to vector representations.
- **Stage 4 - Indexing:** Store vectors in database for retrieval.
- **Stage 5 - Retrieval:** Find relevant chunks for user queries.
- **Stage 6 - Generation:** LLM creates responses using retrieved context.
- Parsing quality at Stage 1 determines all downstream performance.

YHK

Distracting Documents: The Hidden Danger

- Recent RAG research reveals critical finding:
- **Distracting documents** (incorrectly extracted but seemingly relevant) degrade performance MORE than completely irrelevant documents.
- Poor parsing creates false signal for retrieval systems.
- Vector search matches on corrupted patterns, not true content.
- LLM receives contradictory or incomplete context.
- Result: Hallucinations and incorrect responses.
- **Implication:** Parsing accuracy is more critical than retrieval sophistication.

YHK

Multimodal Integration Challenge

- Documents contain multiple content modalities:
  - Text extraction with coordinates
  - Table structure recognition
  - Image classification and description
  - Mathematical notation preservation
- Each modality requires specialized processing.
- Elements must be linked while preserving distinct properties.
- Challenge: Creating unified representation from diverse inputs.
- Solution: Coordinated pipeline with shared document model.

YHK

Cross-Page References: Document-Level Understanding

- Tables and images frequently span multiple pages.
- Text narratives continue across page breaks.
- Footnotes and citations reference earlier content.
- Page-level processing loses these relationships.
- Requires post-processing with document-level awareness.
- Advanced parsers track element continuity.
- Critical for: legal documents, academic papers, technical reports.

YHK

Performance Case Study: Star Tex

- **Use Case:** Safety Data Sheet processing.
- **Before Advanced Parsing:** 10 minutes per document.
- **After Docling Implementation:** 10 seconds per document.
- **60x speedup** through optimized parsing.
- Enabled real-time document processing pipeline.
- Improved accuracy reduced manual review requirements.
- Demonstrates production-scale impact of parsing quality.

YHK

Resource Comparison: Models

Model	VRAM	Speed	Use Case
Nanonets-OCR	17.7 GB	83 sec/doc	High accuracy
Dolphin	5.8 GB	Fast	Balanced
Docling (heron-101)	17.7 GB	1.26 sec/page	Production
Docling (egret-m)	5.8 GB	0.024 sec/img	Edge devices

- Choose model based on hardware constraints.
- GPU acceleration provides 3-33x speedup.
- Edge devices: use smaller models (egret-m).
- Production systems: balance accuracy and throughput.

YHK

Cloud Solutions: Examples and Pricing

- **Microsoft Azure Document Intelligence:**
  - Layout-aware extraction with semantic chunking.
  - $0.03+ per page, enterprise SLAs.
- **AWS Kendra:**
  - Enterprise search with integrated parsing.
  - Volume-based pricing, automatic scaling.
- **LlamaParse:**
  - 10,000 free credits monthly, diverse format support.
  - Pay-per-page beyond free tier.
- **Cost at Scale:** $30,000+ for 1M pages.

YHK

Open-Source Solutions: Ecosystem

- **Docling (IBM):** Comprehensive modular system, state-of-the-art models.
- **Grobid:** Specialized for academic/scientific documents.
- **Camelot:** Focused on table extraction (99.02% accuracy).
- **Deepdoctection:** Framework for orchestrating detection and OCR.
- **Dolphin (ByteDance):** Open-source VLM, GPU requirements.
- **PyMuPDF:** Lightweight, fast text extraction.
- Each specialized for different document types and use cases.

YHK

Privacy and Compliance Considerations

- **Sensitive Document Types:**
  - Legal contracts and case files
  - Medical records (HIPAA compliance)
  - Financial statements (SOX, GDPR)
  - Classified or proprietary information
- **Cloud Risks:** Data transmission to third-party servers.
- **Open-Source Benefits:**
  - Air-gapped deployment possible
  - No data leaves organization
  - Full audit trail and transparency
- Compliance requirements often mandate local processing.

YHK

Hybrid/Enterprise Solutions

- **Advantages:**
  - Combine flexibility of open-source with professional support.
  - Run locally OR on cloud as needed.
  - Pre-trained models with customization options.
  - Enterprise-grade reliability and SLAs.
- **Disadvantages:**
  - Higher costs than pure open-source.
  - Licensing complexity.
  - May still have vendor constraints.
- Best for: Large enterprises with mixed requirements.

YHK

Docling

What is Docling?

- Simplifies document processing across diverse formats
- Provides advanced PDF understanding capabilities
- Offers seamless integrations with generative AI ecosystem
- Handles complex document structures and layouts
- Enables unified document representation format
- Supports both local and cloud-based processing

YHK

Docling: Development and Community

- Developed by IBM Research Zurich.
- Released under MIT license (fully open-source).
- Hosted within LF AI & Data Foundation.
- Active community development and contributions.
- Regular updates and model improvements.
- Comprehensive documentation and examples.
- Integration with major AI frameworks.
- Production-ready with enterprise backing.

YHK

Key Features

- Multi-format parsing: PDF, DOCX, PPTX, XLSX, HTML, audio files, images
- Advanced PDF understanding: layout, reading order, tables, formulas
- Unified DoclingDocument representation format
- Multiple export options: Markdown, HTML, DocTags, JSON
- Local execution for sensitive data and air-gapped environments
- Plug-and-play integrations: LangChain, LlamaIndex, Crew AI, Haystack
- Extensive OCR support for scanned documents
- Visual Language Models support (SmolDocling)
- Automatic Speech Recognition for audio files
- Simple command-line interface

YHK

Docling Overview

- Developed by IBM Research Zurich, open-source under MIT license.
- Converts documents into AI-ready structured formats.
- Processes 1.26 seconds per page on average.
- Supports 15+ formats: PDF, DOCX, PPTX, HTML, images, etc.
- Fully local execution, ensuring privacy.
- Integrates with LangChain, LlamaIndex, CrewAI, Haystack.
- Includes OCR engines: EasyOCR, Tesseract, RapidOCR.

YHK

Docling Architecture

- Architecture built around modular pipelines.
- Core flow: Document Input → Backend → Pipeline → DoclingDocument → Output.
- Parser backends extract raw text and coordinates.
- Pipelines orchestrate AI models for layout, tables, and enrichment.
- DoclingDocument acts as a unified data model.
- Enables chunking and export into multiple formats.

YHK

Parser Backends

- Handle extraction from different formats (PDF, Word, HTML, Images).
- PDF backend built on qpdf; supports reliable text extraction and rendering.
- Markup-based backends retain semantic structures.
- Image parser uses OCR when needed.
- Backends extract both text coordinates and visual representations.
- Enable consistent data for downstream AI models.

YHK

Processing Pipelines

- Orchestrate sequence of AI models for each document type.
- StandardPdfPipeline applies layout, table, and enrichment models.
- SimplePipeline handles markup formats with minimal processing.
- Users can subclass pipelines for customization.
- Supports addition of custom enrichment models or parameters.

YHK

Layout Analysis Models

- Heron and EGRET family based on RT-DETRv2 Transformer.
- Detect 13 element classes (text, title, table, figure, formula, etc.).
- Trained on DocLayNet (150k+ docs) and proprietary datasets.
- heron-101 achieves highest accuracy (0.78 mAP).
- egret-m offers fastest inference for limited hardware.
- GPU use improves speed up to 33× vs CPU.

YHK

Table Structure Recognition – TableFormer

- Transformer-based model for table structure understanding.
- Trained on PubTabNet, FinTabNet, and TableBank datasets.
- Recognizes cell boundaries, merges, and headers.
- Handles multi-page tables and preserves logical hierarchy.
- Outputs structured, embedding-ready table data.

YHK

Vision-Language Model – Granite-Docling

- 258M parameter Vision-Language Model for end-to-end parsing.
- Uses SigLIP2 visual encoder and Granite 3 language backbone.
- Handles multi-page documents and complex layouts.
- Reduces hallucinations and improves stability.
- Slower than modular pipelines but simpler to use.
- Ideal for full-page document understanding.

YHK

DoclingDocument: Unified Data Model

- Central data representation built on Pydantic models.
- Stores texts, tables, images, metadata, and structure.
- Organizes content into body, furniture, and groups.
- Tracks provenance and spatial info for traceability.
- Enables hierarchy-aware chunking and export.

YHK

DoclingDocument Structure

- Content items: TextItem, TableItem, PictureItem, KeyValueItem.
- Structural organization: Body (main), Furniture (headers/footers), Groups (containers).
- Each item labeled (HEADING, TABLE, PICTURE, etc.).
- Supports nested hierarchy and cross-page references.
- Enables accurate source mapping for retrieval.

YHK

DoclingDocument: Core Attributes Deep Dive

- **Identity:** Unique UID per item, label (semantic type).
- **Spatial:** BoundingBox with coordinates (x0, y0, x1, y1).
- **Hierarchy:** Parent (JSON pointer), children (list of pointers).
- **Provenance:** Source page, coordinates, document origin.
- **Content Layer:** BODY (main) vs FURNITURE (headers/footers).
- All attributes preserved through export for traceability.

YHK

TextItem vs TableItem vs PictureItem

TextItem:

- text: content
- orig: pre-normalized
- formatting
- hyperlink
- level (for headings)
- type (LATEX, etc.)

TableItem:

- data: TableData
- cells: boundaries
- labels: header/body
- bbox: table region
- Preserves structure

PictureItem:

- image_ref
- classification
- captions (list)
- bbox
- Linked context

YHK

Reading Order Algorithm: Four-Stage Process

- **Stage 1 - Recursive Grouping:**
  - Analyze element positions, identify natural groupings.
  - Create hierarchical groups (columns, sections, regions).
- **Stage 2 - Within-Group Ordering:**
  - Sort top-to-bottom (primary), left-to-right (secondary).
  - Apply column-awareness for multi-column layouts.
- **Stage 3 - Visual Cue Integration:**
  - Leverage headers, footers, titles for segmentation.
- **Stage 4 - Cross-Page Assembly:**
  - Connect multi-page tables, figures, references.

YHK

Export Formats: Lossless vs Lossy

- **Lossless Exports:**
  - JSON: Full fidelity (metadata, structure, spatial info, provenance).
  - DocTags: Preserves complex elements (code, formulas, tables).
  - Use for: archival, further processing, validation.
- **Lossy Exports:**
  - Markdown: Clean, LLM-friendly; loses precise bounding boxes.
  - HTML: Web-ready with styling; loses internal structure.
  - Text: Plain text extraction only.
  - Use for: RAG pipelines, human readability, web display.

YHK

Chunking: BaseChunker Abstraction

- Chunker operates directly on DoclingDocument (not post-export text).
- Returns iterator of Chunk objects with text + metadata.
- **Chunk Structure:**
  - text: segment content
  - metadata: page, section, provenance, headers
- Enables structure-aware segmentation.
- Compatible with LangChain, LlamaIndex via wrappers.
- Custom chunkers can extend BaseChunker interface.

YHK

Docling Advantages: Structural Fidelity

- Preserves document hierarchy (sections, subsections).
- Maintains element relationships (captions tied to figures).
- Retains spatial information (coordinates per element).
- Distinguishes content types (tables vs paragraphs).
- **RAG Benefits:**
  - Context maintained through relationships.
  - Intelligent, structure-aware chunking.
  - Precise source references for citations.
  - Appropriate handling of diverse elements.

YHK

Docling Limitations: Accuracy Edge Cases

- **Complex Layouts:** Artistic designs, non-traditional structures.
- **Low-Quality Scans:** OCR degrades with poor quality, handwriting.
- **Specialized Content:** Medical tables, chemical formulas, music notation.
- **Table Extraction:** Deeply nested or unconventional tables.
- **Reading Order:** Unusual layouts with floating elements, watermarks.
- May require custom models or fine-tuning for edge cases.

YHK

Resource Requirements and Optimization

- **GPU Dependency:** High-accuracy models need significant VRAM.
  - heron-101: 17.7 GB VRAM
  - egret-m: 5.8 GB VRAM (faster, lower accuracy)
- **Processing Time:** 1.26 sec/page average (varies by complexity).
- **CPU vs GPU:** 3-33x speedup with GPU acceleration.
- **Model Download:** Multi-GB weights (one-time download).
- **Scalability:** Parallel processing for batch operations.

YHK

Performance Benchmarks: Layout Models

Model	mAP	Speed (GPU)	VRAM
egret-m	0.59	0.024 sec	5.8 GB
egret-l	0.59	0.027 sec	8 GB
heron	0.61	0.030 sec	12 GB
heron-101	0.78	0.028 sec	17.7 GB

- heron-101: Best accuracy (78% on DocLayNet).
- egret-m: Fastest for resource-constrained environments.
- Choose based on accuracy vs speed requirements.

YHK

TableFormer: Training and Capabilities

- **Training Datasets:**
  - PubTabNet: 516k+ heterogeneous tables (PubMed Central).
  - FinTabNet: 112k+ financial tables with precise annotations.
  - TableBank: 417k labeled tables (Word, LaTeX).
- **Capabilities:**
  - Row/column boundary detection.
  - Merged cell recognition.
  - Multi-page table handling.
  - Cell relationship preservation.
- State-of-the-art transformer-based architecture.

YHK

Granite-Docling: Vision-Language Model Alternative

- 258M parameter VLM (ultra-compact).
- Visual encoder: SigLIP2; Language: Granite 3.
- **Advantages:**
  - End-to-end page processing (no pipeline).
  - Improved stability (no token repetition loops).
  - Better complex layout handling.
- **Trade-offs:**
  - Slower than modular pipelines.
  - Higher VRAM requirements.
  - Alternative when simplicity ¿ speed.

YHK

Item Attributes and Provenance

- Each item has unique ID, bounding box, and parent-child hierarchy.
- Tracks origin: page number, coordinates, source document.
- Maintains formatting (fonts, links, levels).
- Tables store structured cell data with boundaries.
- Pictures include image references and captions.

YHK

Content Layers and Provenance

- Two content layers: BODY and FURNITURE.
- BODY = main content; FURNITURE = headers/footers.
- Enables selective processing (ignore headers).
- Provenance enables back-tracing extracted info to page source.
- Supports validation and citation in downstream systems.

YHK

Docling Models and Integration

- Models include layout, table, enrichment, VLM, and OCR components.
- Each model wrapped as a pipeline stage.
- Base interfaces: BaseLayoutModel, BaseTableStructureModel, etc.
- Models can be swapped, fine-tuned, or disabled.
- Supports ensemble or specialized pipelines.

YHK

Pipeline Lifecycle Overview

- Input document → backend selection → pipeline execution.
- Each page processed for layout, tables, and grouping.
- Post-processing merges pages and corrects reading order.
- Final output: DoclingDocument with complete structure.
- Enables accurate downstream export and chunking.

YHK

Pipeline Configuration Options

- Enable or disable OCR, table recognition, formula detection.
- Choose OCR engine: EasyOCR, Tesseract, or RapidOCR.
- Configure table structure mode: ACCURATE or FAST.
- Select layout model type (heron, egret-m, etc.).
- Control picture classification and image scaling.

YHK

Custom Pipeline Development

- Subclass existing pipelines to add custom stages.
- Replace or add models (layout, OCR, enrichment).
- Configure via pipeline options.
- Modify post-processing (caption matching, metadata).
- Extend for domain-specific document formats.

YHK

Reading Order Algorithm

- Reconstructs logical flow from spatial layout predictions.
- Uses recursive grouping and top-bottom, left-right ordering.
- Integrates visual cues (headers, footers, columns).
- Handles multi-page and multi-column documents.
- Ensures semantic coherence for downstream chunking.

YHK

Document Export and Serialization

- Exports DoclingDocument to JSON, Markdown, HTML, or plain text.
- JSON and DocTags retain full fidelity.
- Markdown ideal for LLM-based RAG pipelines.
- Enables web-ready or text-only representations.
- Provides high control over structure and metadata.

YHK

Chunking in Docling

- Operates on DoclingDocument objects directly.
- Supports structural, item-based, size-constrained, and semantic chunking.
- Preserves hierarchy and context boundaries.
- Integrates with LangChain and LlamaIndex.
- Provides metadata-rich chunks for RAG embeddings.

YHK

Docling Advantages

- Preserves structural fidelity and spatial context.
- Fast processing (1.26 sec/page, scalable on GPU).
- Fully local for privacy and compliance.
- Open, extensible, and auditable.
- Deep integration with RAG frameworks.
- Multi-format, multilingual, and multimodal support.

YHK

Docling Limitations and Challenges

- Complex or artistic layouts may confuse layout models.
- OCR struggles with poor scan quality or handwriting.
- GPU-heavy for high-accuracy models (heron-101).
- Customization requires deep architectural knowledge.
- Edge cases in reading order and multilingual support.

YHK

Docling Use Cases and Integrations

- Ideal for RAG applications with structured documents.
- Used in legal, financial, academic, and research contexts.
- Integrates with LangChain and LlamaIndex pipelines.
- Exports to Markdown or JSON for embeddings.
- Flexible APIs, CLI tools, and MCP server integration.

YHK

Basic Usage Pattern

- Configure pipeline options (OCR, tables, images).
- Create a DocumentConverter for PDFs.
- Convert document and export structured outputs.
- Save as JSON or Markdown for further processing.

```python
from docling.document_converter import DocumentConverter
from docling.datamodel.pipeline_options import PdfPipelineOptions

options = PdfPipelineOptions()
options.do_ocr = True
converter = DocumentConverter(
    format_options={InputFormat.PDF: PdfFormatOption(pipeline_options=options)}
)
result = converter.convert_single("document.pdf")
doc = result.document
doc.save_as_markdown("out.md")
```

YHK

Chunking for RAG

- HierarchicalChunker creates semantically coherent segments.
- Each chunk contains text and provenance metadata.
- Enables embeddings and vector storage for retrieval.
- Integrates easily with any RAG backend.

```python
from docling_core.chunking import
    HierarchicalChunker

chunker =
    HierarchicalChunker(max_tokens=1024)
chunks = chunker.chunk(doc)

for chunk in chunks:
    embedding =
     embed_model.embed(chunk.text)
    vector_db.store(chunk.text,
     embedding,
     metadata=chunk.metadata)

```

YHK

Integration with LangChain

- DoclingLoader converts parsed documents to LangChain objects.
- Enables seamless embedding and retrieval integration.
- Works with FAISS, Chroma, or any vector store.
- Ideal for building end-to-end RAG workflows.

```
from langchain.document_loaders import
    DoclingLoader
loader =
    DoclingLoader(file_path="document.pdf")
docs = loader.load()

embeddings = OpenAIEmbeddings()
vector_store =
    FAISS.from_documents(docs,
    embeddings)
retriever = vector_store.as_retriever()

```

YHK

Installation and System Requirements

- **Python Version:**
  - Python 3.10 or higher required
  - Python 3.11 recommended for optimal performance
- **Core Dependencies:**
  - docling-core >= 1.0.0
  - pydantic >= 2.0
  - PyTorch >= 2.0 (for advanced features)
- **Installation Methods:**

```
# Basic installation
pip install docling

# With OCR support (recommended)
pip install docling[ocr]

# Full installation with all features
pip install docling[all]

# From source (for development)
git clone https://github.com/DS4SD/docling.git
cd docling
pip install -e ".[dev]"
```

YHK

System Requirements and Hardware Recommendations

- **Minimum Requirements:**
  - CPU: 2 cores, 2.0 GHz
  - RAM: 4 GB
  - Disk: 2 GB free space
  - OS: Linux, macOS, Windows 10+
- **Recommended for Production:**
  - CPU: 8+ cores, 3.0+ GHz
  - RAM: 16 GB
  - GPU: NVIDIA GPU with 8GB+ VRAM (optional, for acceleration)
  - Disk: 10 GB free space (for models and cache)
- **GPU Acceleration:**
  - CUDA 11.8+ or 12.x
  - cuDNN 8.x
  - 3-5x speedup for layout analysis and OCR
- **Docker Support:**
  - Official Docker images available
  - Includes all dependencies and models

YHK

Version Information and Feature Timeline

- **Current Stable Version: 2.x.x (as of October 2024)**
- **Major Version History:**

Version	Release	Key Features
1.0.0	Q2 2024	Initial release, basic PDF parsing
1.5.0	Q3 2024	OCR support, table extraction
2.0.0	Q4 2024	Unified document model, VLM support
2.1.0	Current	Plugin system, confidence scores

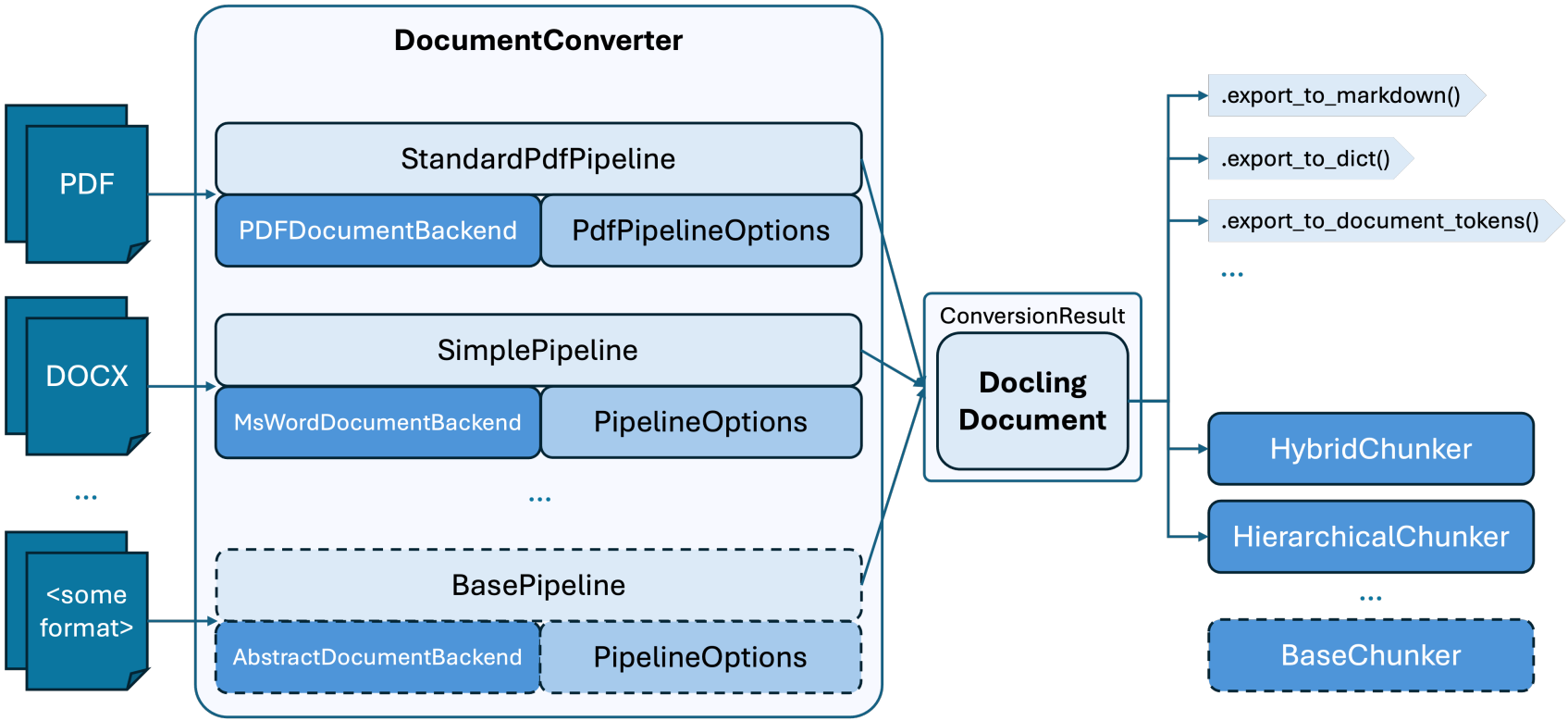
- **Features Covered in This Presentation:**
  - All features from version 2.1.0+
  - DoclingDocument structure (v2.0+)
  - Confidence scoring (v2.1+)
  - Plugin architecture (v2.1+)
- **Check Your Version:**

```
python -c "import docling; print(docling.__version__)"
```

YHK

Architecture Overview

- Document converter selects format-specific backend
- Pipeline orchestrates execution with relevant options
- Conversion result contains DoclingDocument representation
- Export methods available for various output formats
- Serializers handle document transformation
- Chunkers enable document segmentation



DoclingDocument Structure

- Unified document representation using Pydantic datatype
- Expresses text, tables, pictures, and hierarchical sections
- Distinguishes main body from headers/footers (furniture)
- Includes layout information with bounding boxes
- Maintains provenance information for traceability
- Supports disambiguation between content types
- Enables structured document analysis and processing

YHK

Document Content Categories

- **Content Items:**
  - texts: All text representations (paragraphs, headings, equations)
  - tables: Table structures with annotations
  - pictures: Image content with metadata
  - key_value_items: Structured data pairs
- **Content Structure:**
  - body: Main document tree structure
  - furniture: Headers, footers, and non-body content
  - groups: Container items for organizing content
- All items inherit from DocItem type with JSON pointer references

YHK

Document Hierarchy

- Reading order maintained through body tree structure
- Items nested hierarchically under parent elements
- Children ordered sequentially within each tree node
- Page-level organization with title-based grouping
- JSON pointer system for parent-child relationships

```yaml
version: 1.0.0
schema_name: DoclingDocument

body: # The root node of the document content (excluding headers, footers, ...)
  children:
  - $ref: '#/texts/0' # text: Summer activities
  - $ref: '#/texts/1' # title: Swimming in the lake
  label: unspecified
  name: _root_
  self_ref: '#/body'

texts: # The plain text items in this document.
- self_ref: '#/texts/0'
  orig: Summer activities
  text: Summer activities
  label: paragraph # The semantics of a text element are represented by the label
  children: []
  parent:
    $ref: '#/body'
  prov: []
- self_ref: '#/texts/1'
  orig: Swimming in the lake
  text: Swimming in the lake
  label: title
  children: # Any item can have children to reflect section hierachy
  - $ref: '#/texts/2' # text: Duck
  - $ref: '#/texts/3' # text: (empty text)
  - $ref: '#/texts/4' # text: Figure 1: This is a cute duckling
  - $ref: '#/texts/5' # section_header: Let's swim!
  # ...
  parent:
    $ref: '#/body'
  prov: []
# ...
```

Summer activities

Swimming in the lake

Duck

Figure 1: This is a cute duckling

Let's swim!

Content Grouping

- Items grouped under section headings
- Children include both text items and group containers
- List elements organized within group structures
- Group items stored in top-level groups field
- Hierarchical nesting preserves document structure

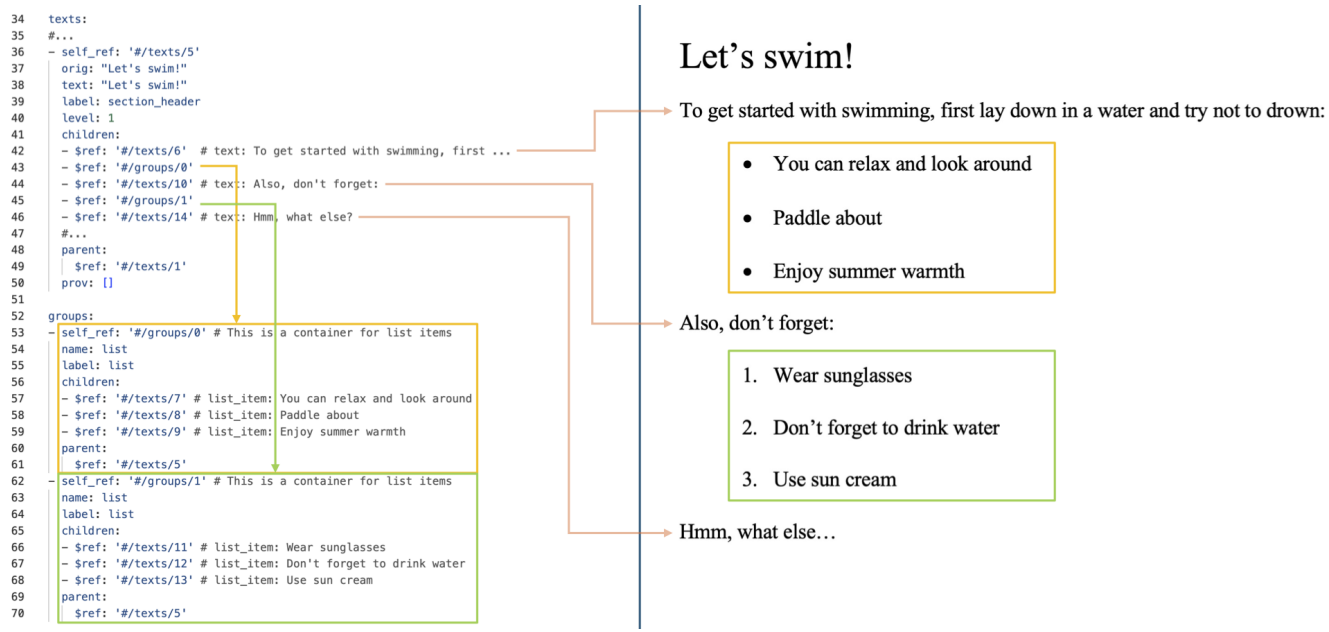


Table Extraction with Docling's Table Transformer

- Docling uses specialized Table Transformer models for accurate table structure recognition
- Detects table boundaries, rows, columns, and cell relationships
- Preserves complex table features: merged cells, nested tables, headers

```python
from docling.document_converter import DocumentConverter, PdfFormatOption
from docling.datamodel.base_models import InputFormat
from docling.datamodel.pipeline_options import PdfPipelineOptions, TableFormerMode

# Configure table extraction
pipeline_options = PdfPipelineOptions()
pipeline_options.do_table_structure = True
pipeline_options.table_structure_options.mode = TableFormerMode.ACCURATE
# Options: FAST, ACCURATE

converter = DocumentConverter(
    format_options={
        InputFormat.PDF: PdfFormatOption(pipeline_options=pipeline_options)
    }
)

result = converter.convert("document.pdf")
```

Table Extraction with Docling's Table Transformer

```

# Access extracted tables
for item, level in result.document.iterate_items():
    if item.label == DocItemLabel.TABLE:
        print(f"Table on page {item.prov[0].page_no}")
        print(f"Rows: {len(item.data)}, Columns: {len(item.data[0])}")
        # Export as markdown, HTML, or JSON
        table_md = item.export_to_markdown()
```

OCR Engine Options and Configuration

- Docling supports multiple OCR engines: EasyOCR, Tesseract, RapidOCR
- Choose based on accuracy needs, speed requirements, and language support
- Tesseract: Fast, good for English; EasyOCR: Better for non-Latin scripts

```python
from docling.datamodel.pipeline_options import PdfPipelineOptions, OcrOptions
from docling.backend.docling_parse_backend import OcrEngine

# Option 1: Tesseract OCR (default, fastest)
pipeline_options = PdfPipelineOptions()
pipeline_options.do_ocr = True
pipeline_options.ocr_options = OcrOptions(
    engine=OcrEngine.TESSERACT,
    lang="eng",  # Language code
    psm=3  # Page segmentation mode
)
```

YHK

OCR Engine Options and Configuration

```
# Option 2: EasyOCR (better accuracy, slower)
pipeline_options.ocr_options = OcrOptions(
    engine=OcrEngine.EASYOCR,
    lang=["en", "ch_sim"],  # Multiple languages
    gpu=True  # Use GPU acceleration
)

# Option 3: RapidOCR (balanced speed/accuracy)
pipeline_options.ocr_options = OcrOptions(
    engine=OcrEngine.RAPIDOCR
)

converter = DocumentConverter(
    format_options={InputFormat.PDF: PdfFormatOption(pipeline_options)}
)
```

YHK

Pipeline Customization: Advanced Configuration

- Customize pipeline for specific document types and use cases
- Enable/disable features based on performance vs accuracy tradeoffs
- Configure models, thresholds, and processing strategies

```python
from docling.datamodel.pipeline_options import (
    PdfPipelineOptions, TableFormerMode, EasyOcrOptions
)

# Custom pipeline for technical documents with tables and formulas
pipeline_options = PdfPipelineOptions()

# Layout and structure
pipeline_options.do_ocr = False  # Digital PDF, no OCR needed
pipeline_options.generate_page_images = False  # Save memory
pipeline_options.generate_picture_images = True  # Extract diagrams

# Table extraction
pipeline_options.do_table_structure = True
pipeline_options.table_structure_options.mode = TableFormerMode.ACCURATE
pipeline_options.table_structure_options.min_confidence = 0.8
```

YHK

Pipeline Customization: Advanced Configuration

```
# Formula extraction
pipeline_options.do_formula = True

# Apply custom pipeline
converter = DocumentConverter(
    format_options={
        InputFormat.PDF: PdfFormatOption(pipeline_options=pipeline_options)
    }
)

result = converter.convert("technical_report.pdf")

# Custom export with specific elements
markdown = result.document.export_to_markdown(
    include_tables=True,
    include_images=True,
    image_placeholder="[Image: {caption}]"
)
```

Image Export and Processing

- Extract embedded images from documents with metadata
- Save images separately or embed as base64
- Access image properties: size, format, bounding box, page location

```python
from pathlib import Path
from docling.datamodel.base_models import DocItemLabel

# Convert document and extract images
result = converter.convert("document.pdf")
doc = result.document

# Create output directory for images
image_dir = Path("extracted_images")
image_dir.mkdir(exist_ok=True)
```

YHK

Image Export and Processing

```python
# Iterate through all picture items
for idx, (item, level) in enumerate(doc.iterate_items()):
    if item.label == DocItemLabel.PICTURE:
        # Access image metadata
        page_no = item.prov[0].page_no
        bbox = item.prov[0].bbox  # Bounding box coordinates

        # Get image data
        if hasattr(item, 'image'):
            # Save image to disk
            image_path = image_dir / f"page_{page_no}_img_{idx}.png"
            item.image.pil_image.save(image_path)

            print(f"Extracted image: {image_path}")
            print(f"  Size: {item.image.size}")
            print(f"  Location: page {page_no}, bbox {bbox}")

            # Get caption if available
            if item.caption:
                print(f"  Caption: {item.caption}")
```

YHK

Metadata Preservation and Provenance

- Docling maintains complete provenance information for all extracted elements
- Track source page, coordinates, hierarchy, and parent relationships
- Essential for citations, source attribution, and document traceability

```python
from docling.document_converter import DocumentConverter

result = converter.convert("research_paper.pdf")
doc = result.document

# Access document-level metadata
print(f"Title: {doc.name}")
print(f"Total Pages: {len(doc.pages)}")
print(f"Confidence Score: {result.confidence.mean_grade}")
```

YHK

Metadata Preservation and Provenance

```python
# Iterate through elements with provenance
for item, level in doc.iterate_items():
    # Get provenance information
    prov = item.prov[0]  # First provenance entry

    print(f"\nElement: {item.label}")
    print(f"  Page: {prov.page_no}")
    print(f"  Bounding Box: {prov.bbox}")  # (x0, y0, x1, y1)
    print(f"  Hierarchy Level: {level}")

    # Parent–child relationships via JSON pointers
    if item.parent:
        print(f"  Parent: {item.parent}")

    # Export with metadata preserved
    item_dict = item.dict()
    print(f"  Full Metadata: {item_dict.keys()}")

# Export with provenance
json_output = doc.export_to_json(indent=2)
# JSON includes all provenance and metadata
```

Serialization Framework

- Document serializer converts DoclingDocument to textual format
- Component serializers: text, table, picture, list, inline
- Serializer provider abstracts serialization strategy
- Base classes enable flexibility and out-of-the-box utility
- Hierarchy includes BaseDocSerializer and specific subclasses
- serialize() method returns text with contribution metadata
- Predefined serializers for Markdown, HTML, DocTags
- Export methods act as user-friendly shortcuts

YHK

Confidence Scores

- Quantitative assessment of document conversion quality
- Numerical scores from 0.0 to 1.0 (higher = better quality)
- Quality grades: POOR, FAIR, GOOD, EXCELLENT
- Helps identify documents requiring manual review
- Enables adjustment of conversion pipelines per document type
- Supports confidence thresholds for batch processing
- Early detection of potential conversion issues
- Available in ConversionResult confidence field

YHK

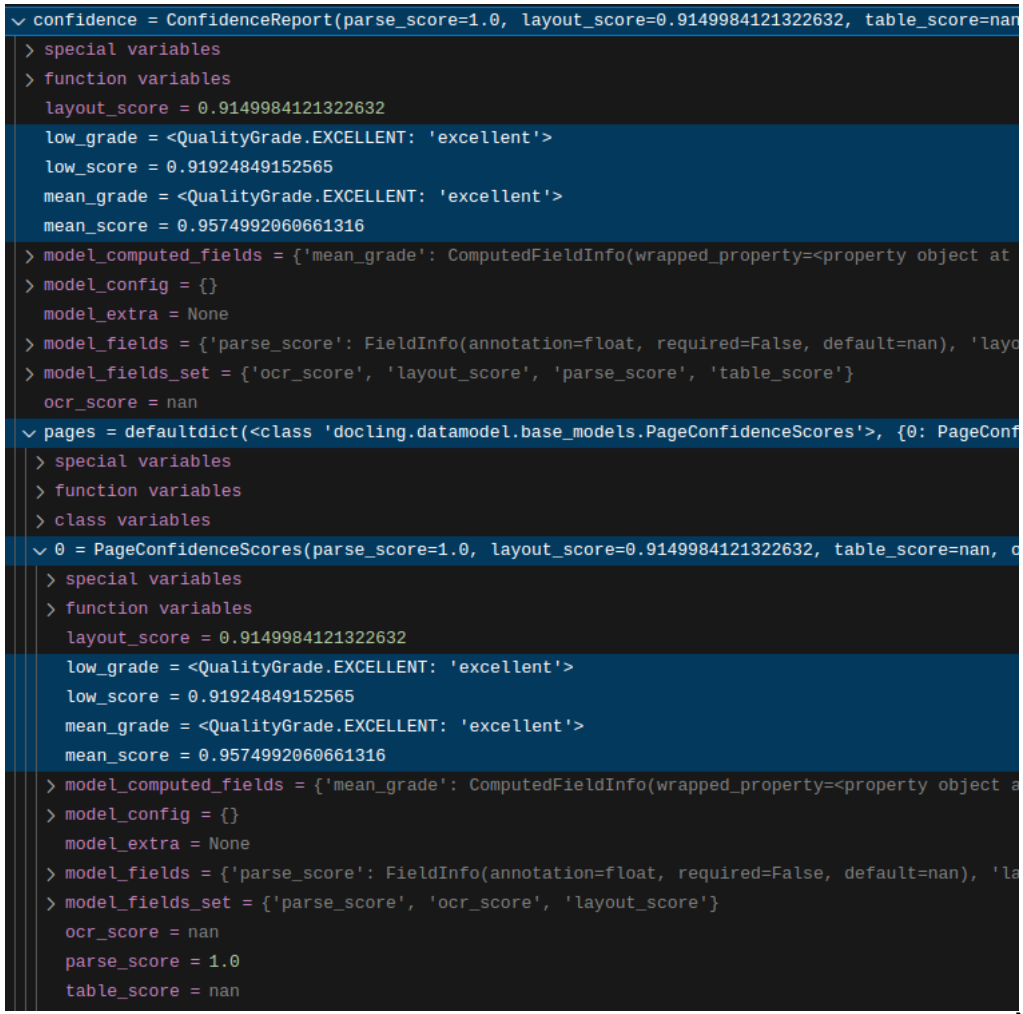
Confidence Score Components

- **Four component scores:**
  - layout_score: Document element recognition quality
  - ocr_score: OCR-extracted content quality
  - parse_score: 10th percentile of digital text cells
  - table_score: Table extraction quality (in development)
- **Summary grades:**
  - mean_grade: Average of four component scores
  - low_grade: 5th percentile score (worst areas)
- Available at both page-level and document-level

YHK

Confidence Example

- Page-level scores stored in pages field
- Document-level scores as averages in root ConfidenceReport
- Numerical values for internal processing
- Categorical grades for user interpretation
- Comprehensive quality assessment framework



YHK

Chunking Framework

- Chunker abstracts document segmentation from DoclingDocument
- Returns stream of chunks with metadata
- Base class hierarchy: BaseChunker and specific subclasses
- Integration with LlamaIndex and other gen AI frameworks
- chunk() method returns iterator of BaseChunk objects
- contextualize() method enriches chunks with metadata
- Enables flexible downstream application integration
- Supports embedding model and generation model feeding

YHK

Chunking Implementations

- **Hybrid Chunker:**
  - Tokenization-aware refinements on hierarchical chunks
  - Splits oversized chunks based on token limits
  - Merges undersized successive chunks with same headings
  - User-configurable merge_peers parameter
- **Hierarchical Chunker:**
  - Uses document structure for element-based chunking
  - Merges list items by default (configurable)
  - Attaches relevant metadata including headers and captions

```python
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()
```

Basic Table Serialization

Inspect the first chunk containing a table — using the default serialization strategy (first example)

```
chunker = HybridChunker(tokenizer=tokenizer)

chunk_iter = chunker.chunk(dl_doc=doc)

chunks = list(chunk_iter)
i, chunk = find_n_th_chunk_with_label(chunks, n=0, label=DocItemLabel.TABLE)
print_chunk(
    chunks=chunks,
    chunk_pos=i,
)
```

Advanced Table Serialization

Specify a different table serializer that serializes tables to Markdown instead of the triplet notation used by default:

```
from docling_core.transforms.chunker.hierarchical_chunker import (
    ChunkingDocSerializer,
    ChunkingSerializerProvider,
)
from docling_core.transforms.serializer.markdown import MarkdownTableSerializer

class MDTableSerializerProvider(ChunkingSerializerProvider):
    def get_serializer(self, doc):
        return ChunkingDocSerializer(
            doc=doc,
            table_serializer=MarkdownTableSerializer())

chunker = HybridChunker(
    tokenizer=tokenizer,
    serializer_provider=MDTableSerializerProvider(),)

chunk_iter = chunker.chunk(dl_doc=doc)

chunks = list(chunk_iter)
i, chunk = find_n_th_chunk_with_label(chunks, n=0, label=DocItemLabel.TABL)
print_chunk(chunks=chunks,chunk_pos=i,)
```

YHK

Plugin System

- Extensible architecture for third-party plugins
- Loaded via pluggy system with setuptools entrypoints
- Unique plugin names required across ecosystem
- Plugin factories for different capabilities
- OCR factory enables additional OCR engines
- Must implement BaseOcrModel with OcrOptions
- External plugins require explicit enablement
- CLI support for plugin management and usage

YHK

Plugin Configuration

- Entry point definition in pyproject.toml:
- Factory registration example:
- Enable external plugins via allow_external_plugins option
- CLI commands for plugin discovery and usage

```
pyproject.toml:
[project.entry-points."docling"]
your_plugin_name = "your_package.module"

--
def ocr_engines():
    return {
        "ocr_engines": [
            YourOcrModel,
        ]
    }
```

External Plugin Usage

- Python API configuration:
- CLI usage with external plugins:

```
pipeline_options = PdfPipelineOptions()
pipeline_options.allow_external_plugins = True
pipeline_options.ocr_options = YourOptions

doc_converter = DocumentConverter(
    format_options={
        InputFormat.PDF: PdfFormatOption(
            pipeline_options=pipeline_options
        )
    }
)

--
docling --show-external-plugins
docling --allow-external-plugins --ocr-engine=NAME
```

Simple Usage Example

- Basic document conversion workflow:
- Supports both local file paths and URLs
- Single converter instance handles multiple formats
- Direct export to various output formats
- Minimal setup required for basic usage
- Automatic format detection and processing

```python
from docling.document_converter import DocumentConverter

source = "https://arxiv.org/pdf/2408.09869"
converter = DocumentConverter()
result = converter.convert(source)  # Returns ConversionResult
doc = result.document
print(doc.export_to_markdown())
```

YHK

Use Cases and Applications

- Document digitization and modernization projects
- Content management and knowledge base creation
- RAG (Retrieval-Augmented Generation) system preparation
- Academic paper processing and analysis
- Business document automation workflows
- Multi-modal AI training data preparation
- Legal document processing and compliance
- Technical documentation conversion and maintenance

YHK

Integration Benefits

- Seamless AI framework integration (LangChain, LlamaIndex)
- Standardized document representation across pipelines
- Consistent quality assessment and monitoring
- Flexible chunking strategies for different use cases
- Extensible plugin architecture for custom requirements
- Local processing for data privacy and security
- Comprehensive format support reduces tool complexity
- Confidence scoring enables quality-based workflows

YHK

Comparative Analysis: Docling vs. Alternatives

- Evaluated alongside Unstructured.io, Marker, PyMuPDF, and LlamaParse.
- Docling provides full structural reconstruction, not just text extraction.
- Marker focuses on OCR-heavy pipelines but lacks metadata fidelity.
- Unstructured.io integrates easily with LangChain but is cloud-biased.
- Docling offers open-source transparency and enterprise-grade accuracy.
- LlamaParse provides cloud performance, but limited customization.

YHK

Performance Comparison

- Docling: 1.26 sec/page average (CPU); 0.04 sec/page (GPU T4).
- Marker: 3.7 sec/page average; slower on complex layouts.
- Unstructured.io: depends on API latency and batch size.
- LlamaParse: GPU-optimized cloud models, but cost per token.
- Docling shows best balance between fidelity and runtime.

YHK

Accuracy Comparison

- Docling (heron-101): 78
- Marker (via PaddleOCR + LayoutLMv3): 65–70
- Unstructured.io: approx. 72
- LlamaParse (VLM): 74
- Docling excels in table detection and reading order.

YHK

Feature Comparison Matrix

- **Docling:** Open, modular, accurate, supports 15+ formats.
- **Marker:** Focused on OCR text extraction.
- **Unstructured.io:** Cloud-first integration tool.
- **LlamaParse:** Fast VLM-based cloud parser.
- **PyMuPDF:** Lightweight library for plain PDF text.
- Docling = best for hybrid local AI and enterprise RAG systems.

YHK

Enterprise Integration Considerations

- Docling supports full offline operation → ideal for secure data.
- Modular APIs enable integration into existing ETL workflows.
- Supports CLI batch processing for large-scale document ingestion.
- JSON and Markdown exports fit RAG pipelines directly.
- Docker images and MCP servers simplify deployment.

YHK

Use Case: RAG System Deployment

- Parse docs → Chunk → Embed → Index → Retrieve → Generate.
- Docling handles the "Parse → Chunk" steps.
- Embeddings and vector storage handled by LangChain/LlamaIndex.
- Downstream model uses parsed context for accurate answers.
- Fully local RAG achievable with no cloud dependence.

```
from docling.document_converter import
    DocumentConverter
from docling_core.chunking import
    HierarchicalChunker
from langchain.vectorstores import
    FAISS

doc =
    DocumentConverter().convert_single("repo
chunks =
    HierarchicalChunker(max_tokens=1024).chu
vector_db = FAISS.from_texts(
    [c.text for c in chunks],
    embedding_function
)
```

YHK

RAG Impact of Parsing Quality

- Accurate structure improves embedding relevance.
- Preserves semantic boundaries $\rightarrow$ better contextual retrieval.
- Reduces LLM hallucinations from fragmented chunks.
- Improves citation and traceability via metadata.
- Parsing is a key determinant of RAG accuracy.

YHK

Docling Strengths Summary

- Structural accuracy and provenance tracking.
- Fully open source and local deployment.
- Integrates natively with AI and RAG frameworks.
- Excellent performance on tables and layouts.
- Modular, customizable, and future-proof.

YHK

Areas for Future Improvement

- Enhance multi-language and RTL support.
- Optimize OCR handling for low-quality scans.
- Simplify custom pipeline configuration.
- Expand VLM coverage for complex documents.
- Build real-time streaming conversion for interactive RAG.

YHK

Recommended Docling Pipeline Setup

- Use `StandardPdfPipeline` for most PDFs.
- Enable OCR only if source is scanned.
- Use `heron-101` for accuracy; `egret-m` for speed.
- Activate table structure mode = `ACCURATE`.
- Export Markdown for embedding and RAG.

```
from
      docling.datamodel.pipeline_options
      import PdfPipelineOptions

options = PdfPipelineOptions()
options.do_ocr = False
options.table_structure_recognition =
      "ACCURATE"
options.do_picture_classification =
      True
converter = DocumentConverter(
    format_options={InputFormat.PDF:
      PdfFormatOption(
         pipeline_options=options)}
)
result =
      converter.convert_single("whitepaper.pdf
```

YHK

Conclusion

- Document parsing is crucial for RAG system quality.
- Docling sets a new benchmark for open-source parsers.
- Combines speed, accuracy, and modularity.
- Ideal for enterprise AI and local privacy contexts.
- Open ecosystem encourages collaboration and innovation.
- Parsing fidelity directly amplifies retrieval and generation accuracy.

YHK

Key Takeaways

- Parsing is the foundation of RAG intelligence.
- Docling achieves best-in-class accuracy with open tools.
- Integration with LangChain simplifies deployment.
- Hybrid pipelines balance speed and semantic depth.
- Future work: domain fine-tuning and adaptive chunking.

YHK

Conclusions

Conclusion: Document Parsing is the Foundation

- **Parsing Quality Determines RAG Success**
  - 20-40 point accuracy difference between parsers
  - Parser upgrades often outperform model upgrades (2x impact)
  - "Garbage in, garbage out" - no amount of prompt engineering can fix bad parsing
- **One Size Does Not Fit All**
  - Document types require different parsing strategies
  - Simple text PDFs $\neq$ Complex layouts $\neq$ Scanned documents $\neq$ Cloud-native formats
  - Evaluate parsers on YOUR data, not benchmark datasets
- **Multi-Modal is the Future**
  - Real-world documents contain text, tables, images, diagrams, code
  - Traditional text-only RAG loses 40-60% of information
  - Modern parsers like Docling preserve all content types
- **Investment in Parsing Pays Off**
  - Reduces downstream costs (fewer errors, less manual review)
  - Enables more accurate and trustworthy RAG systems
  - Foundation for scalable production deployments

YHK

Key Takeaways: Technology and Tools

- **Open Source Solutions are Production-Ready**
  - Docling: State-of-the-art parsing with local execution
  - LlamaIndex: Robust orchestration for RAG pipelines
  - HuggingFace ecosystem: Embeddings and LLMs without vendor lock-in
  - Cost-effective for most use cases (less than 1M docs/year)
- **Docling's Competitive Advantages**
  - Unified document model across all formats
  - Built-in confidence scoring for quality control
  - Advanced layout understanding and reading order
  - Plugin architecture for extensibility
  - 91% accuracy on complex documents (vs 53% for basic parsers)
- **RAG Pipeline Best Practices**
  - Semantic chunking ¿ Fixed-size chunking
  - Preserve document structure and metadata
  - Implement quality filtering based on confidence scores
  - Use modality-specific processing for tables and images

YHK

Implementation Roadmap: From Prototype to Production

Phase 1: Evaluation (Week 1-2)

- Collect representative document samples (50-100)
- Test 2-3 parsers (e.g., PyPDF, Unstructured, Docling)
- Run end-to-end RAG evaluation
- Measure accuracy, speed, cost
- Select parser based on data

Phase 2: Prototype (Week 3-4)

- Implement basic RAG pipeline
- Configure chunking strategy
- Set up vector store (FAISS/Chroma)
- Integrate with LLM
- Build simple UI (Streamlit)

Phase 3: Optimization (Week 5-6)

- Add error handling and retry logic
- Implement batch processing
- Optimize for memory and speed
- Add confidence-based filtering
- Set up monitoring and logging

Phase 4: Production (Week 7-8)

- Implement checkpointing
- Add human review workflow
- Set up CI/CD pipeline
- Performance testing at scale
- Documentation and handoff

YHK

Common Pitfalls and How to Avoid Them

- **Pitfall 1: Ignoring Parser Quality**
  - Problem: "Any parser will do, let's focus on the model"
  - Solution: Evaluate parsers first, before optimizing other components
  - Impact: Can improve accuracy by 20-40 points
- **Pitfall 2: Fixed-Size Chunking**
  - Problem: Breaking tables, code blocks, logical sections
  - Solution: Use semantic/hierarchical chunking (Docling + HybridChunker)
  - Impact: 15-25% improvement in retrieval accuracy
- **Pitfall 3: Losing Metadata**
  - Problem: Can't cite sources or verify answers
  - Solution: Preserve provenance information (page numbers, bounding boxes)
  - Impact: Enables transparency and trust in RAG outputs
- **Pitfall 4: No Quality Monitoring**
  - Problem: Silent failures, degraded accuracy over time
  - Solution: Use confidence scores, log failures, implement human review
  - Impact: Catch issues early, maintain system reliability
- **Pitfall 5: Optimizing Too Early**
  - Problem: Complex optimizations before understanding bottlenecks
  - Solution: Start simple, measure, then optimize based on data
  - Impact: Faster time to production, better ROI

YHK

Future Directions and Emerging Trends

- **Vision-Language Models for Parsing**
  - Models like GPT-4V, Gemini understand document layout visually
  - Can handle complex formats without specialized parsers
  - Trade-off: Higher cost but better accuracy on edge cases
- **Streaming and Real-Time Processing**
  - Parse documents as they're created or edited
  - Incremental updates to vector stores
  - Low-latency requirements for live applications
- **Cross-Document Understanding**
  - Knowledge graphs connecting entities across documents
  - Multi-hop reasoning over document collections
  - Citation networks and reference tracking
- **Domain-Specific Fine-Tuning**
  - Custom parser models for specialized domains (legal, medical, financial)
  - Few-shot learning for new document types
  - Active learning to improve parser accuracy over time
- **Automated Quality Assurance**
  - AI-powered validation of parsing outputs
  - Anomaly detection for corrupted or unusual documents
  - Self-healing pipelines that adapt to failures

YHK

Final Recommendations

For Beginners:

- Start with Docling for best out-of-box experience
- Use LlamaIndex for RAG orchestration
- Follow the evaluation framework:
  1. Visual inspection ("vibe check")
  2. End-to-end RAG testing
  3. Compare on YOUR documents
- Don't over-optimize initially
- Focus on getting working pipeline

For Production Systems:

- Confidence scores for quality control
- Set up monitoring and alerting
- Plan for human-in-the-loop review

Cost Optimization:

- Less than 100K docs: Any
- 100K-1M docs: Docling
- More than 1M docs: Hybrid or custom solution
- Balance quality vs cost per document

Resources to Explore:

- Docling: `https://github.com/DS4SD/docling`
- Parser Benchmarks: `https://github.com/genieincodebottle/parsemypdf`

Remember: Good parsing is the foundation of good RAG. Invest the time to get it right!

YHK

Thanks . . .

- Office Hours: Saturdays, 3 to 5 pm (IST); Free-Open to all; email for appointment to yogeshkulkarni at yahoo dot com
- Call + 9 1 9 8 9 0 2 5 1 4 0 6

(https://www.linkedin.com/in/yogeshkulkarni/)

(https://medium.com/@yogeshharibhaukulkarni)

(https://www.github.com/yogeshhk/)

YHK

Pune AI Community (PAIC)

- Two-way communication:
  - Website puneaicommunity dot org
  - Email puneaicommunity at gmail dot com
  - Call + 9 1 9 8 9 0 2 5 1 4 0 6
  - LinkedIn: https://linkedin.com/company/pune-ai-community
- One-way Announcements:
  - Twitter (X) @puneaicommunity
  - Instagram @puneaicommunity
  - WhatsApp Community: Invitation Link https://chat.whatsapp.com/LluOrhyEzuQLDr25ixZ
  - Luma Event Calendar: puneaicommunity
- Contribution Channels:
  - GitHub: Pune-AI-Community and puneaicommunity
  - Medium: pune-ai-community
  - YouTube: @puneaicommunity

Website

YHK

Pune AI Community (PAIC) QR codes

Website

LinkedIn Page

Luma Events

Medium Blogs

Github Repository

YouTube Videos

Twitter-X

WhatsApp Invite

Instagram

YHK