Introduction to Parsing in Retrieval Augmented Generation (RAG)

Yogesh Haribhau Kulkarni

YHK

Outline

YHK

Parsing is the key

(Ref: Key to RAG Success: Document Parsing Explained - EyeLevel)

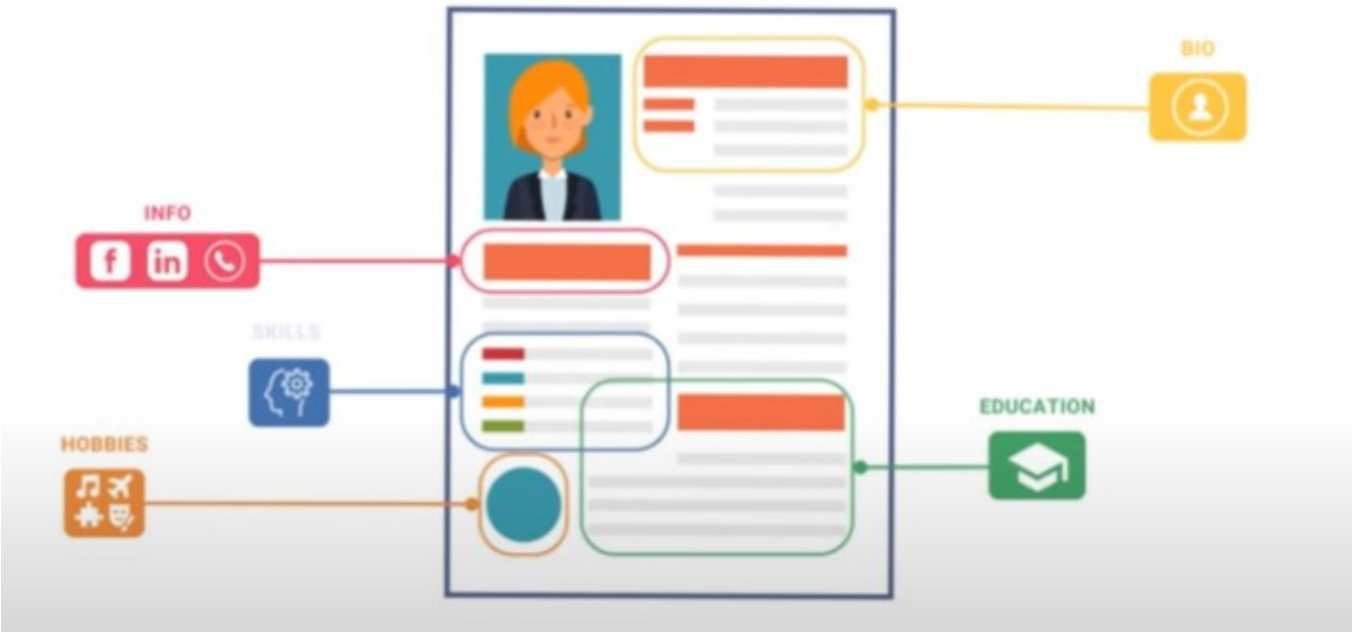
Document Parsing: The Foundation of RAG

- Parsing is the first step in any RAG pipeline.
- Bad parsing undermines even the best RAG strategies.
- Garbage in, garbage out: poor inputs = poor outputs.
- Many overlook parsing in favor of flashy AI tools.
- Without good extraction, nothing else matters.
- Most language models require clean, structured text.
- RAG applications depend on text quality from source docs.
- Advanced RAG still fails without reliable input.
- Models can't fix broken, messy data.
- Real-world systems have failed due to poor parsing.

YHK

What is Document Parsing?

- Converts formats like PDF, DOCX, HTML into usable text.
- Extracts meaningful content for language model input.
- Cleans, structures, and normalizes the data.
- Essential step before chunking or embedding.
- Involves handling many formats and edge cases.



(Ref: Key to RAG Success: Document Parsing Explained - EyeLevel)

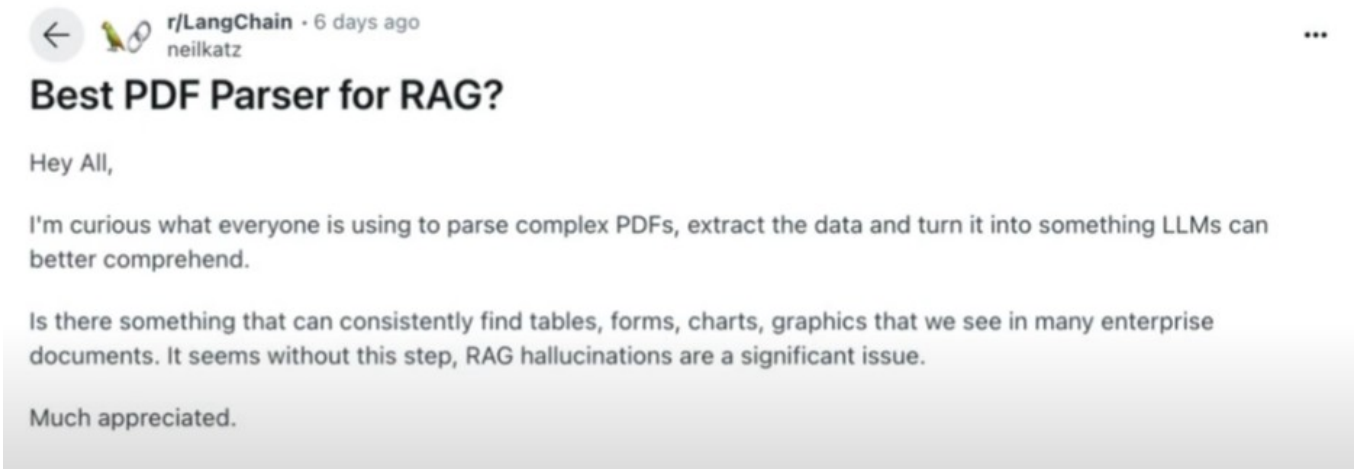
Common Misconceptions

- Engineers often ignore parsing during development.
- Focus tends to be on model tuning or retrieval logic.
- Parsing is wrongly assumed to be solved or trivial.
- Most systems lack formal evaluation of parsers.
- Homemade or ad-hoc solutions dominate practice.

YHK

The Reddit Survey Insight

- Survey on LangChain subreddit revealed no consensus.
- 57 replies yielded 30+ different parsing techniques.
- Most users hacked together informal solutions.
- Few performed proper parser evaluation or comparison.
- Highlights need for standardized testing and benchmarking.



(Ref: Key to RAG Success: Document Parsing Explained - EyeLevel)

YHK

Popular Parsing Tools Compared

- PyPDF – well-known, older, basic PDF extraction.
- Tesseract – OCR-based, handles scanned documents.
- Unstructured – handles messy formats, layout-aware.
- Tools vary widely in output quality and reliability.
- Choice depends on document type and project needs.
- Start by identifying your document types.
- Evaluate parsers with real-world examples.
- Compare outputs side by side.
- Look for structural fidelity, cleanliness, completeness.
- Test rigorously — don't rely on "it seems to work".

YHK

Real-World Example: Medical Bill

- Parsing tested on de-identified medical bill.
- Chosen for layout complexity and format irregularities.
- Shows strengths and weaknesses of each parser.
- Realistic example of what RAG apps encounter.
- Highlights need for resilient parsing strategies.

UNIVERSITY IMAGING CENTER

SUMMARY BILL OF ALL CHARGES

TAX ID: 0893765213
PLEASE REMIT PAYMENT TO:
7835 KESTER AVE SUITE 345
VAN NUYS CA 91405

BILLING STATEMENT

Patient ID	Statement Date
PRE 907354	11/29/2022

Grant, Victoria
5436 S. Trent street
Ontario, CA 91761

ALWAYS REFERENCE PATIENT ID
PHONE : 818 654 3876
DOB : 05/16/1992

TECHNICAL IMAGING PROCEDURES AND BILLINGS

Technical Component,(Imaging) performed by : Dr. Vince

EXAM DATE	PROC. CODE	DESCRIPTION	MOD	B.PART	DX	AMOUNT
7652 Central Street , Montclair CA 91763						
11/17/2022	72141	MRI NECK SPINE W/O DYE	TC	CSPINE	M54.2	$ 1,600.00
11/17/2022	72148	MRI LUMBAR SPINE W/O DYE	TC	LSPINE	M54.5	$ 1,600.00

PROFESSIONAL PROCEDURES AND BILLINGS

EXAM DATE	PROC. CODE	DESCRIPTION	MOD	B.PART	DX	AMOUNT
Professional Component (Radiology Report) Interpreted by: Dr. Vince						
7652 Central Street , Montclair CA 91763						
11/17/2022	72141	MRI NECK SPINE W/O DYE	26	CSPINE	M54.2	$ 600.00
11/17/2022	72148	MRI LUMBAR SPINE W/O DYE	26	LSPINE	M54.5	$ 600.00

Total Charges	Total Payments	Total Adjustments	Balance Due
$4,400.00	$0.00	$0.00	$4,400.00

Internal use Only: 9/27/2022

ADDITIONAL OR REVISED BILLING MAY OCCUR
PLEASE EMAIL BILLING@UIC.COM TO CONFIRM FINAL BILLING AMOUNT

YHK

Limitations of PyPDF

- PyPDF ok for texts but struggles with complex formats like tables.
- It failed to extract key data from real-world docs.
- Parsing tables often results in empty or broken content.
- Not due to PyPDF's fault—PDFs are inherently hard to parse.
- Many PDFs use inconsistent encoding and layouts.

TAX ID: 0893765213
PLEASE REMIT PAYMENT TO:
7835 KESTER AVE SUITE 345
VAN NUYS CA 91405

Grant, Victoria
5436 S. Trent street
Ontario, CA 91761

Dr. Vince

7652 Central Street , Montclair CA 91763

7652 Central Street , Montclair CA 91763

UNIVERSITY IMAGING CENTER

SUMMARY BILL OF ALL CHARG

TAX ID: 0893765213
PLEASE REMIT PAYMENT TO:
7835 KESTER AVE SUITE 345
VAN NUYS CA 91405

BILLING STATEM

Patient ID | Statem

PRE 967304 | 11/2

Grant, Victoria
5436 S. Trent street
Ontario, CA 91761

ALWAYS REFERENCE PA

PHONE : 818 654 3876
DOB : 05/16/1992

TECHNICAL IMAGING PROCEDURES AND BILLINGS

EXAM DATE	PROC. CODE	DESCRIPTION	MOD	B.PART	DX
7652 Central Street , Montclair CA 91763					
11/17/2022	72141	MRI NECK SPINE W/O DYE	TC	CSPINE	M54.2
11/17/2022	72148	MRI LUMBAR SPINE W/O DYE	TC	LSPINE	M54.5

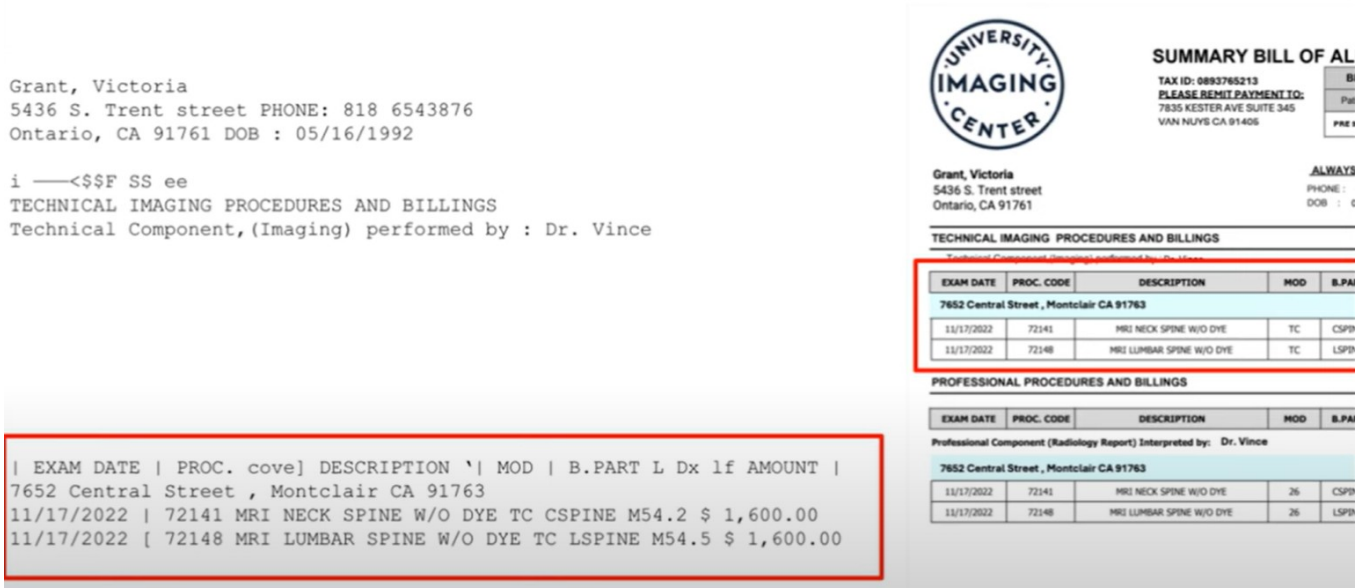
PROFESSIONAL PROCEDURES AND BILLINGS

EXAM DATE	PROC. CODE	DESCRIPTION	MOD	B.PART	DX
Professional Component (Radiology Report) Interpreted by: Dr. Vince					
7652 Central Street , Montclair CA 91763					
11/17/2022	72141	MRI NECK SPINE W/O DYE	26	CSPINE	M54.2
11/17/2022	72148	MRI LUMBAR SPINE W/O DYE	26	LSPINE	M54.5

YHK

Tesseract OCR: Strengths and Weaknesses

- Tesseract uses image-based OCR to extract text.
- Better at recognizing tables than PyPDF.
- Still introduces errors in column alignment.
- Column headers often get merged or misread.
- OCR also introduces spelling mistakes (e.g. "Cove" vs. "Code").



```
Grant, Victoria
5436 S. Trent street PHONE: 818 6543876
Ontario, CA 91761 DOB : 05/16/1992

i ——<$$F SS ee
TECHNICAL IMAGING PROCEDURES AND BILLINGS
Technical Component,(Imaging) performed by : Dr. Vince

| EXAM DATE | PROC. cove] DESCRIPTION '| MOD | B.PART L Dx lf AMOUNT |
7652 Central Street , Montclair CA 91763
11/17/2022 | 72141 MRI NECK SPINE W/O DYE TC CSPINE M54.2 $ 1,600.00
11/17/2022 [ 72148 MRI LUMBAR SPINE W/O DYE TC LSPINE M54.5 $ 1,600.00
```


YHK

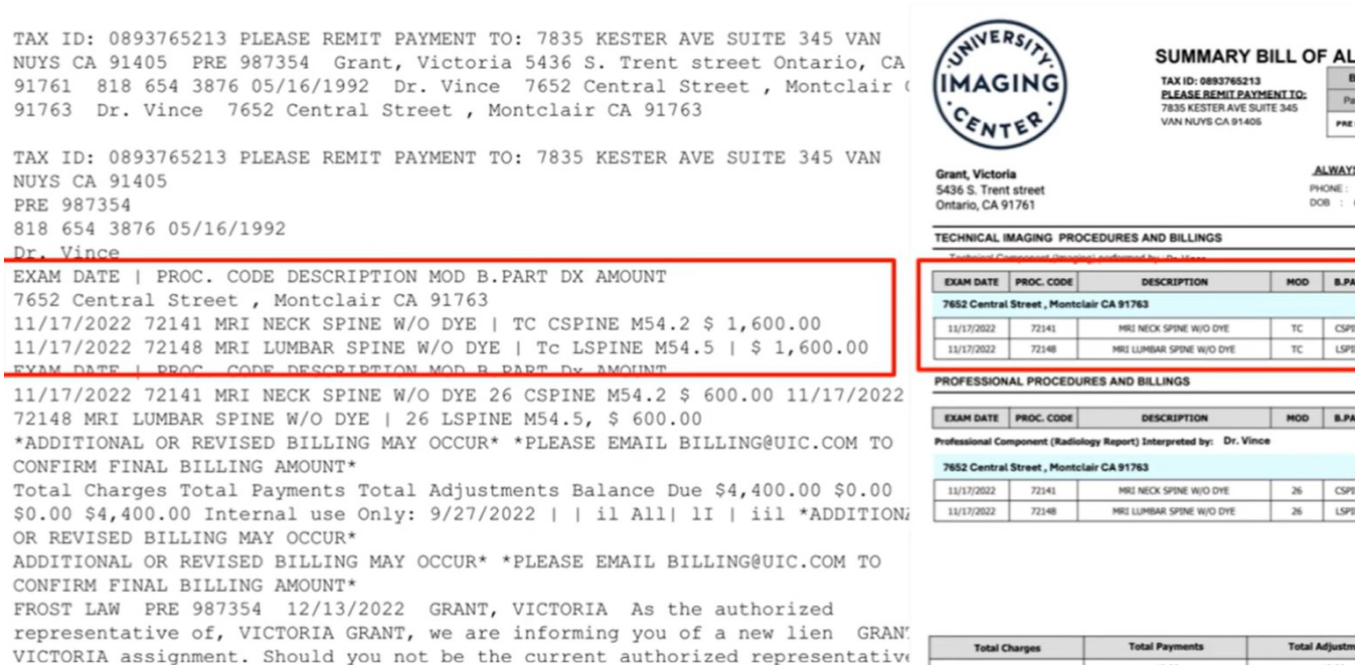
Challenges with OCR Outputs

- Language models must infer structure from broken text.
- Humans can "guess" meaning—models may not.
- Noisy extractions increase risk of incorrect answers.
- Inconsistent column separation confuses models.
- Clean layout is crucial for reliable RAG responses.

YHK

Unstructured: A Common Default

- Popular choice 'Unstructured' company; default in LangChain integrations.
- Handles layout better than OCR in some cases.
- Still suffers from column misalignments.
- Model must rely on context instead of structure.
- Reasonable quality, but far from perfect.



```
TAX ID: 0893765213 PLEASE REMIT PAYMENT TO: 7835 KESTER AVE SUITE 345 VAN
NUYS CA 91405  PRE 987354  Grant, Victoria 5436 S. Trent street Ontario, CA
91761  818 654 3876 05/16/1992  Dr. Vince  7652 Central Street , Montclair (
91763  Dr. Vince  7652 Central Street , Montclair CA 91763

TAX ID: 0893765213 PLEASE REMIT PAYMENT TO: 7835 KESTER AVE SUITE 345 VAN
NUYS CA 91405
PRE 987354
818 654 3876 05/16/1992
Dr. Vince
EXAM DATE | PROC. CODE DESCRIPTION MOD B.PART DX AMOUNT
7652 Central Street , Montclair CA 91763
11/17/2022 72141 MRI NECK SPINE W/O DYE | TC CSPINE M54.2 $ 1,600.00
11/17/2022 72148 MRI LUMBAR SPINE W/O DYE | Tc LSPINE M54.5 | $ 1,600.00
EXAM DATE | PROC. CODE DESCRIPTION MOD B.PART Dx AMOUNT
11/17/2022 72141 MRI NECK SPINE W/O DYE 26 CSPINE M54.2 $ 600.00 11/17/2022
72148 MRI LUMBAR SPINE W/O DYE | 26 LSPINE M54.5, $ 600.00
*ADDITIONAL OR REVISED BILLING MAY OCCUR* *PLEASE EMAIL BILLING@UIC.COM TO
CONFIRM FINAL BILLING AMOUNT*
Total Charges Total Payments Total Adjustments Balance Due $4,400.00 $0.00
$0.00 $4,400.00 Internal use Only: 9/27/2022 | | il All| lI | iil *ADDITIONA
OR REVISED BILLING MAY OCCUR*
ADDITIONAL OR REVISED BILLING MAY OCCUR* *PLEASE EMAIL BILLING@UIC.COM TO
CONFIRM FINAL BILLING AMOUNT*
FROST LAW  PRE 987354  12/13/2022  GRANT, VICTORIA  As the authorized
representative of, VICTORIA GRANT, we are informing you of a new lien  GRANT
VICTORIA assignment. Should you not be the current authorized representative
```


YHK

Parsing Tradeoffs: Model vs. Parser

- Many teams focus on improving the model first.
- Upgrading the parser might yield better gains.
- Better input can reduce model burden.
- Smaller or older models benefit more from clean text.
- Strong parsing reduces reliance on inference tricks.

YHK

LlamaParse: Cleaner Table Extraction

- Developed by LlamaIndex, supports markdown output.
- Clearly separates rows and columns with pipes.
- Markdown format improves model interpretability.
- Some formatting quirks but largely usable.
- Outperforms other parsers in structural clarity.

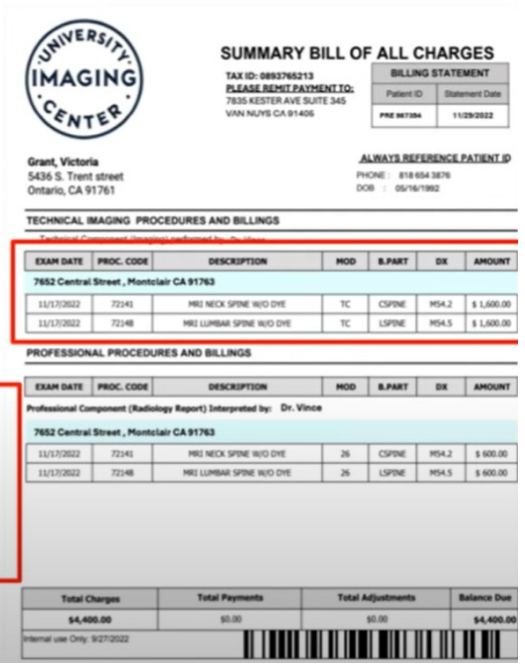
```
|BILLING@PRECISEMRICOM3| | | |
| |6710 KESTER AVE SUITE 126| | |
| |7835 KESTER AVE SUITE 345| | |
| |VAN NUYS CA 91405 VAN NUYS, CA 91405| | |
| | | |PRE 987354PRE79617211/29/2022|
|DARBYSHIRE, JUSTIN JOHN| | | |
|2848 E. BERRYLOOP PRIVADO 54| | | |
|Grant, Victoria| | | |
|ONTARIO,CA 91761| | | |
|5436 S. Trent street| | | |
|Ontario, CA 91761| | | |
| | |PHONE|909-609-6087|
| | |DOB|1/21/1995818 654 3876|
| | | |05/16/1992|

TECHNICAL IMAGING PROCEDURES AND BILLINGS

|Technical Component,(Imaging) performed by Dr. VincePrecise
Imaging| | | |
|---|---|---|---|
|EXAM DATE|PROC. CODE|DESCRIPTION|MOD|B.PART|DX|AMOUNT|
|11/17/2022|72141|MRI NECK SPINE W/O DYE|TC|CSPINE|M54.2|$ 1,600.00|
|11/17/2022|72148|MRI LUMBAR SPINE W/O DYE|TC|LSPINE|M54.5|$
1,600.00|

PROFESSIONAL PROCEDURES AND BILLINGS

|EXAM DATE|PROC. CODE|DESCRIPTION|MOD|B.PART|DX|AMOUNT|
```



X-ray Parser: Multimodal Approach

- Combines vision models with grounding strategies.
- Detects tables and layout visually before parsing.
- Converts visual structure into usable text format.
- Produces reliable and model-friendly outputs.
- Especially effective on visually complex documents.


```
[
  {
    "summary": "The following table contains details of technical
imaging procedures and billings performed by Dr. Vince at the
University Imaging Center. It includes exam date, procedure code,
description, modifier, body part, diagnosis, and amount."
  },
  {
    "AMOUNT": "$1,600.00",
    "B.PART": "CSPINE",
    "DESCRIPTION": "MRI NECK SPINE W/O DYE",
    "DX": "M54.2",
    "EXAM DATE": "11/17/2022",
    "MOD": "TC",
    "PROC. CODE": "72141"
  },
  {
    "AMOUNT": "$1,600.00",
    "B.PART": "LSPINE",
    "DESCRIPTION": "MRI LUMBAR SPINE W/O DYE",
    "DX": "M54.5",
    "EXAM DATE": "11/17/2022",
    "MOD": "TC",
    "PROC. CODE": "72148"
  }
]
```

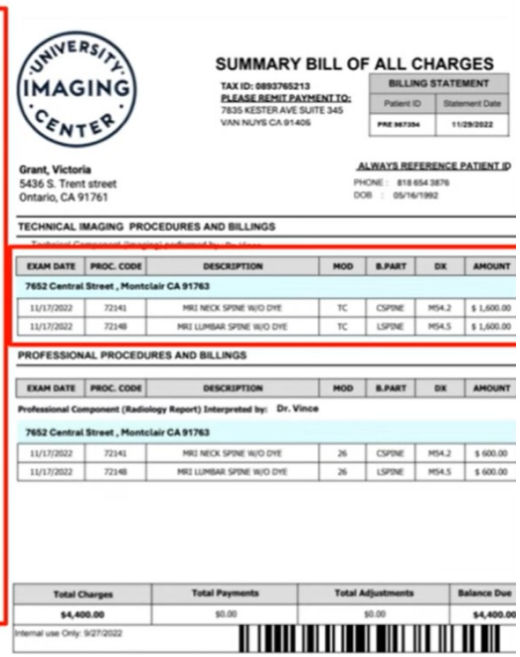
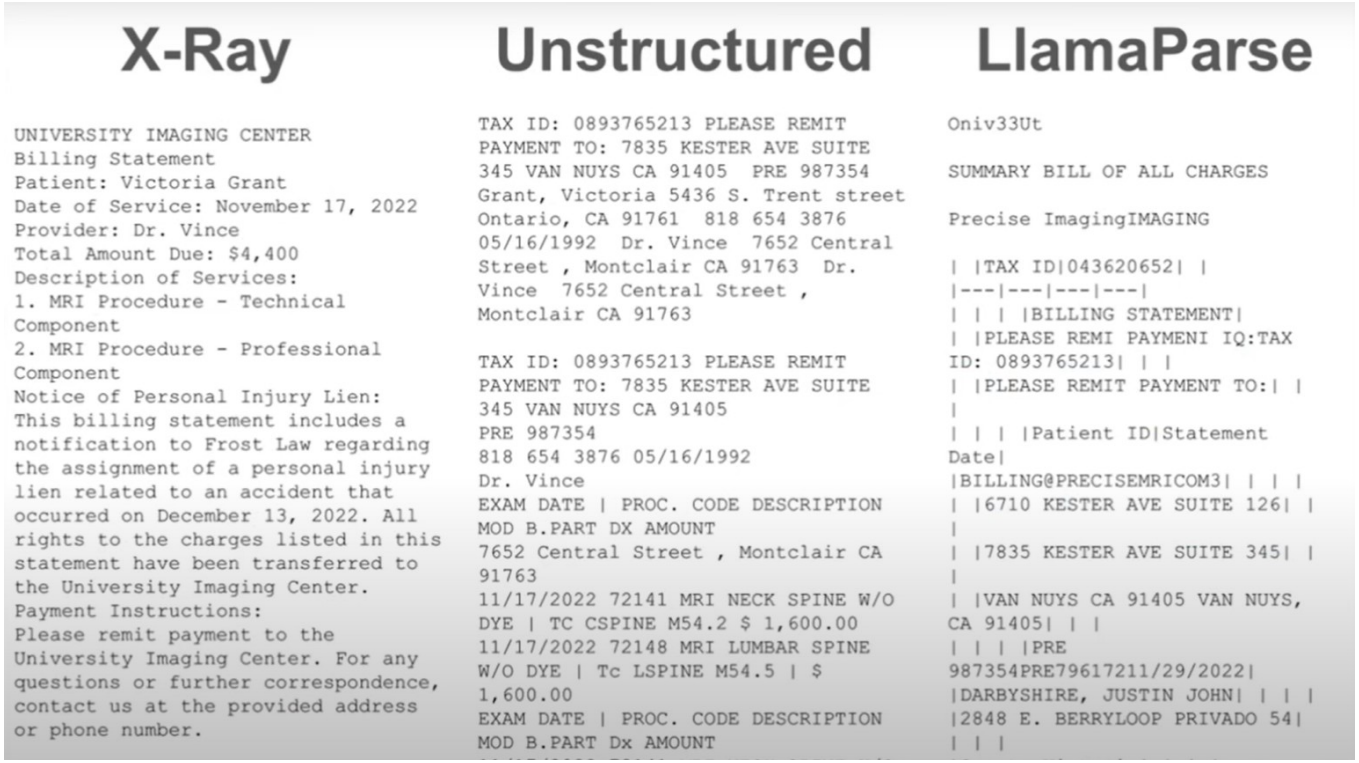



Table Extraction Comparison

- PyPDF fails with complex tables.
- Tesseract detects tables but mangles headers.
- Unstructured does OK, but not perfectly.
- LlamaParse gives clean markdown tables.
- X-ray produces structured, grounded output.



Narrative + JSON: A Guided Format

- Output begins with a narrative summary for context.
- Clearly explains the purpose and structure of the table.
- Follows with a clean JSON representation of table data.
- Format: cell-by-cell structured, easy to interpret.
- "Tell-then-show" approach improves model comprehension.

YHK

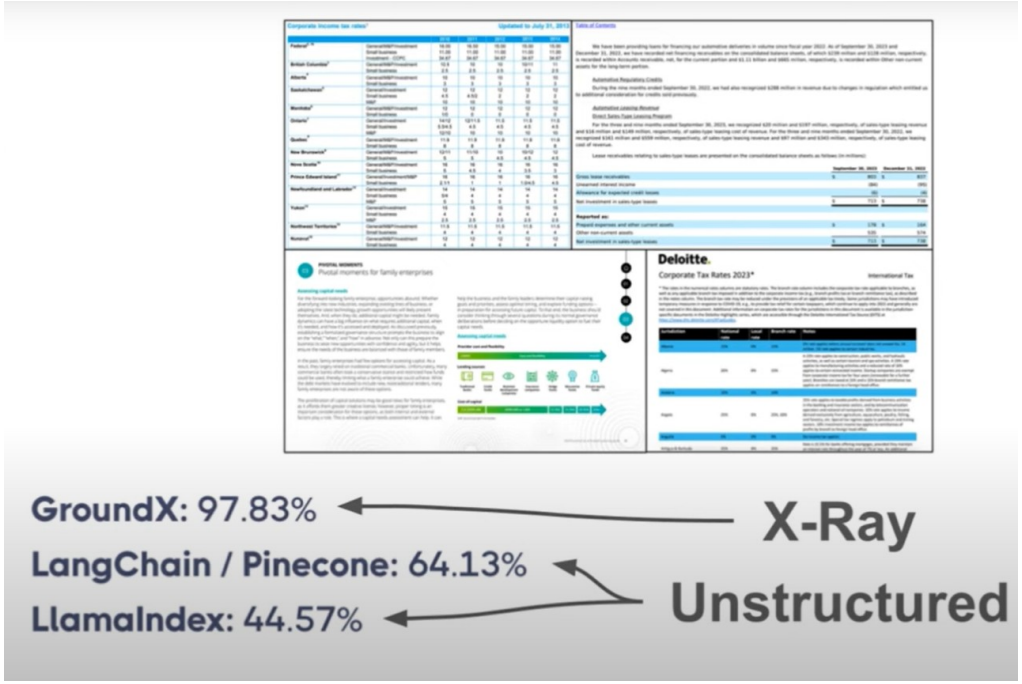
Why Output Format Matters

- Parsers may extract the same data, but format it differently.
- Output style can strongly affect model performance.
- JSON and markdown help structure information clearly.
- Human-readable structure supports better inference.
- Cleaner format = better grounding for language models.

YHK

Impact of Parsing Quality

- We ran the same RAG pipeline over identical documents.
- Different parsers resulted in drastically different performance.
- Main variable: parsing quality—not model or retriever.
- Shows how foundational parsing is to good RAG results.
- A poor parser can undermine even advanced models.



YHK

Benchmark Setup (Clarified)

- **LlamaIndex**: Used PyPDF (not LlamaParse).
- **LangChain + Pinecone**: Used Unstructured.
- **GroundX**: Used X-ray with vision + grounding.
- All pipelines ran the same questions on same documents.
- Parser choice significantly influenced accuracy.

DUE: End-to-End Document Understanding Benchmark

Łukasz Borchmann*[♡†] Michał Pietruszka*[♡‡] Tomasz Stanisławek*[♡‡]
Dawid Jurkiewicz[♡¶] Michał Turski[♡¶] Karolina Szyndler[♡] Filip Graliński[♡¶]

[♡]Applica.ai
firstname.lastname@a

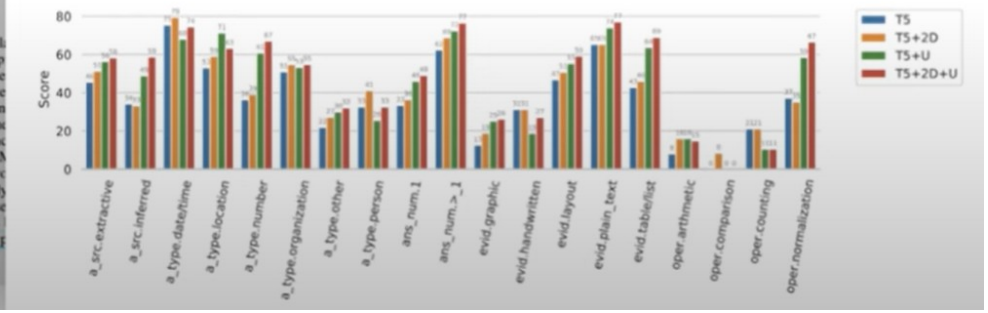
[†]Poznan University of Technology

[‡]Warsaw University of Technology [¶]Ad

Abstract

Understanding documents with rich layouts pl
hyper-automation but remains a challenging top
Additionally, the lack of a commonly accepte
quantify progress in the domain. To empowe
duce the Document Understanding Evaluation
both available and reformulated datasets to m
of systems in real-world scenarios. The benc
Answering, Key Information Extraction, and
tasks over various document domains and layo
and infographics. In addition, the current study
analyzes challenges in currently available datase
aware language modeling. We open both the
mentations and make them available at http
https://github.com/due-benchmark.

OCR	DocVQA	IVQA	Charity	DeepForm	Average	Average scores for different diagnostic categories				
						Extractive	Inferred	Handwritten	Table/list	Plain text
Azure CV (v3.2)	71.8	40.0	57.7	74.8	61.1	51.3	33.0	31.3	46.0	65.3
Tesseract (v4.0)	55.7	28.3	55.7	66.8	51.6	43.1	29.5	12.5	27.2	61.1



Future Test Considerations

- Results likely to improve if LlamaParse replaces PyPDF.
- Parsing upgrades often outperform model upgrades.
- Models can only reason with what they're given.
- Better structured data = better answers, less guessing.
- Parsing is the cheapest way to level up your RAG stack.

Parsing Alone Can Move the Needle

- Academia confirms: changing only the parser impacts performance.
- Benchmark: same RAG system, different parsers $\rightarrow$ up to 20-point difference.
- Parser quality matters more than fancy downstream techniques.
- Quick wins: swap out low-quality parsers before tweaking your RAG logic.

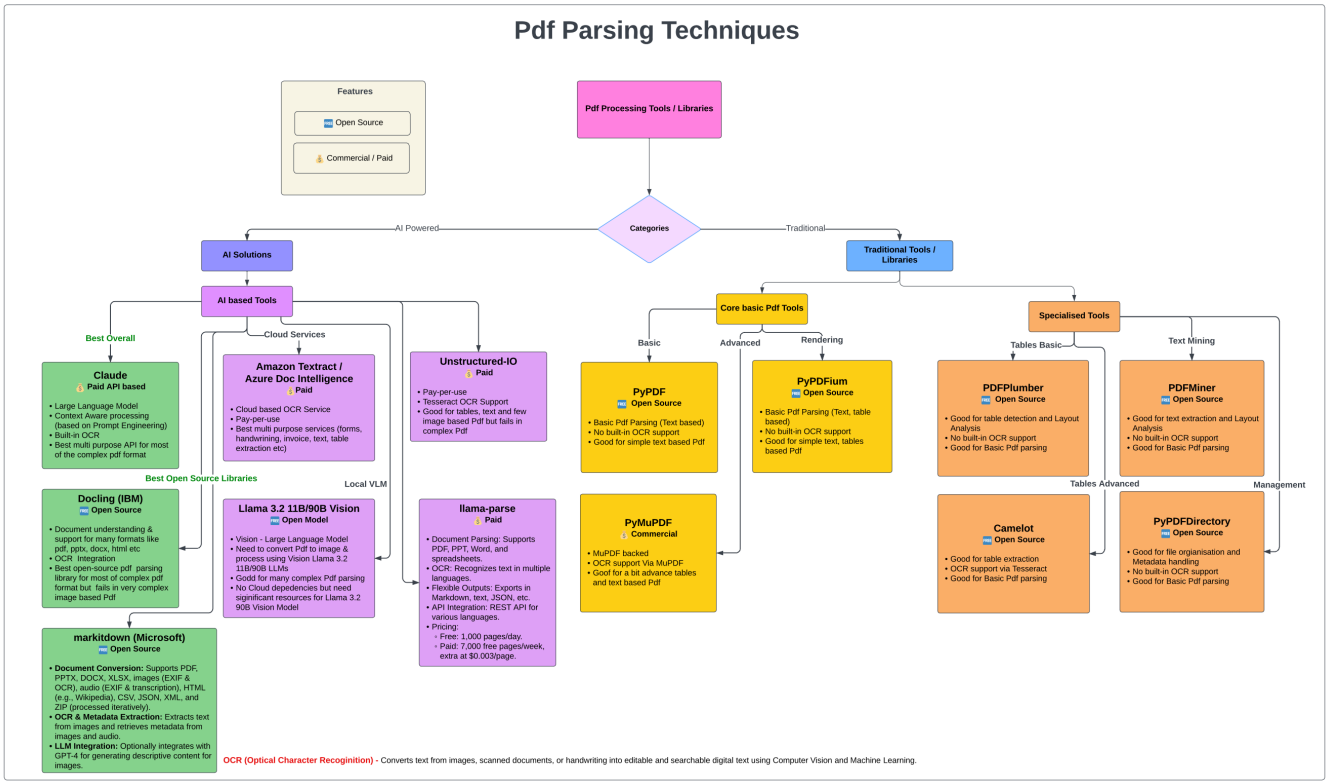
YHK

Document Context is Crucial

- Not all documents are created equal.
- Scientific papers, 10-Ks, clinical notes – all behave differently.
- Choose a parser suited to your specific domain.
- No single parser wins for all use cases.

YHK

PDF Parsing Guide

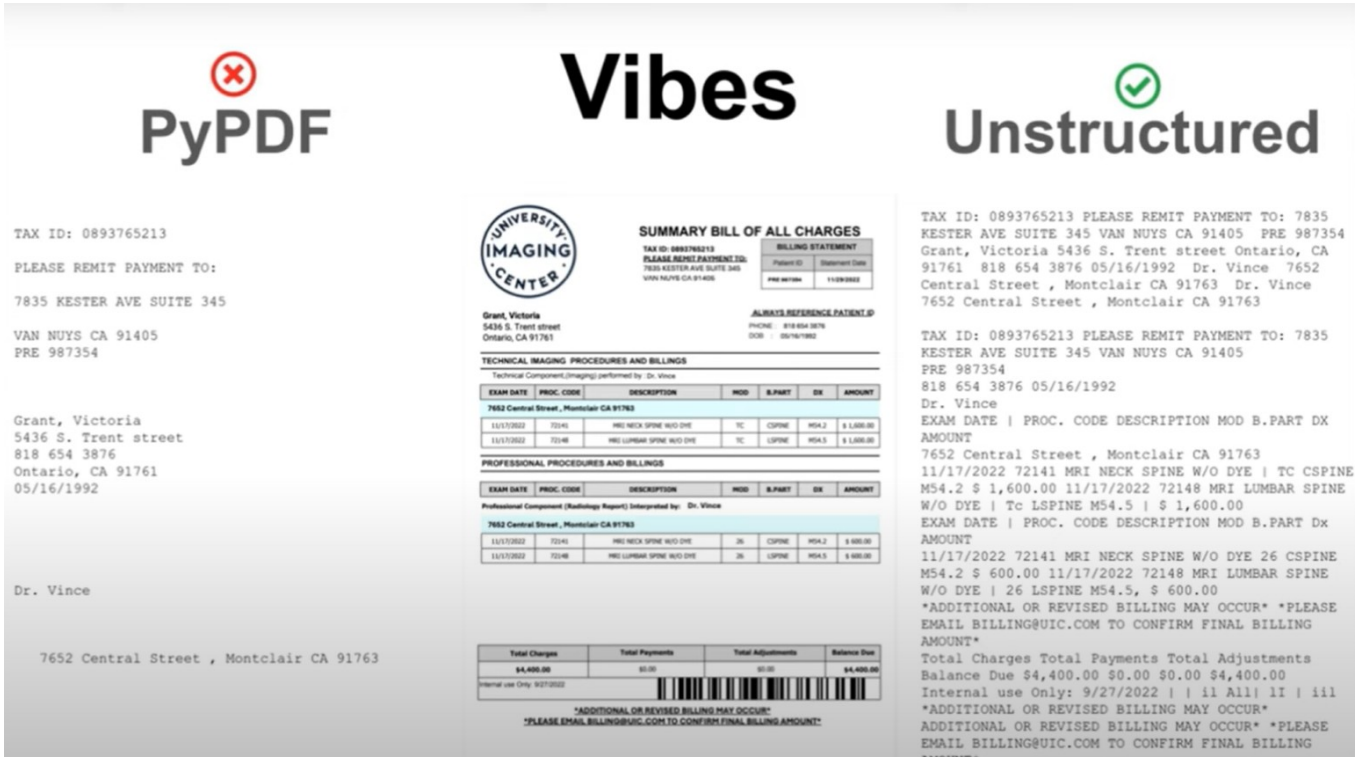


(Ref: https://github.com/genieincodebottle/parsemypdf/blob/main/pdf-parsing-guide.pdf)

Picking a Parser: A Two-Pronged Approach

1. **Vibe check**: Run your data through multiple parsers. Look at the outputs.
2. **End-to-end eval**: Keep the RAG system constant, vary only the parser, then compare results.

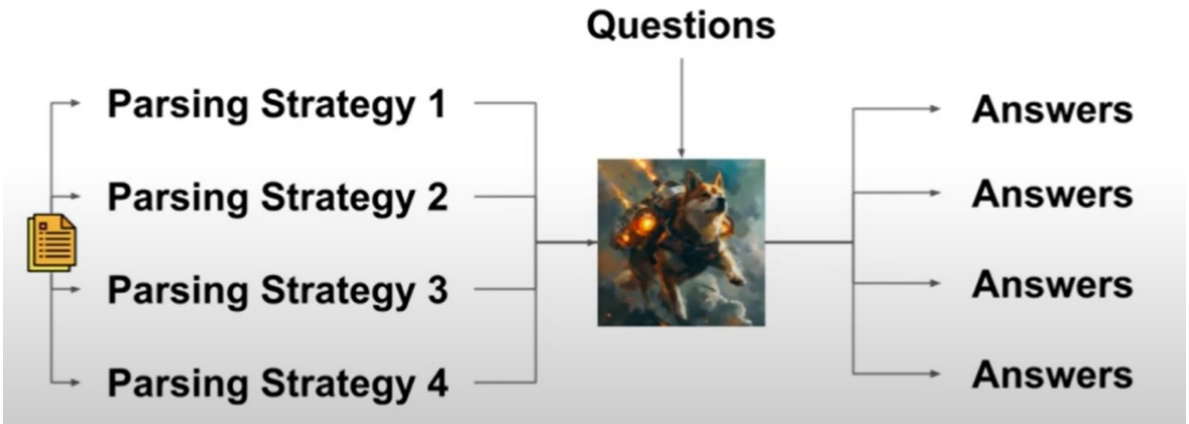
"Your brain is the best model—start with your eyes."

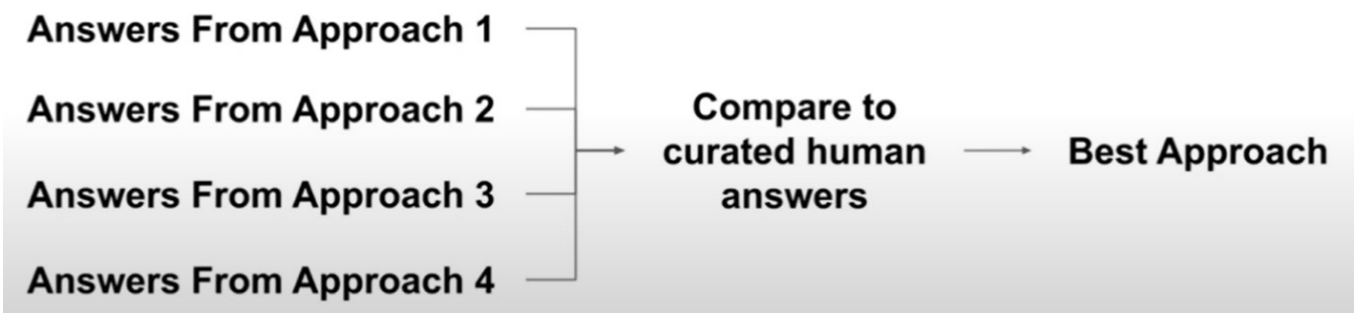


YHK

How to Run an End-to-End Evaluation

- Change one component: the parser.
- Keep the rest of the RAG pipeline fixed.
- Feed questions through each parser's output.
- Compare generated answers to ground truth.
- Labor-intensive, but the most reliable evaluation method.





Evaluation

Auto-Eval: A Helping Hand

- Start with human-generated QA pairs (ground truth).
- Use LLMs to compare parser outputs to ground truth answers.
- Helps scale eval, but still requires initial human input.
- Avoids the trap of "models grading their own homework."

Alternative Eval: ELO Ranking

- Useful when answers are subjective or non-falsifiable.
- Compare outputs pairwise: "Which one is better?"
- Rank parsers using ELO-style systems (used in chess).
- Great for stylistic or qualitative tasks.

YHK

Final Takeaways

- **Parsing is foundational.** Bad parsing = bad RAG, no matter the model.
- **There is no one-size-fits-all parser.**
- **Evaluate in context.** Use real documents and real questions.
- **Combine human intuition with structured evals.**
- **Opportunities exist.** Big gap in parser testing and tooling.
- Parsing is hard, but absolutely critical.
- Tools like LlamaParse, Unstructured, and X-ray are changing the game.
- Try multiple parsers and test thoroughly on your data.
- Don't trust models to validate their own output.
- Huge room for innovation in parser evaluation and automation.

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

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

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

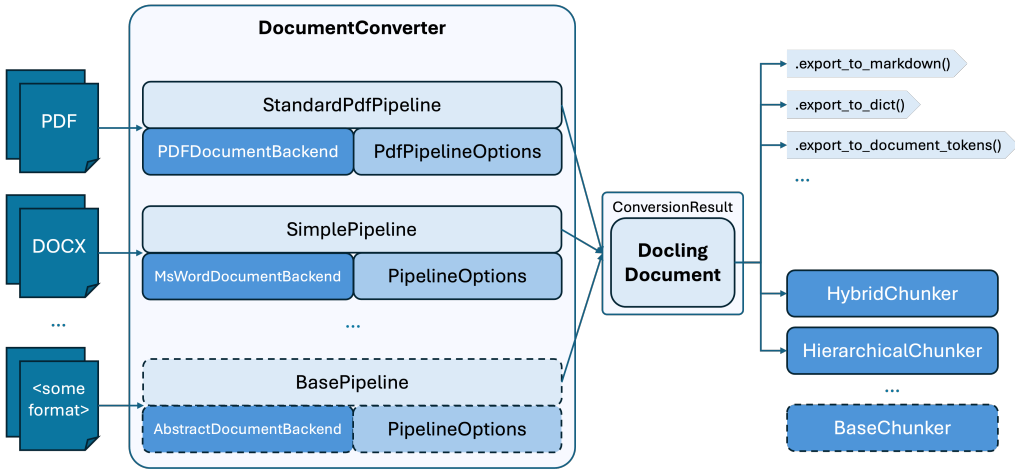
- **Features Covered in This Presentation:**
  - All features from version 2.1.0+
  - DoclingDocument structure (v2.0+)
  - Confidence scoring (v2.1+)
  - Plugin architecture (v2.1+)
- **Check Your Version:**

```
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

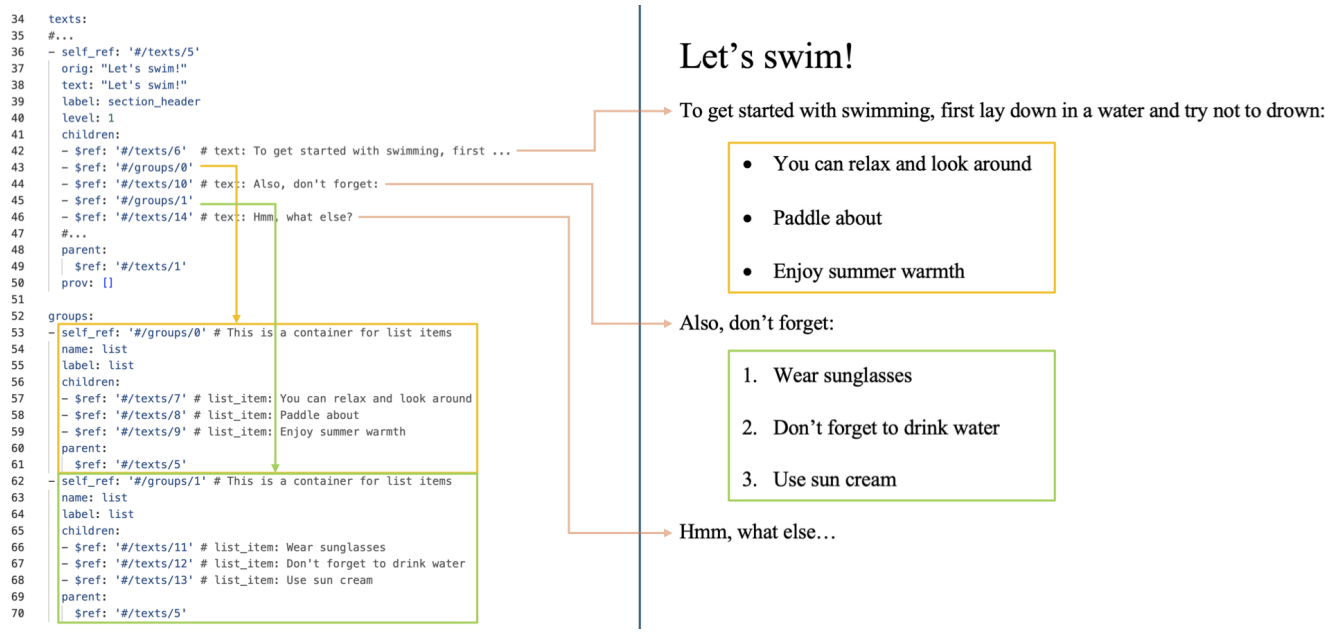


Table Extraction with Docling's Table Transformer

- 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")
```

YHK

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

```
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)}
)
```

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)
```

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}")
```

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

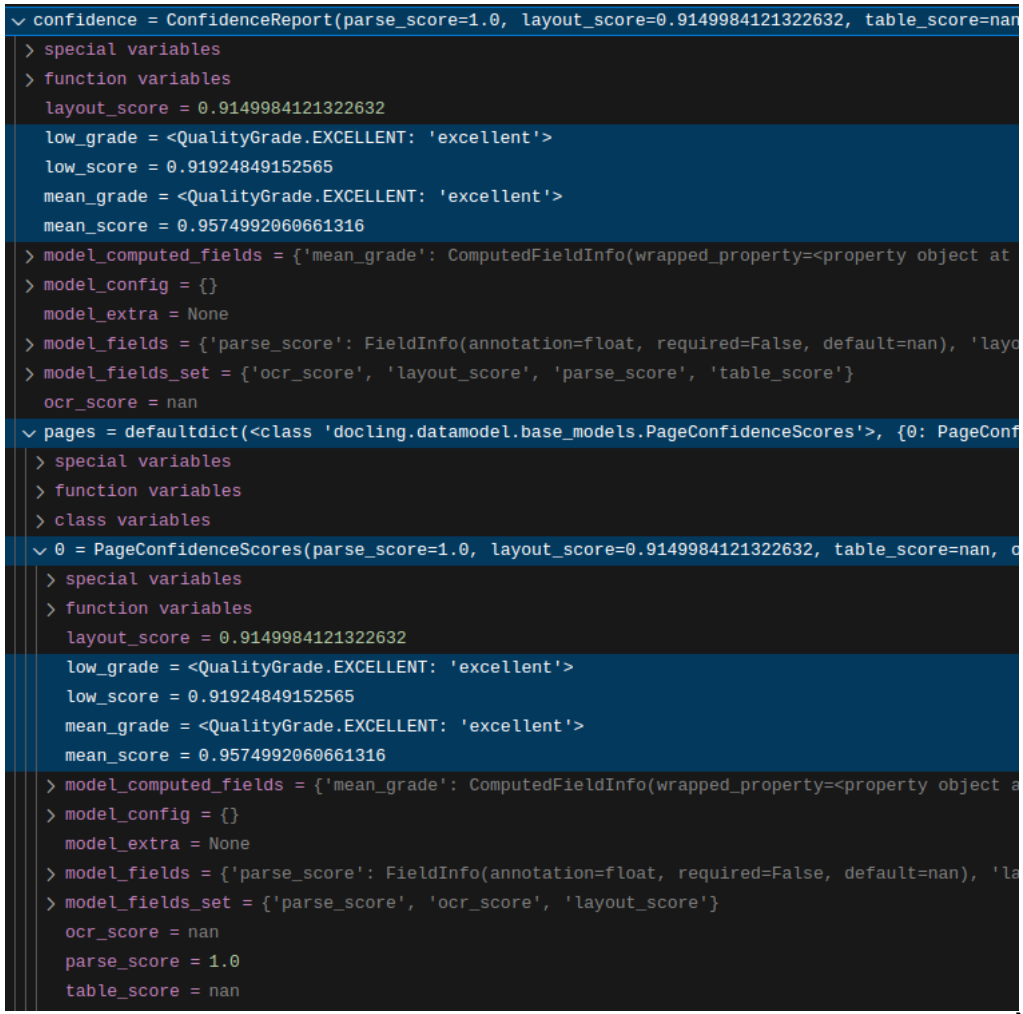
Confidence Score Components

- **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:

```python
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
```

YHK

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

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 ...

- Search **"Yogesh Haribhau Kulkarni"** on Google and follow me on LinkedIn and Medium
- Office Hours: Saturdays, 2 to 3 pm (IST); Free-Open to all; email for appointment.
- Email: yogeshkulkarni at yahoo dot com

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

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

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

YHK