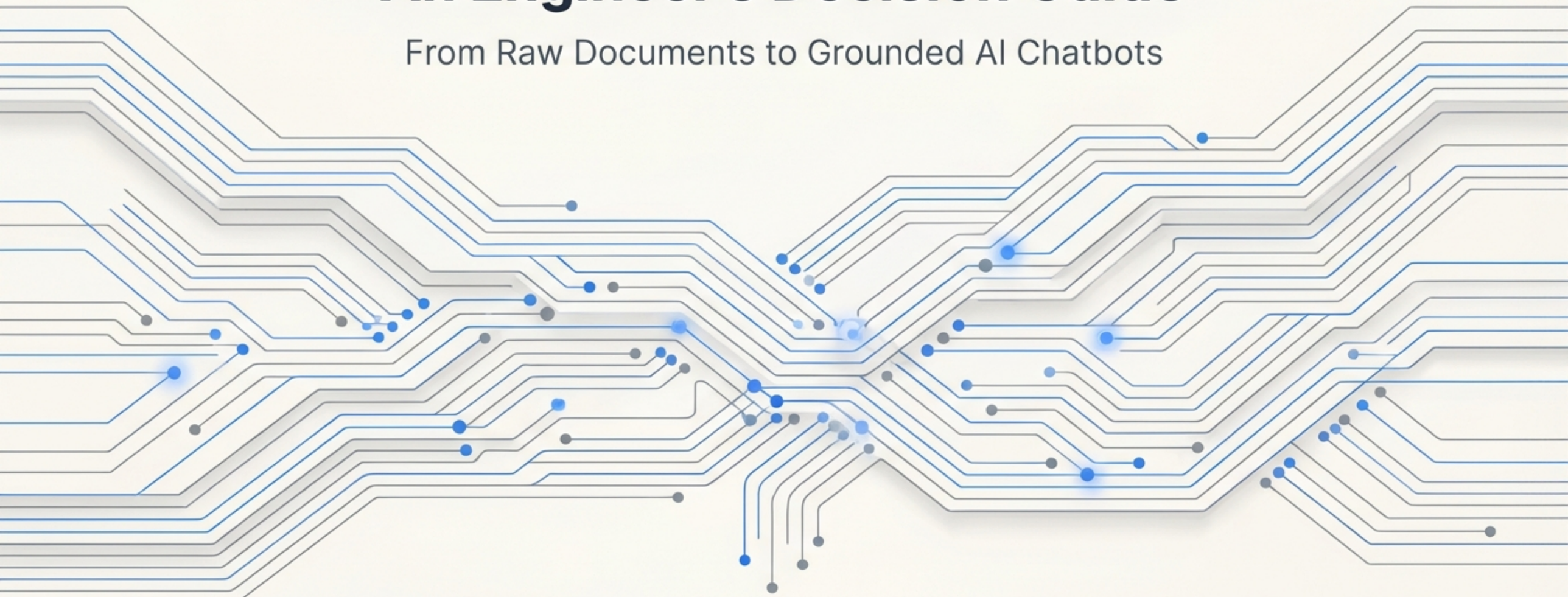
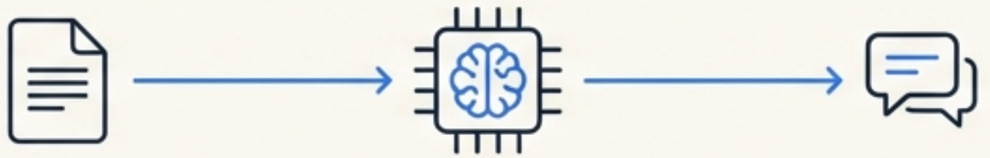
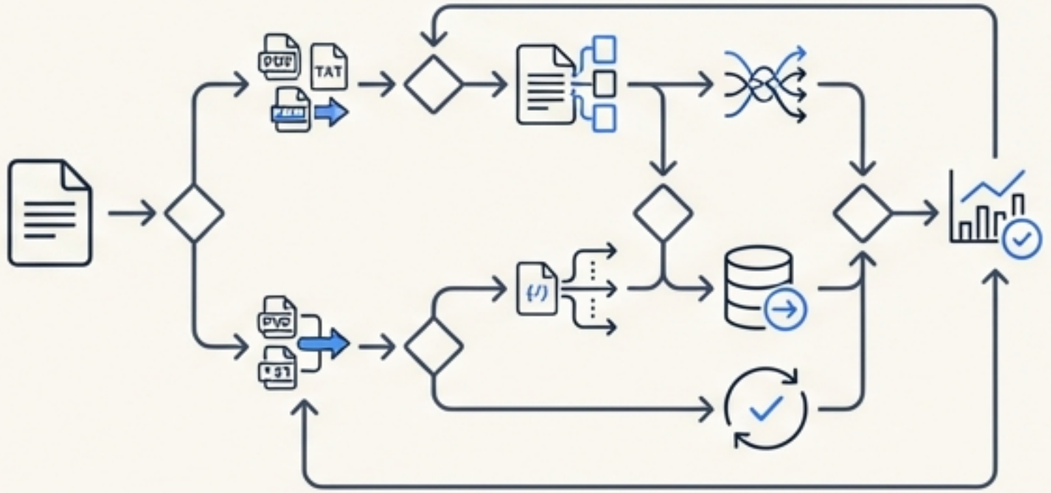
Production-Level RAG: An Engineer's Decision Guide

From Raw Documents to Grounded AI Chatbots



Based on the workshop by Vizuara

The Gap Between a 5-Minute Tutorial and a Production System

Tutorial	Production
	

Tutorials Show

Upload PDF → Ask Question

Production Reality Hides

- Complex File Parsing
- Strategic Chunking
- Embedding Model Selection
- Vector Store Trade-offs
- Continuous Evaluation (Evals)

This Guide's Goal

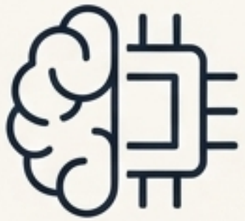
Equip you with the mental models to make the right engineering choices for *your* specific problem. The best engineers figure out the best solution for a *given* problem.

The Goal: Grounding an LLM in Factual, Private Knowledge

A client wants a nutritional chatbot that **only** uses information from a **1200-page "Human Nutrition"** textbook.

400,000-token PDF

The Naive Approach: Why not just feed the entire PDF to the LLM with every query?



LLM

The Inevitable Failures

Context Window Exceeded

A 400,000-token document won't fit in a 128k context window.

Prohibitive Cost

You're charged per token. Feeding a massive document on every query is financially unviable.

Hallucination Risk

If relevant info isn't in the context, the LLM defaults to its general knowledge, leading to ungrounded, incorrect answers.

The Solution: Retrieval-Augmented Generation (RAG)

The **“Open Book Exam” Analogy**: Instead of forcing the LLM to memorize the entire book, we enable it to quickly *look up* the most relevant passages before answering.

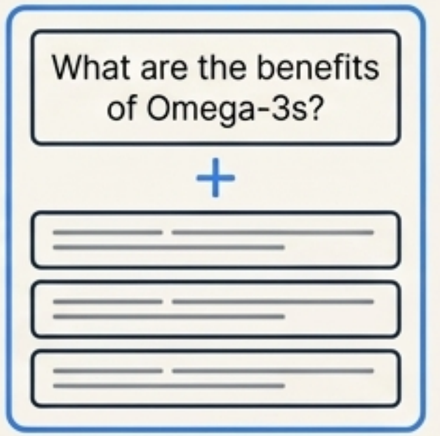
1. User Asks

“What are the benefits of Omega-3s?”

2. Retrieve

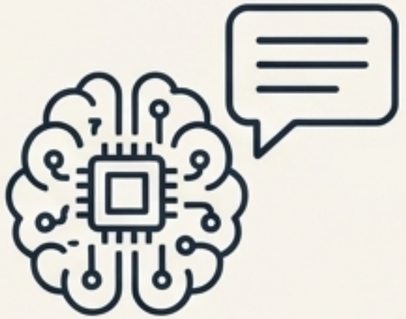
System searches the indexed textbook for the most relevant text snippets.

3. Augment



These snippets are added to the prompt alongside the original question.

4. Generate

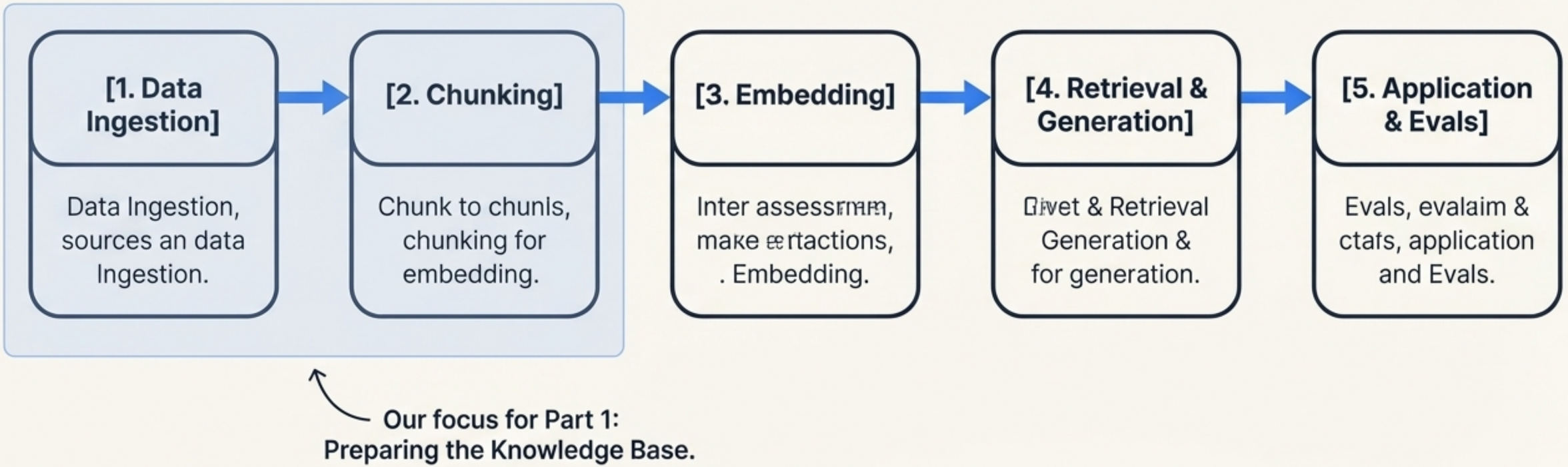


The LLM receives the question *and* the supporting context, then generates a grounded, factual answer.

Key Takeaway

This augments the generation process with retrieved knowledge, solving the context window, cost, and hallucination problems.

Our Roadmap: The Production RAG Pipeline



Station 1: Data Ingestion

How you parse your documents determines the quality of everything that follows. The right tool depends entirely on your source material's complexity. "It's not just a file, it's the foundation."

Simple Text

Scanned Image

Structured Report

Web Content

Engineer’s Choice #1: Selecting Your Document Parsing Toolkit

Which tool should you use to parse your documents?

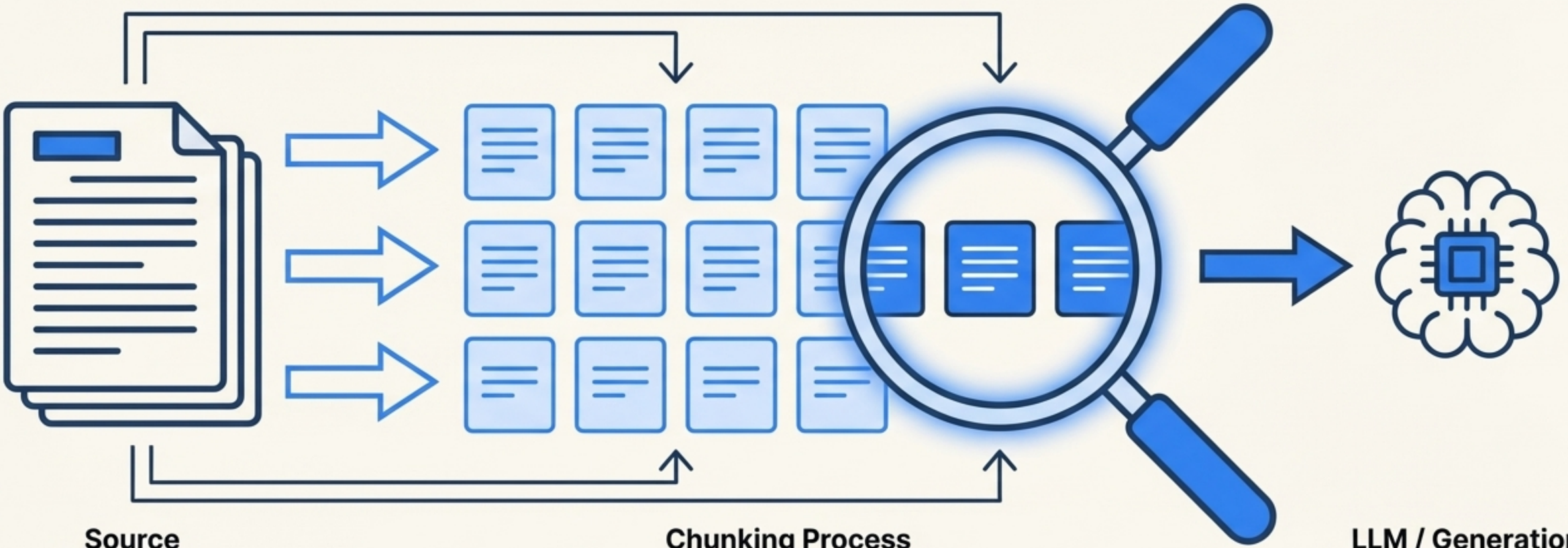
Tool	Best For	Key Trade-off
PyMuPDF	Simple, text-based digital PDFs.	Fast and efficient, but cannot read text within images.
Tesseract (OCR)	Scanned documents or images with text.	Essential for non-digital text, but slower and can have accuracy issues.
Docling	Complex documents with tables, schemas, and mixed content.	Powerful and structure-aware, but can be 50x slower than simpler tools.
Scraping Tools (Firecrawl, Puppeteer)	Web-based knowledge that needs to be collected first.	Necessary for online sources, but requires handling website structure and potential blocks.

Guiding Principle

Start with the simplest tool that meets your document’s complexity. Don’t use an elephant to kill an ant, even if you can.

Station 2: Chunking, The Most Critical Step

'Chunks' are the units of information the system will retrieve. The quality of your chunks directly determines the quality of your RAG system's answers. Bad chunks = bad retrieval = bad answers.



The Chunking Strategy Spectrum: Five Ways to Split a Document

There is no single ‘best’ strategy. The optimal choice is a trade-off between speed, cost, and contextual integrity, tailored to your document.

Engineer’s Choice #2: Selecting Your Chunking Strategy (Part 1)

How should you chunk your content?

IF your data is...

Large-scale, unstructured, and messy (e.g., web, web scrapes, logs)

THEN consider -> Fixed-Size Chunking.

Why: It’s fast, doesn’t require understanding the content, and can tolerate some loss of coherence.

IF your data is...

Lacking clear structure, but ideas flow logically (e.g., debate transcripts, educational videos)

THEN consider -> Semantic Chunking.

Why: It preserves the integrity of an idea by grouping semantically similar sentences. Computationally expensive.

Engineer’s Choice #2: Selecting Your Chunking Strategy (Part 2)

IF your data is...

Highly structured (e.g., legal docs, API documentation)

THEN consider -> Structural Chunking.

Why: The most intuitive approach; it leverages the document's inherent logic.

Risk: chunks can become too large.

IF your data is...

Structured, but with highly variable section lengths (e.g., research papers, textbooks)

THEN consider -> Recursive Chunking.

Why: The best of both worlds. It respects the document's structure (sections -> paragraphs -> sentences) but recursively splits any piece that exceeds a max size, ensuring consistency.

Case Study: The Impact of Chunking the Nutrition PDF

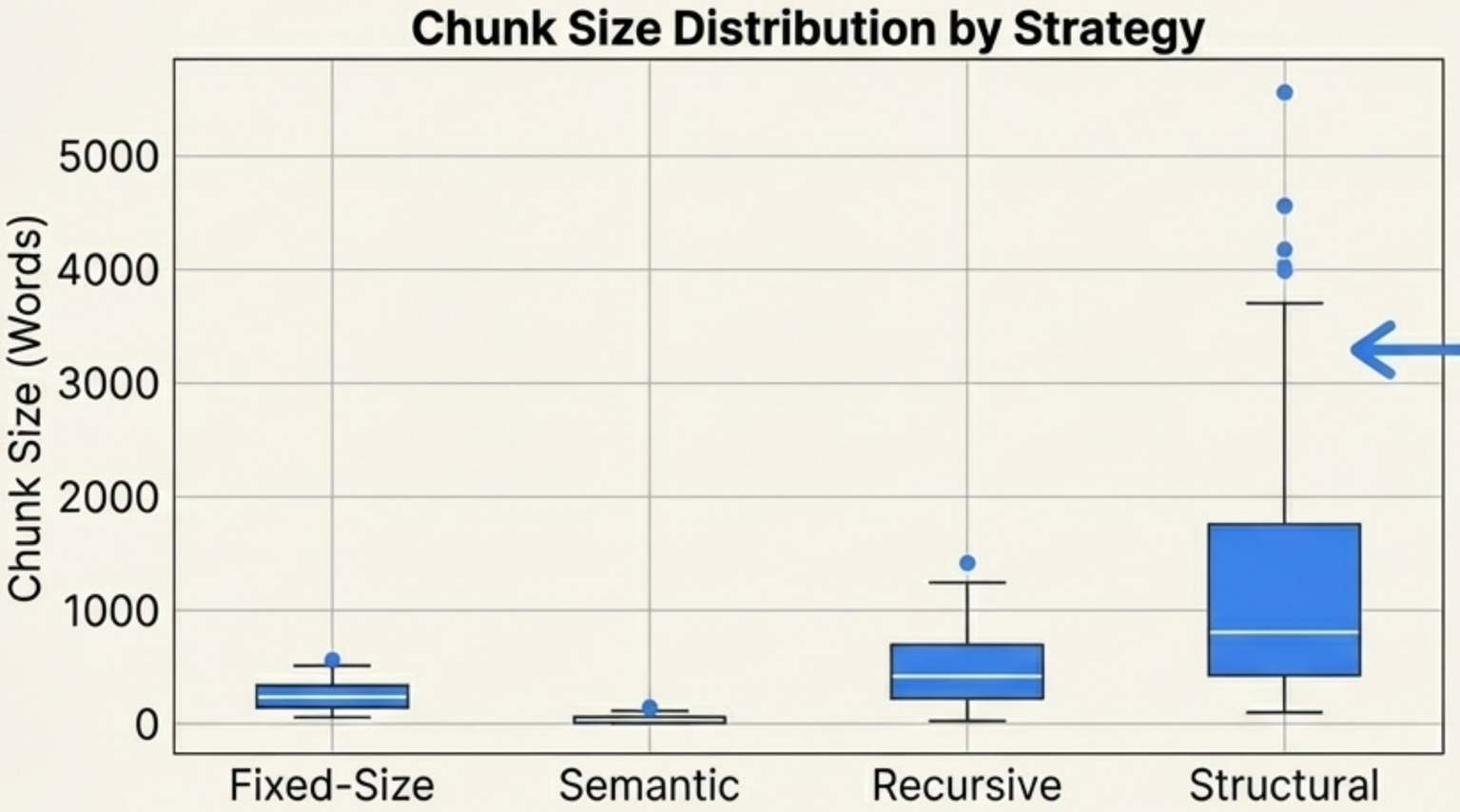
We applied four chunking strategies to the 1200-page nutrition PDF. The results reveal the extreme trade-offs of each choice before a single embedding is created.

Strategy	Number of Chunks	Avg. Chunk Size (words)	Chunk Size Variance
Fixed-Size	3,321	62	Low
Semantic	**12,006**	**16**	Low
Recursive	2,434	135	Moderate
Structural (Chapter-based)	**171**	**1,488**	**Very High**

Key Insight

Notice the extremes: Structural chunking gives only 171 meaningful chunks, but their massive size is a problem for context windows. Semantic chunking creates over 12,000 tiny, fragmented chunks, risking over-fragmentation and loss of context.

Visualizing the Chunk Size Distribution



Huge variance and large outliers. Problematic for downstream models.

Very small and compact. Risks over-fragmentation, losing meaning.

A balanced, consistent approach that respects structure.

Your First Two Engineering Decisions: A Summary

Key Takeaways

- **Data Ingestion is Foundational**
  Match your parsing tool to your document's complexity. A simple text document doesn't need a tool 50x slower.
- **Chunking is Mission-Critical**
  There is no single best strategy. For structured documents, Recursive Chunking often provides the best balance of contextual integrity and size consistency.
- **Evaluate Before You Embed**
  Simple statistical analysis (chunk count, size, variance) can reveal major issues with your strategy before you commit to the expensive downstream steps of a full pipeline.

The Path Forward

With our knowledge base prepared, the next engineering choices involve selecting the right **Embedding Models** and **Vector Stores** to enable efficient retrieval.