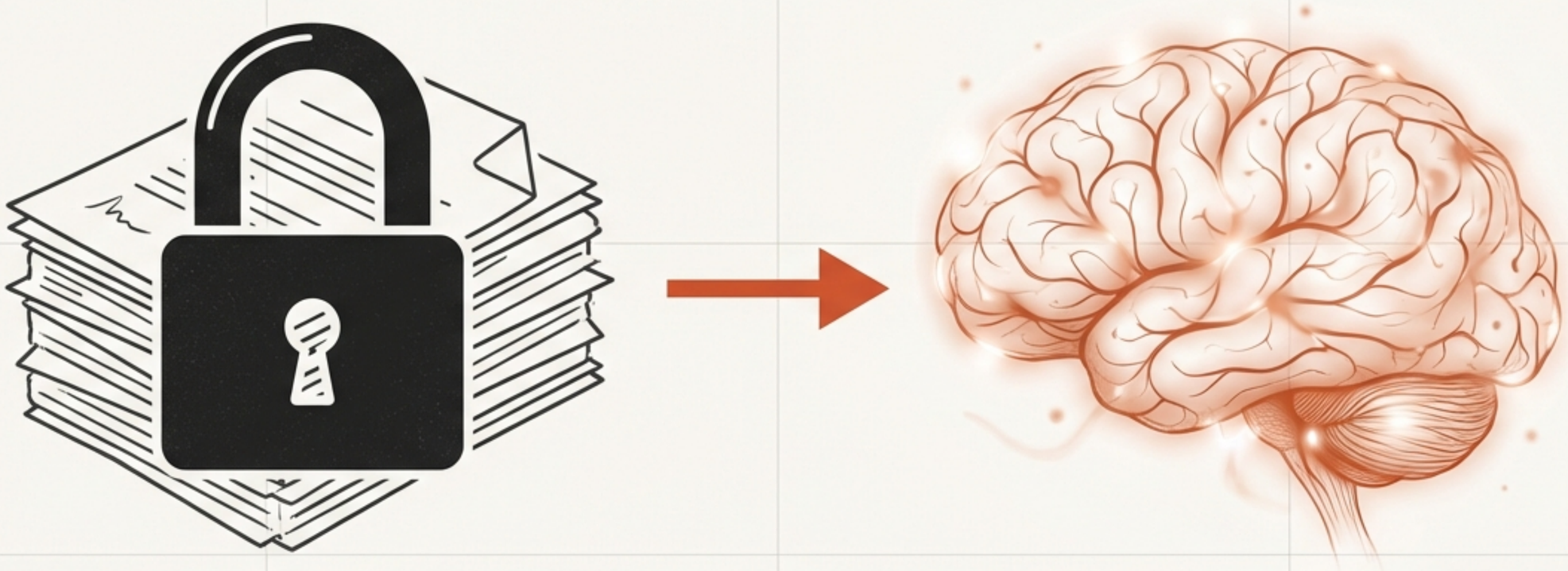
Your Most Valuable Data is Trapped.

Private documents—reports, invoices, research—hold critical insights.
How do you unlock them without compromising security?

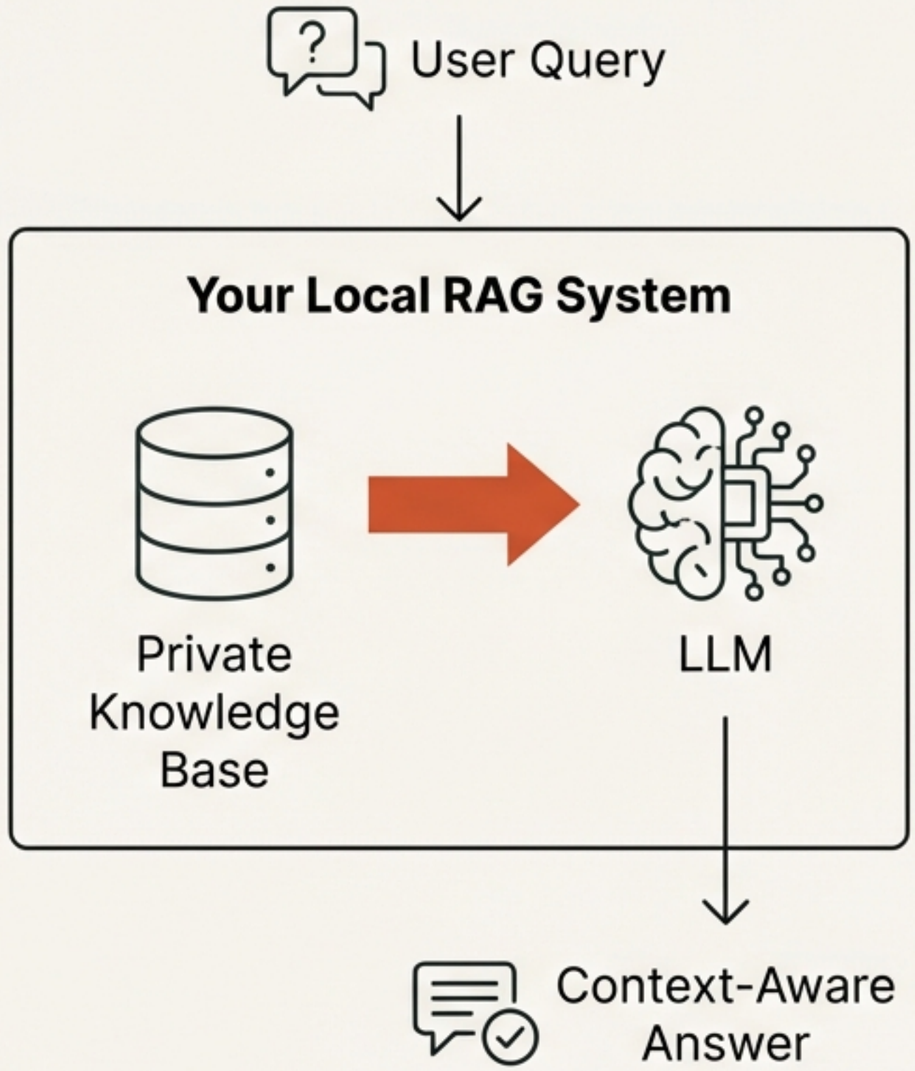


The challenge isn't a lack of data; it's a lack of secure access. Public AI models require you to send sensitive information over the internet. A local, private solution is the only way to maintain full control.

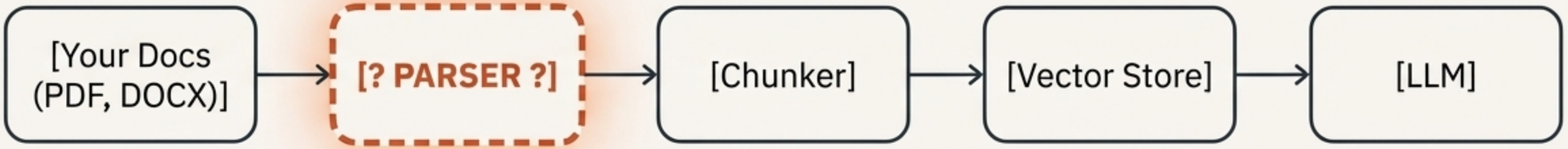
The Local RAG Promise: Privacy, Control, and Customization.

Retrieval-Augmented Generation (RAG) enhances a language model with your private knowledge base. By deploying it locally, you gain unmatched benefits:

- **Enhanced Privacy & Security**: Your sensitive data never leaves your secure environment. No third-party exposure, no risk of breaches.
- **Full Customization**: You have complete control over the architecture, from the models to the data processing pipeline.
- **Independence**: Your application runs without internet dependency, guaranteeing uninterrupted service.



Every RAG Pipeline Starts with One Critical Step

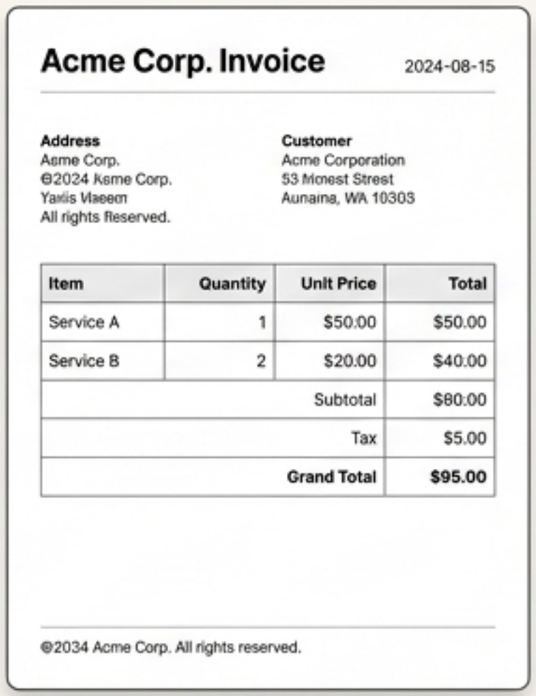


The journey from raw document to intelligent answer begins with parsing. This is where the system reads your files and extracts the text. However, this is the most common point of failure. If the parser can’t accurately understand the structure of your documents, the entire system is built on a flawed foundation.

Garbage In, Garbage Out: The Document Parsing Problem

"You would spend more time fixing parsers than building your RAG."

Source Document



Acme Corp. Invoice 2024-08-15

Address
Acme Corp.
©2024 Acme Corp.
Yanis Maeen
All rights reserved.

Customer
Acme Corporation
53 Monest Street
Aunaina, WA 10303

Item	Quantity	Unit Price	Total
Service A	1	$50.00	$50.00
Service B	2	$20.00	$40.00
		Subtotal	$80.00
		Tax	$5.00
		Grand Total	**$95.00**

©2034 Acme Corp. All rights reserved.

Standard Parser Output

Acme Corp. Invoice
Date: 2024-08-15
Item
Quantity
Service A 1
Service B 2
©2024 Acme Corp. All rights reserved.
Unit Price Total
Service $50.00
Service A $20.00
Service B $40.00
Subtotal $90.00
Tax $5.00
Grand Total $95.00...

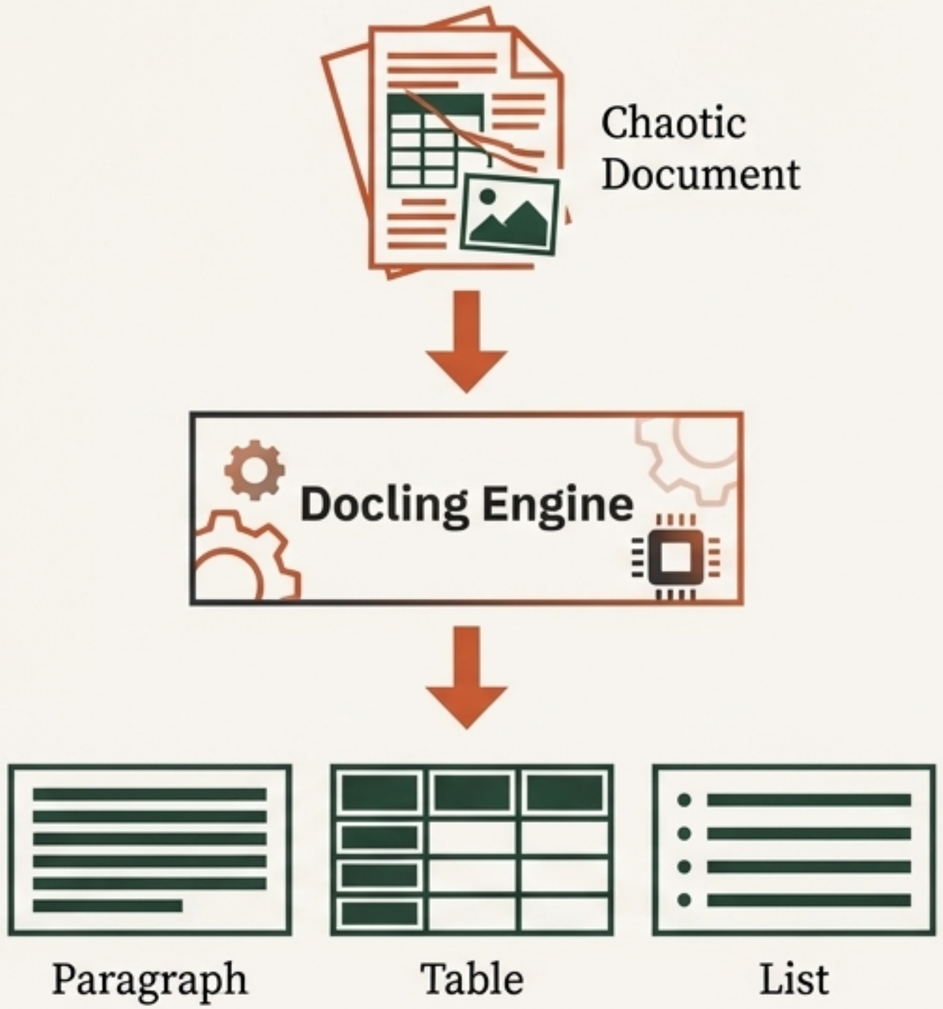
Most standard parsers fail to understand document layout. They strip away vital context from tables, lists, and multi-column formats, feeding your LLM noisy, unreliable data.

A Better Foundation with IBM Docling.

Docling

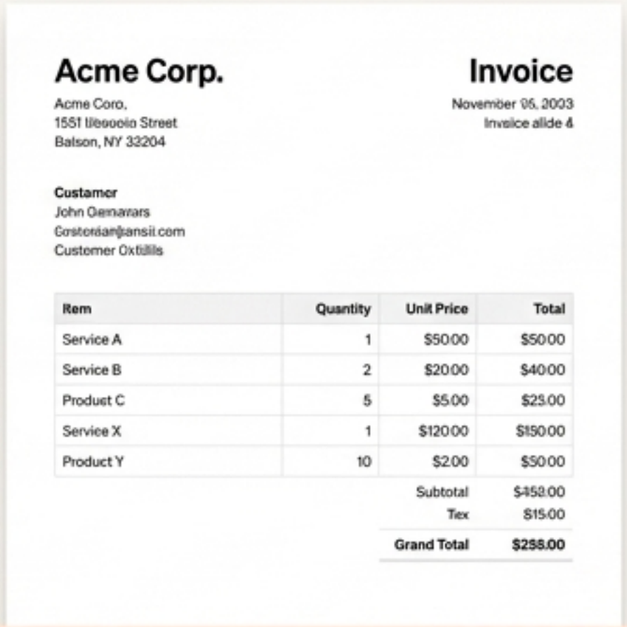
Docling is an open-source document parser from IBM Research designed to transform unstructured files into clean, structured data ready for AI pipelines. It's powered by specialized AI models to understand documents like a human would:

- **Layout Analysis:** Uses models like **DocLayNet** to identify page elements (text, tables, lists, figures).
- **Table Structure Recognition:** Employs **TableFormer** to accurately extract tabular data, preserving rows and columns.
- **OCR for Scanned PDFs:** Intelligently handles scanned documents and images.
- **Structured Output:** Converts complex files into clean, model-friendly Markdown or JSON.



From Document Chaos to Structured Clarity

Source Document



Acme Corp.

Acme Corp.
1551 Illegible Street
Balson, NY 33204

Invoice

November 05, 2023
Invoice #1

Customer
John Germanars
Customer@ansii.com
Customer Details

Item	Quantity	Unit Price	Total
Service A	1	$50.00	$50.00
Service B	2	$20.00	$40.00
Product C	5	$5.00	$25.00
Service X	1	$120.00	$150.00
Product Y	10	$2.00	$50.00
		Subtotal	$450.00
		Tax	$15.00
		Grand Total	**$255.00**

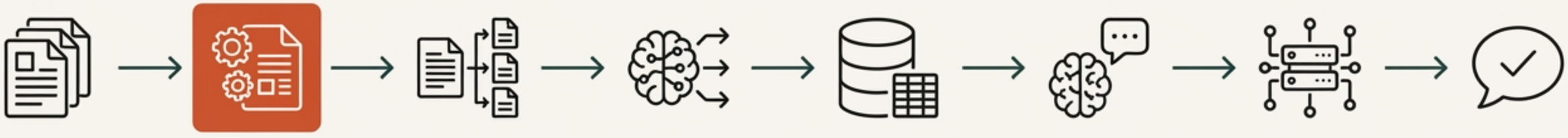
Docling Markdown Output

```
| Item      | Quantity | Unit Price | Total    |
|-----------|----------|------------|----------|
| Service A | 1        | $50.00     | $50.00   |
| Service B | 2        | $20.00     | $40.00   |
| Product C | 5        | $5.00      | $25.00   |
| Service X | 1        | $120.00    | $120.00  |
| Product Y | 10       | $2.00      | $20.00   |
```

With this clean output, the RAG system can now answer precise questions that were previously impossible:

- **Q: How many items are in this invoice?** → A: There are five items in the invoice.
- **Q: What is the total amount?** → A: The total amount is $255.00.
- **Q: What is the most expensive item?** → A: The most expensive item is Service X with a total cost of $120.

Our Complete Local RAG Blueprint



Docs

Docling

AI-powered parsing for clean, structured data

Chunker

Splits structured text into meaningful segments

Embedding Model (Ollama)

Converts text chunks into numerical vectors (via Ollama `nomic-embed-text`)

Vector Store (ChromaDB)

Indexes and stores vectors for fast retrieval (ChromaDB)

LLM (Ollama)

Generates answers based on retrieved context (via Ollama `mistral` or `llama3.2`)

API (Flask)

Serves the application to the user

Answer

Your Local Development Environment Setup.

Python & Project Setup

```
# 1. Create a project folder
and virtual environment

mkdir local-rag && cd local-rag
python3 -m venv venv
source venv/bin/activate
```

Core Dependencies

```
# 2. Install Vector DB &
Frameworks
pip install --q chromadb
pip install --q langchain
pip install --q langchain
langchain-community
langchain-text-splitters

# Note: Docling offers its own
LangChain/LlamaIndex loaders

pip install --q flask
```

The AI Engine (Ollama)

```
# 3. Install Ollama and pull
models
(Install from ollama.com)

ollama pull mistral
ollama pull nomic-embed-text
ollama serve
```

Bringing the Blueprint to Life with Code.

Snippet 1: embed.py - Ingesting and Storing Documents

```python
# embed.py                                  # <-- Use Docling for superior parsing
from langchain_docling import DoclingLoader
from langchain_text_splitters import MarkdownHeaderTextSplitter

def embed(file_path):
    # 1. Load & Parse with Docling
    loader = DoclingLoader(file_path=file_path)
    docs = loader.load()

    # 2. Split the clean Markdown output
    headers_to_split_on = [("#", "H1"), ("##", "H2")]
    splitter = MarkdownHeaderTextSplitter(headers_to_split_on)
    chunks = splitter.split_text(docs[0].page_content)

    # 3. Add to Vector Store
    db = get_vector_db() # ChromaDB instance
    db.add_documents(chunks)
    db.persist()
```

Snippet 2: query.py - Retrieving and Generating Answers

```python
# query.py
from langchain_community.chat_models import ChatOllama
from langchain.chains import create_retrieval_chain

def query(input_query):
    llm = ChatOllama(model="mistral")  # <-- Local LLM via Ollama
    db = get_vector_db()
    retriever = db.as_retriever()

    # Create a RAG chain to "stuff" context into the prompt
    rag_chain = create_retrieval_chain(retriever, ...)

    response = rag_chain.invoke({"input": input_query})
    return response["answer"]
```

Masterclass

Choosing the Right Chunking Strategy.

With clean, structured text from Docling, your chunking strategy becomes far more effective. The goal is to create segments that are semantically coherent. Here are three primary approaches:

Fixed-Size Chunking

- **How it works:** Simply splits text into chunks of a specific character or token count (e.g., 1000 tokens) with an optional overlap.
- **Pros:** Simple, fast, predictable.
- **Cons:** Can split sentences or ideas mid-thought, breaking context.

Recursive Chunking

- **How it works:** Attempts to split along a hierarchy of separators (e.g., `\n\n`, then \n`, then ``) to keep paragraphs and sentences intact.
- **Pros:** Better at preserving semantic boundaries than fixed-size.
- **Cons:** Still relies on simple characters, not true semantic understanding.

Semantic Chunking

- **How it works:** Groups sentences based on the similarity of their vector embeddings. Breaks occur when the semantic topic shifts.
- **Pros:** Creates the most contextually-aware chunks. Aligns with how models process information.
- **Cons:** Computationally more intensive.

From the Trenches: Real-World Troubleshooting & Optimization

Handling Conversion Failures

Problem
You see a generic `Failed to convert` error in Docling, especially in batch processing.

Common Causes
- Non-standard PDFs: Some files have vectorized text or complex internal structures.
- Scanned Documents: Require OCR, which can fail on low-quality images.

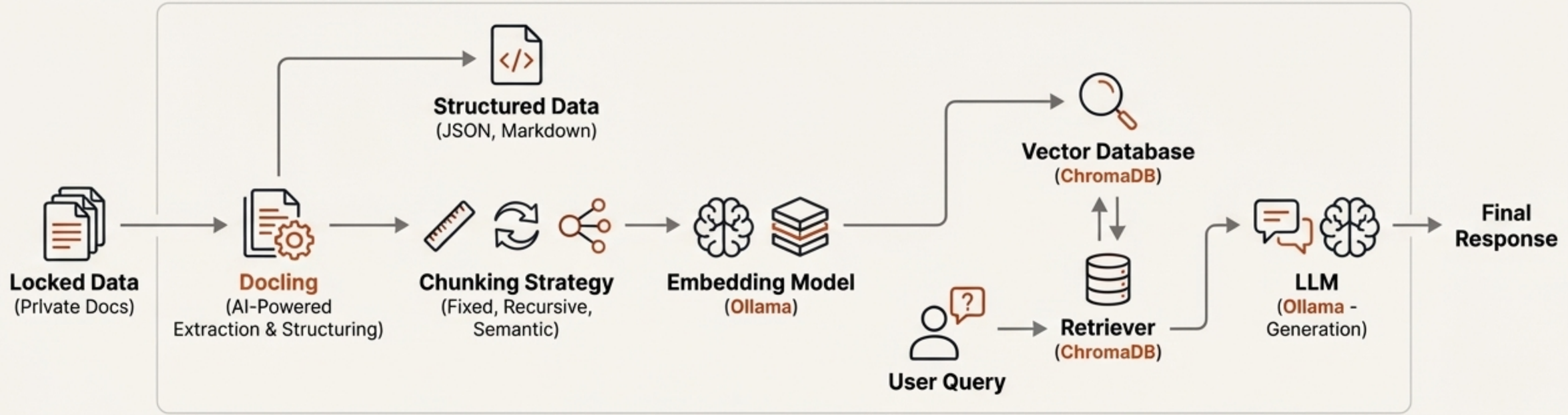
Pro-Tip
As a first step, try converting problematic PDFs to the **PDF/A** (Archive) format. This standardized format often resolves parsing issues by flattening complex elements.

Optimizing for Low-Resource Machines

Problem
Docling's AI models can be slow, taking several minutes for large documents.

Performance Levers
- **Turn off OCR:** If your documents are digital-native, disable OCR for a significant speed boost. (CLI: `--no-ocr`).
- **Skip Table Recognition:** If tables are not critical, disable this feature.
- **Switch PDF Backend:** Use the experimental `DoclingParseV2DocumentBackend` for a potential 10x speedup in PDF loading.

The Docling-Powered RAG Stack: From Locked Data to Production-Ready AI.



We started with a common but critical challenge: extracting value from private documents. The journey revealed that success hinges on getting the first step right.

1. **The Challenge:** Standard parsers create a 'garbage in, garbage out' problem.
2. **The Solution:** Docling provides an AI-powered, structured foundation.
3. **The Stack:** A fully local, private, and powerful RAG system built with best-in-class open-source tools like Ollama and ChromaDB.

"Mastering document ingestion isn't just a preliminary step; it's the core differentiator for building robust, reliable, and truly intelligent RAG applications."

Continue Your Journey: The Docling Ecosystem

The principles in this guide are the starting point. Docling integrates with a wide range of tools and frameworks, enabling even more advanced RAG workflows.

Docling Documentation

A deep dive into advanced options, vision models, and GPU support.

RAG with LlamaIndex

Official examples and best practices.

RAG with LangChain

Deep integration with the LangChain framework.

Advanced Vector Stores

Examples for RAG with Milvus, Qdrant, Weaviate, and more.

The GitHub Repo

Link to the main Docling GitHub project for source code and community discussions.

Explore the official examples to unlock capabilities like figure/table export, visual grounding with VLMs, and multimodal RAG.