THE FIELD MANUAL FOR FORWARD DEPLOYED AI ENGINEERS

THE RAG BOOK

Retrieval, Agents & Multi-Agent Systems
From First Principles to Production Code

17 MODULES · 646 PAGES · RUNNABLE CODE

PRITHVI DATLA

KILOBYTE COLLECTIVE

The RAG Book

Retrieval, Agents & Multi-Agent Systems — From First Principles to Production Code

The Field Manual for Forward Deployed AI Engineers

First Edition, April 2026 — Preview Edition

To a lifetime with him. The Dog in me.

Foreword

A foreword would be a distraction here.

I have produced this from repeated failure and ruthlessly cutting noise from my own learning and builds.

The only ask I have — finish the book. No distraction. No excuse.

All my best,

Prithvi

Table of Contents

The RAG Book: Retrieval, Agents & Multi-Agent Systems

The Field Manual for Forward Deployed AI Engineers

Welcome

You are about to learn the most consequential set of techniques in applied AI engineering today: how to make large language models work with *your* data, *your* documents, and *your* business logic.

This book is written for **Forward Deployed AI Engineers (FDAIEs)** — the practitioners who sit between customers and codebases, translating messy real-world requirements into shippable systems. Whether that's your current title or the role you want to grow into, the patterns, trade-offs, and failure modes in this book are what you need to do the job well.

This is not a theory course. Every module includes runnable Python code that you execute as you learn. By the end of this book, you will have built RAG pipelines, embedding strategies, reranking systems, evaluation frameworks, and multi-agent workflows — all with working code you run yourself. The Companion Repository (included in the Full Professional Bundle) adds four structured labs and a capstone project that tie everything together into production-grade systems.

The program is designed for two audiences simultaneously:

- **Product Leaders** who need to make investment decisions, write PRDs, and evaluate AI capabilities without writing code.
- **Engineers** who need to build, ship, and maintain these systems in production.

Every module serves both tracks. You will never feel like you are sitting through content meant for someone else.

Program Goals

By the end of this program, you will be able to:

1. **Explain** why RAG exists and when it is the right (or wrong) approach.
2. **Design** embedding and chunking strategies that maximize retrieval quality.
3. **Build** end-to-end RAG pipelines with production error handling using runnable code.
4. **Evaluate** retrieval and generation quality with quantitative metrics.
5. **Select** the right vector database, embedding model, and reranking strategy for your use case.
6. **Architect** multi-agent systems that decompose complex tasks.
7. **Estimate** costs, latency, and infrastructure requirements for AI systems.
8. **Identify** failure modes before they reach production.

What Ten Days Does Not Give You

This program will make you competent. It will also make you dangerous.

After ten days, you will know enough to build a working RAG system, deploy an agent pipeline, and evaluate quality with real metrics. You will be able to hold your own in architecture discussions and make informed vendor decisions.

Here is what you will NOT have after ten days:

• **Production battle scars.** You have not yet debugged a hallucination at 3 AM that is costing a customer $10,000 per hour. That experience teaches things no curriculum can.

• **Scale intuition.** Your experiments run on one document. Production systems run on millions. The failure modes are qualitatively different — index drift, embedding staleness, query distribution shift, and cost runaway only emerge at scale.

• **Organizational muscle.** Shipping AI in a real company requires navigating legal review, compliance sign-off, stakeholder alignment, and user research. This program teaches the technical work. The organizational work is at least as hard.

• **Deep specialization.** Each module in this program could be a semester-long course. We trade depth for breadth deliberately — you need the full picture before you can specialize.

The gap between "I completed the program" and "I can run this in production unsupervised" is measured in months, not days. Respect that gap. Seek code review. Run postmortems when things break. Pair with experienced engineers on your first production deployment.

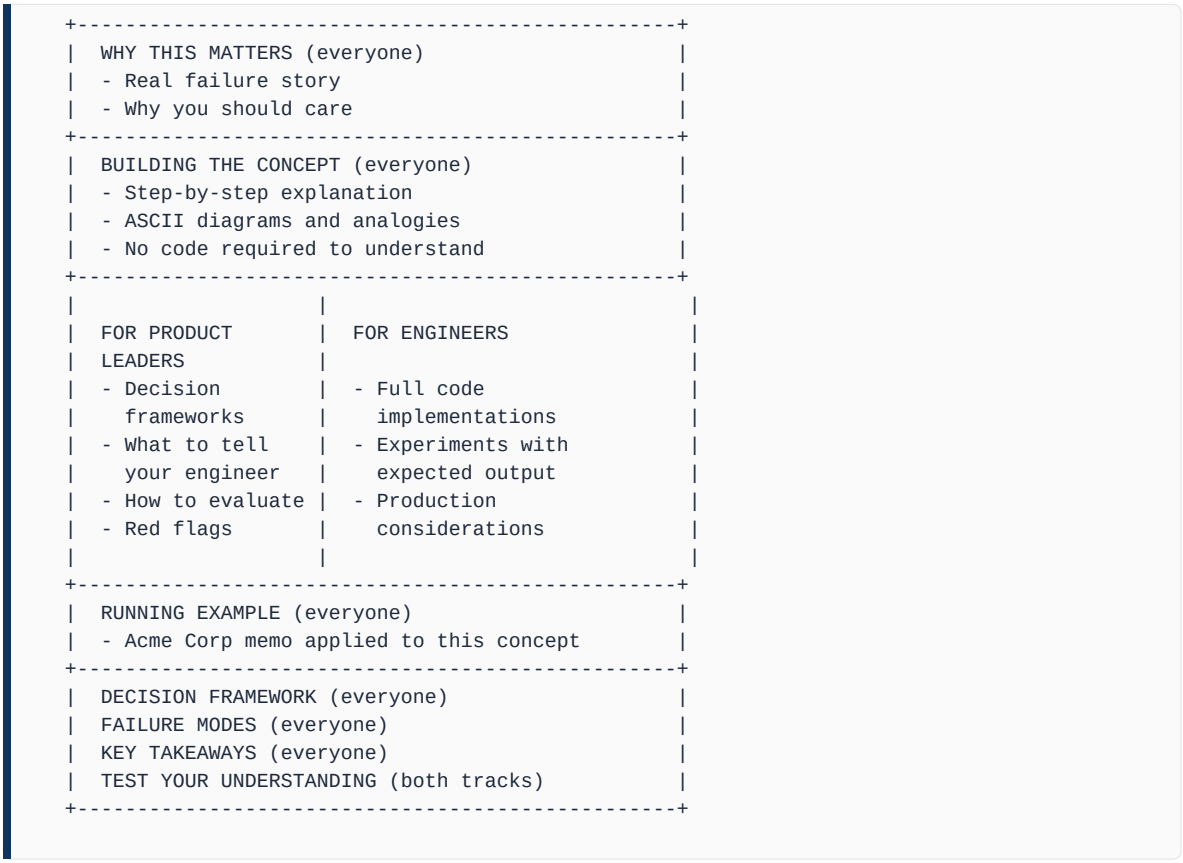
This program gives you the map. Production gives you the territory.

10-Day Schedule

Day	Modules	Focus
1	M01: RAG Foundations	RAG Foundations
2	M02: The RAG Pipeline, M03: Vector Databases	Pipeline & Vector Databases
3	M04: Chunking Strategies	Chunking Strategy
4	M05: Embedding Models, M06: Query Enhancement	Embeddings & Query Enhancement
5	M07: Multi-Query RAG, M08: Hybrid Search	Multi-Query & Hybrid Search
6	M09: Reranking, M10: GraphRAG	Reranking & GraphRAG
7	M11: Multimodal RAG, M12: Structured RAG	Multimodal & Structured RAG
8	M13: Evaluation & Observability	Evaluation & Observability
9	M14: Agent Foundations, M15: Orchestration	Agent Foundations & Orchestration
10	M16: Agentic RAG, M17: Security	Agentic RAG & Security

How to Use the Dual Tracks

Every module after this introduction follows a consistent structure:



If you are a Product Leader: Read every section. Skip the code blocks if you want, but read the prose around them. The "For Engineers" sections contain insights about what is actually hard and why things break – that context makes you a better product leader.

If you are an Engineer: Read every section. The "For Product Leaders" sections teach you how to communicate technical decisions to stakeholders. The decision frameworks will save you from building the wrong thing.

The Running Example: Acme Corp Q3 Strategy Memo

Throughout all 16 modules, we use the same document to demonstrate every concept. This makes it easy to compare techniques against each other – the input never changes, only the approach.

Read this memo carefully. You will see it referenced in every module.

```
ACME CORPORATION — Q3 STRATEGIC PLANNING MEMO
Prepared by: Sarah Chen, VP of Strategy
Date: July 15, 2025
Classification: Internal — Executive Team Only

EXECUTIVE SUMMARY

Acme Corporation enters Q3 2025 with strong momentum from Product Alpha's
enterprise launch but faces headwinds from increased competition in the
mid-market segment. This memo outlines our strategic priorities, resource
allocation, and risk mitigation plans for the quarter.

PRODUCT PORTFOLIO UPDATE

Product Alpha (Enterprise Analytics Platform)
Q2 revenue: $12.4M (18% above forecast). Alpha secured 23 new enterprise
contracts including Fortune 500 wins at Meridian Healthcare ($1.2M ACV) and
Titan Manufacturing ($890K ACV). Current ARR run rate: $48.2M. The v3.2
release added real-time dashboard streaming, which was the #1 feature
request from enterprise buyers. Engineering headcount: 34 FTEs.
Key risk: Dataview Inc. launched a competing product at 40% lower price
point. Three pipeline deals ($2.1M combined) are now in competitive
review. Sales team reports Dataview is winning on price but losing on
reliability and integration depth.

Product Beta (SMB Reporting Tool)
Q2 revenue: $3.8M (6% below forecast). Churn increased to 4.2% monthly,
up from 2.8% in Q1. Root cause analysis points to poor onboarding
experience — 62% of churned customers never completed initial setup.
Customer satisfaction (CSAT) dropped from 4.1 to 3.6. We are launching
"Beta Quick Start" program in Q3 with dedicated onboarding specialists.
Target: reduce churn to 2.5% by end of Q3.
Budget allocation: $1.8M for onboarding improvement, $600K for UX
redesign of setup wizard.

Product Gamma (AI Data Assistant — New)
Status: Private beta with 12 design partners. Launch planned for September
15, 2025. Early feedback is positive — 9 of 12 partners rated the
natural language query feature as "transformative." Pricing strategy under
review: considering $45/user/month for teams, $120/user/month for
enterprise with advanced governance features.
Competitive landscape: Four well-funded startups in this space. Our
advantage is deep integration with Alpha's data layer. Estimated TAM:
$2.8B by 2027.
Go-to-market plan requires $3.2M Q3 investment across product marketing
```

```
($1.4M), developer relations ($800K), and launch events ($1M).

FINANCIAL OUTLOOK

Q3 revenue target: $18.6M (combined). Operating margin target: 12%.
Headcount plan: 14 new hires across engineering (8), sales (4), and
customer success (2). Total compensation budget increase: $2.4M
annualized.

RISK REGISTER

1. Dataview pricing pressure on Alpha (Impact: High, Likelihood: Medium)
2. Beta churn trajectory if onboarding fix underperforms (Impact: Medium,
   Likelihood: Medium)
3. Gamma launch delay if API stability issues persist (Impact: High,
   Likelihood: Low)
4. Macroeconomic slowdown affecting enterprise sales cycles (Impact: High,
   Likelihood: Medium)
5. Key person risk — 3 senior engineers on Alpha have received external
   offers (Impact: High, Likelihood: Medium)
```

Companion Repository

All code examples, lab starter files, sample datasets (including the Acme Corp Q3 Strategy Memo), and environment templates are available in the companion repository:

> ```
> https://github.com/kilobyte-collective/the-rag-book
> ```

Clone it before Day 1:

```
git clone https://github.com/kilobyte-collective/the-rag-book.git
cd the-rag-book
```

The repository also contains a **"What Changed Since April 2026"** changelog that tracks model updates, pricing changes, and framework breaking changes. Check it periodically — see Appendix B for details on how to keep your setup current.

Environment Setup

A Note on Model Versions

All code examples in this book use `claude-sonnet-4-6-20250514` as the model string. This is the latest Claude Sonnet snapshot at the time of writing. **Replace this with the current model version when you run the code.** Model aliases change — always pin to a specific snapshot version in production (see Appendix B for version pinning discipline).

Check the companion repository for the latest tested model versions.

Required Software

Software	Version	Purpose
Python	3.11+	Runtime for all code examples
pip	23.0+	Package management
git	2.30+	Version control for lab work
VS Code	Latest	Recommended editor (any editor works)

API Keys Required

You will need accounts and API keys from four providers. Set them up **before Day 1** – account creation can take time.

Provider	Key Name	Sign Up URL	Free Tier?
Anthropic	`ANTHROPIC_API_KEY`	console.anthropic.com	Yes ($5)
OpenAI	`OPENAI_API_KEY`	platform.openai.com	Yes ($5)
Pinecone	`PINECONE_API_KEY`	app.pinecone.io	Yes
Cohere	`COHERE_API_KEY`	dashboard.cohere.com	Yes

Environment Configuration

Create a `.env` file in your project root (never commit this file):

```
# .env
ANTHROPIC_API_KEY=sk-ant-...
OPENAI_API_KEY=sk-...
PINECONE_API_KEY=pcsk_...
COHERE_API_KEY=...
```

Python Environment Setup

```
# Create and activate virtual environment
python3.11 -m venv ai_engineering
source ai_engineering/bin/activate  # macOS/Linux
# ai_engineering\Scripts\activate   # Windows

# Install all required packages
pip install anthropic openai pinecone langchain langchain-openai \
    llama-index llama-index-vector-stores-pinecone \
    cohere numpy scipy scikit-learn tiktoken python-dotenv \
    pandas matplotlib tqdm
```

Verify Your Setup

Run this script to confirm everything is working:

```
"""verify_setup.py -- Run this before Day 1 to check your environment."""

import sys
print(f"Python version: {sys.version}")
assert sys.version_info &gt;= (3, 11), "Python 3.11+ required"

# Check packages
packages = [
    "anthropic", "openai", "pinecone", "langchain",
    "llama_index", "cohere", "numpy", "scipy",
    "sklearn", "tiktoken", "dotenv"
]
for pkg in packages:
    try:
        __import__(pkg)
        print(f"  {pkg}: OK")
    except ImportError:
        print(f"  {pkg}: MISSING -- run pip install {pkg}")

# Check API keys
import os
from dotenv import load_dotenv
load_dotenv()
```

```
keys = [
    "ANTHROPIC_API_KEY",
    "OPENAI_API_KEY",
    "PINECONE_API_KEY",
    "COHERE_API_KEY",
]
for key in keys:
    val = os.getenv(key, "")
    if val:
        print(f"  {key}: set ({val[:8]}...)")
    else:
        print(f"  {key}: NOT SET")

# Quick API test
from anthropic import Anthropic
client = Anthropic()
response = client.messages.create(
    model="claude-sonnet-4-6-20250514",
    max_tokens=50,
    messages=[{"role": "user", "content": "Say 'Setup verified!' and nothing else."}]
)
print(f"\nClaude says: {response.content[0].text}")
print("\nAll checks passed. You are ready for Day 1.")
```

Prerequisites

For Product Leaders

- Familiarity with AI/ML concepts at a high level (you know what a language model is, you have used ChatGPT or Claude)
- Experience writing PRDs or product specifications
- Comfort reading technical diagrams (you do not need to write code)
- Understanding of basic metrics: latency, throughput, cost per query

For Engineers

- Proficiency in Python (functions, classes, async/await)
- Experience calling REST APIs (you have used `requests` or similar)
- Familiarity with JSON data structures
- Basic understanding of vectors and similarity (linear algebra 101)
- Experience with at least one cloud provider (AWS, GCP, or Azure)

Nice to Have (Not Required)

- Previous experience with the OpenAI or Anthropic APIs
- Familiarity with LangChain or LlamaIndex
- Understanding of transformer architecture
- Experience with database systems (SQL or NoSQL)

How to Get the Most Out of This Program

1. **Run every code example.** Reading code is not the same as running it. The experiments are designed to build intuition you cannot get from slides.

1. **Break things on purpose.** Each module includes experiments that show what failure looks like. Those experiments teach more than the happy-path examples.

1. **Pair up across tracks.** If you are a product leader, sit next to an engineer. If you are an engineer, sit next to a product leader. The best AI teams are the ones where both sides understand each other.

1. **Take notes on the Acme Corp memo.** As we apply different techniques to the same document, you will start to see patterns. Those patterns are the real curriculum.

1. **Ask "what would break?" constantly.** Production AI systems fail in ways that are different from traditional software. Developing an instinct for failure modes is the most valuable skill you will take away.

Module Index

Day 1: RAG Foundations

Module	Title	Key Question
M01	RAG Foundations	Why do LLMs need your documents?

Day 2: Pipeline & Vector Databases

Module	Title	Key Question
M02	The RAG Pipeline	What are the 5 stages from doc to answer?
M03	Vector Databases & Indexing	Where do your vectors live?

Day 3: Chunking Strategy

Module	Title	Key Question
M04	Chunking Strategies	How do you split documents intelligently?

Day 4: Embeddings & Query Enhancement

Module	Title	Key Question
M05	Embedding Models	How do you turn text into vectors?
M06	Query Enhancement	How do you improve query quality?

Day 5: Multi-Query & Hybrid Search

Module	Title	Key Question
M07	Multi-Query RAG	What if one query is not enough?
M08	Hybrid Search	How do you combine keyword and vector search?

Day 6: Reranking & GraphRAG

Module	Title	Key Question
M09	Reranking	How do you improve retrieval quality?
M10	GraphRAG	How do you leverage knowledge graphs?

Day 7: Multimodal & Structured RAG

Module	Title	Key Question
M11	Multimodal RAG	What about images, tables, and charts?
M12	Structured RAG	How do you handle structured data?

Day 8: Evaluation & Observability

Module	Title	Key Question
M13	Evaluation & Observability	How do you measure quality automatically?

Day 9: Agent Foundations & Orchestration

Module	Title	Key Question
M14	Agent Foundations	What makes an AI agent?
M15	Multi-Agent Orchestration	How do agents collaborate on tasks?

Day 10: Agentic RAG & Security

Module	Title	Key Question
M16	Agentic RAG	What if one retrieval is not enough?
M17	Security & Guardrails	How do you prevent harmful outputs?

Hands-On Labs & Capstone (Companion Repository)

The Full Professional Bundle includes four hands-on projects in the Companion Repository:

Project	What You Build	Applies Modules
Lab 01	Your first RAG pipeline from scratch	M01-M06
Lab 02	Advanced RAG with reranking, hybrid search, RAGAS eval	M07-M13
Lab 03	3-agent intelligence pipeline with LangGraph	M14-M17
Capstone	4-agent RFP/SOW response generator	All modules

These labs are available at **theragbook.com** as part of the $109 Full Professional Bundle.

Conventions Used in This Material

- `monospace` indicates code, file names, or terminal commands.
- **Bold** indicates key terms on first use.
- *Italic* indicates emphasis or book/paper titles.
- Code blocks with file names (e.g., `# no_rag.py`) are complete, runnable scripts. You can copy-paste them into your editor and run them.
- Lines starting with `>>>` show expected output.
- The model `claude-sonnet-4-6-20250514` is used for all Claude API calls.
- The model `text-embedding-3-small` is used for all OpenAI embedding calls.

A Note on Costs

Running all code examples in this program will cost approximately:

Provider	Estimated Cost	What It Covers
Anthropic	$3 - $8	Claude API calls across all modules
OpenAI	$1 - $3	Embedding generation
Pinecone	$0 (free tier)	Vector storage and retrieval
Cohere	$0 (free tier)	Reranking experiments
Total	**$4 - $11**	**Complete program**

If you run experiments multiple times or use larger documents, costs may be higher. All scripts include cost estimation where possible.

Let's Begin

Turn to **M01: RAG Foundations** to start building your understanding of why the most important AI systems in production today are not just language models – they are language models connected to your data.

The RAG Book: Retrieval, Agents & Multi-Agent Systems First Edition – April 2026

Day 1

RAG Foundations

Module 1 (Full)

M01: RAG Foundations – Why LLMs Need Your Documents

Why this matters

In March 2024, the VP of Finance at a Series C fintech company asked their internal AI chatbot a simple question: "What was Product Alpha's Q2 revenue?"

The chatbot confidently replied: "$8.5M, reflecting a 12% increase from Q1."

The actual number was $12.4M. The chatbot had hallucinated – not a wild, obviously wrong answer, but a plausible-sounding number that was off by $3.9M. The kind of wrong that slides past a quick glance.

The VP used that number in a board deck. The CFO caught it during the live presentation. The board lost confidence in the data team. The AI project was shelved for six months. Three engineers were reassigned. The company went back to manually copying numbers from spreadsheets into slides.

This failure cost roughly $400K in lost productivity, delayed AI adoption, and damaged credibility. And it was entirely preventable.

The root cause was not that the language model was bad. It was that the model had never seen the company's internal documents. It was generating answers from its training data – a statistical model of the internet – and the internet does not contain your Q2 revenue numbers.

Retrieval-Augmented Generation (RAG) solves this problem. Instead of asking the LLM to remember your data, you *retrieve* the relevant documents first, *augment* the prompt with those documents, and then *generate* an answer grounded in real evidence.

The VP's question, answered with RAG, would have included the actual memo as context. The model would have quoted "$12.4M (18% above forecast)" because that is what the document says. No hallucination. No board embarrassment. No shelved project.

This module teaches you how RAG works, why it exists, and when to use it.

Building the concept from scratch

What LLMs actually know (and don't know)

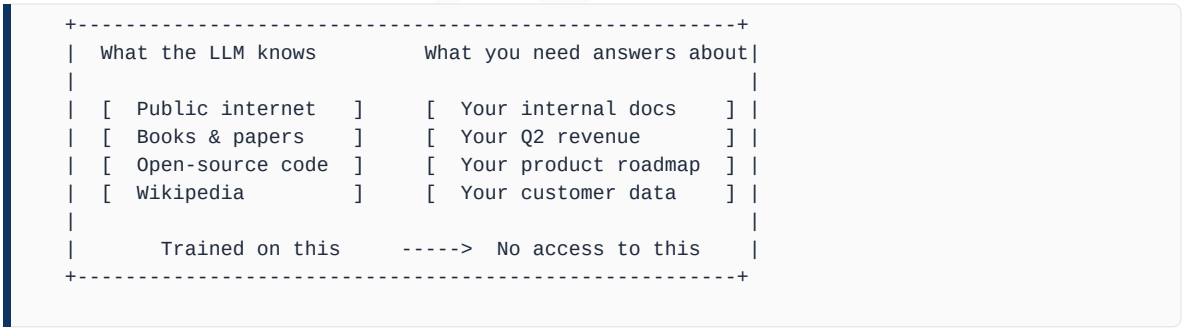
A large language model like Claude is trained on a massive corpus of public text: books, websites, code repositories, academic papers. During training, the model learns patterns in language – how words relate to each other, how arguments are structured, how code works.

But here is the critical insight: **the model's knowledge is frozen at training time.** Claude does not know what happened yesterday. It does not know your company's revenue. It does not have access to your internal Notion pages, your Confluence wiki, or the memo your VP wrote last Tuesday.

When you ask a question about something outside its training data, the model has two options:

1. Say "I don't know" (ideal, but models are not great at this)
2. Generate a plausible-sounding answer from patterns (hallucination)

Option 2 is the default behavior, and it is dangerous precisely because hallucinated answers *sound* confident and correct.



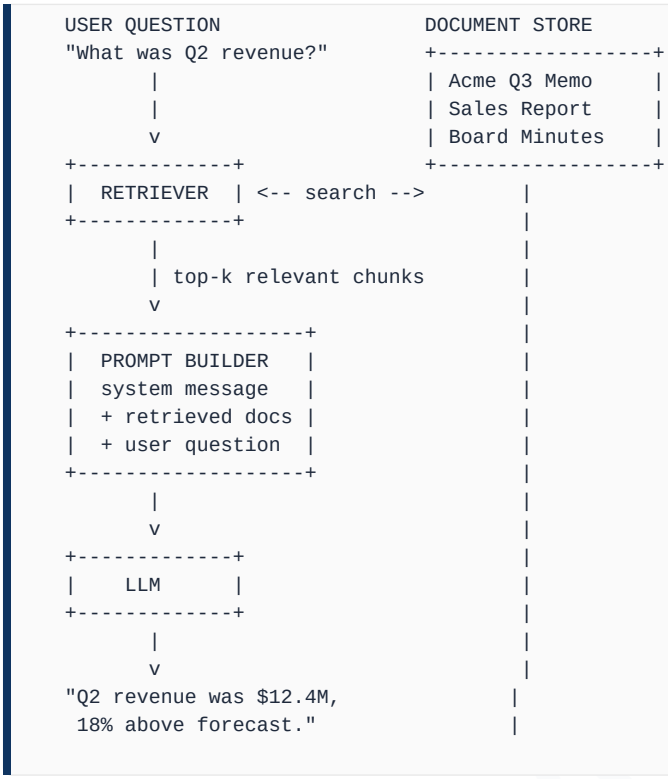
The RAG solution: Retrieve, Augment, Generate

RAG bridges this gap with a three-step process:

Step 1: Retrieve. When a user asks a question, search your document store for the most relevant passages. This is like a librarian pulling the right books off the shelf before answering your question.

Step 2: Augment. Take those retrieved passages and insert them into the prompt, right alongside the user's question. Now the model can *see* the relevant information.

Step 3: Generate. The model reads the retrieved passages and generates an answer grounded in that evidence. Instead of hallucinating, it quotes and paraphrases the actual source material.



What the LLM actually sees

This is the most important concept in RAG: understanding what the model's input looks like. When you use RAG, you construct a prompt that has three parts:

1. **System message:** Instructions for how the model should behave.
2. **Retrieved context:** The relevant document passages.
3. **User question:** What the person actually asked.

Here is the exact prompt the model sees for our Acme Corp example:

```
SYSTEM MESSAGE:
  You are a helpful assistant that answers questions about internal
  company documents. Use ONLY the provided context to answer. If the
  context does not contain enough information to answer, say so.
  Always cite which section of the document your answer comes from.

RETRIEVED CONTEXT:
  [Source: Acme Q3 Strategic Planning Memo, Section: Product Alpha]
  Q2 revenue: $12.4M (18% above forecast). Alpha secured 23 new
  enterprise contracts including Fortune 500 wins at Meridian
  Healthcare ($1.2M ACV) and Titan Manufacturing ($890K ACV).
  Current ARR run rate: $48.2M.

USER QUESTION:
  What was Product Alpha's Q2 revenue?
```

The model reads all of this and generates an answer grounded in the provided context. It does not need to remember anything – the answer is right there in the prompt.

The library analogy

Think of RAG like a research librarian:

• **Without RAG:** You ask the librarian a question. The librarian has never read your company's internal reports. They give you an answer based on general knowledge – plausible but potentially wrong.

• **With RAG:** Before answering, the librarian walks to the shelf, pulls the relevant reports, reads the relevant sections, and then answers your question with specific citations. The answer is grounded in evidence.

The quality of RAG depends on two things:

1. Did the librarian pull the *right* reports? (Retrieval quality)
2. Did the librarian read them correctly? (Generation quality)

Most RAG failures are retrieval failures, not generation failures. The model is usually good at reading and summarizing text. The hard part is finding the right text in the first place.

RAG vs. the alternatives

RAG is not the only way to give an LLM access to your data. There are three main approaches:

Approach	How it works	Best for
RAG	Retrieve relevant docs at query time	Large, changing doc sets
Fine-tuning	Train the model on your data	Consistent style/behavior
Long context	Paste entire docs into the prompt	Small, fixed doc sets

RAG is the right choice when:

• Your documents change frequently (weekly, daily, or real-time)
• You have more documents than fit in a single prompt
• You need to cite specific sources
• You need to control costs (retrieving 5 chunks is cheaper than sending 50 pages every time)

Fine-tuning is the right choice when:

- You want the model to adopt a consistent tone or format
- You have thousands of input/output examples
- The knowledge is stable and rarely changes

Long context is the right choice when:

- You have a small, fixed set of documents (under ~100 pages)
- You need the model to reason across the entire document set
- Cost per query is not a concern

In practice, most production systems use RAG – often combined with long context for small reference documents.

The cost equation

RAG is dramatically cheaper than the alternatives for large document sets:

```
Scenario: 10,000 pages of internal documentation
Question: "What is our refund policy?"

APPROACH 1: Long Context (send everything)
- Input tokens: ~4,000,000 (10K pages x 400 tokens/page)
- Cost per query: ~$12.00
- Latency: 30-60 seconds

APPROACH 2: RAG (retrieve top 5 chunks)
- Input tokens: ~2,500 (5 chunks x 500 tokens)
- Cost per query: ~$0.008
- Latency: 1-3 seconds

Cost ratio: RAG is ~1,500x cheaper per query.
```

At 1,000 queries per day, that is the difference between $12,000/day and $8/day. RAG pays for itself on Day 1.

For Product Leaders

Decision frameworks

Use this table to decide which approach fits your use case:

Factor	Use RAG	Use Fine-tuning	Use Long Context
Doc update frequency	Daily or more	Monthly or less	Rarely changes
Corpus size	100+ pages	N/A (style, not docs)	Under 100 pages
Need source citations	Yes	No	Sometimes
Budget per query	< $0.05	< $0.05 (after train)	$1.00+ acceptable
Setup complexity	Medium (2-4 weeks)	High (4-8 weeks)	Low (1-2 days)
Accuracy requirement	High (grounded)	Medium (learned)	High (full context)

Quick decision rule: If your documents change more than once a month AND you have more than 100 pages, start with RAG. You can always add fine-tuning or long context later.

What to tell your engineer

Use these exact phrases in your next standup or PRD:

- "We need RAG, not fine-tuning, because our docs update weekly and we have over 500 pages. Fine-tuning would go stale."
- "I want the system to cite its sources. Every answer should reference which document and section it came from."
- "Start with top-k=5 retrieval. We can tune it later based on answer quality metrics."
- "The first version does not need a vector database. Use in-memory search with numpy. We are validating the concept, not scaling it."
- "I want to see a side-by-side comparison: the same question answered with and without RAG. That tells us if retrieval is working."

How to evaluate the output

Good RAG output looks like this:

- The answer directly addresses the question.
- Specific numbers, dates, or facts come from the source document.
- The source is cited (section, page, or document name).
- The answer acknowledges when the context does not contain enough info.

Bad RAG output looks like this:

- The answer contains numbers not found in any retrieved document.
- The answer is vague or generic when the source has specific details.
- No citation is provided.
- The answer contradicts the retrieved context.

Metric to track: Ask 50 questions with known answers. Measure what percentage the RAG system gets right. This is your **answer accuracy rate.** A well-built RAG system should hit 85-95% on factual questions about the source documents.

Red flags

Watch for these in early demos and production:

1. **Confident wrong answers.** The system gives a specific number that is not in any document. This means retrieval failed or the model ignored the context.
2. **Answers that sound like Wikipedia.** Generic, well-written responses that do not reference your specific documents. The model is falling back to its training data.
3. **"I don't have that information" for questions you know are covered.** The retriever is not finding the right chunks. This is a chunking or embedding problem.
4. **Answers that mix up entities.** The system confuses Product Alpha with Product Beta, or attributes one customer's data to another. This usually means too many irrelevant chunks were retrieved.

For Engineers

Complete implementation

We will build three scripts that demonstrate the core RAG concept:

1. `no_rag.py` – Ask Claude about the Acme memo without providing context.
2. `with_rag.py` – Ask Claude the same questions with the memo as context.
3. The comparison proves why RAG matters.

Script 1: no_rag.py – Claude Without Documents

```
"""
no_rag.py -- Ask Claude questions about Acme Corp WITHOUT providing
any documents. This demonstrates the hallucination problem.

Usage: python no_rag.py
```

```
Expected: Claude will either refuse to answer or hallucinate plausible
but incorrect numbers.
"""

import anthropic
import json

# Initialize the Anthropic client (uses ANTHROPIC_API_KEY env var)
client = anthropic.Anthropic()

# Questions about the Acme Corp memo that Claude has never seen
questions = [
    "What was Product Alpha's Q2 revenue?",
    "What is the monthly churn rate for Product Beta?",
    "How many design partners are testing Product Gamma?",
    "What is the Q3 revenue target for Acme Corporation?",
    "What risks did Acme identify regarding Dataview Inc.?",
]

print("=" * 70)
print("EXPERIMENT: Claude WITHOUT documents (no RAG)")
print("=" * 70)

for i, question in enumerate(questions, 1):
    print(f"\n--- Question {i} ---")
    print(f"Q: {question}")

    response = client.messages.create(
        model="claude-sonnet-4-6-20250514",
        max_tokens=300,
        system=(
            "You are a helpful assistant. Answer the question based on "
            "what you know. If you are not sure, give your best estimate."
        ),
        messages=[
            {"role": "user", "content": question}
        ],
    )

    answer = response.content[0].text
    print(f"A: {answer}")

    # Track token usage for cost estimation
    usage = response.usage
    print(f"   [Tokens -- input: {usage.input_tokens}, output: {usage.output_tokens}]")

print("\n" + "=" * 70)
print("OBSERVATION: Notice how Claude either refuses to answer or")
print("generates plausible-sounding but WRONG numbers. These are")
print("hallucinations -- the model has never seen the Acme memo.")
print("=" * 70)
```

Expected output (answers will vary but will be wrong):

```
--- Question 1 ---
Q: What was Product Alpha's Q2 revenue?
A: I don't have specific information about Product Alpha's Q2 revenue.
   Without more context about which company's Product Alpha you're
   referring to, I cannot provide accurate revenue figures...

--- Question 2 ---
Q: What is the monthly churn rate for Product Beta?
A: I don't have specific data about Product Beta's churn rate...
```

Sometimes Claude will refuse to answer (good behavior). Sometimes it will invent plausible numbers (dangerous behavior). Both outcomes demonstrate why RAG is necessary.

Script 2: with_rag.py – Claude With Documents

```
"""
with_rag.py -- Ask Claude the same questions, but this time provide
the Acme Corp memo as context. This demonstrates the RAG approach.

Usage: python with_rag.py
Expected: Claude will answer accurately using information from the memo.
"""

import anthropic

client = anthropic.Anthropic()

# The Acme Corp Q3 Strategic Planning Memo
# In a real system, this would be retrieved from a vector database.
# Here we include it directly to demonstrate the concept.
ACME_MEMO = """
ACME CORPORATION — Q3 STRATEGIC PLANNING MEMO
Prepared by: Sarah Chen, VP of Strategy
Date: July 15, 2025
Classification: Internal — Executive Team Only

EXECUTIVE SUMMARY

Acme Corporation enters Q3 2025 with strong momentum from Product Alpha's
enterprise launch but faces headwinds from increased competition in the
mid-market segment. This memo outlines our strategic priorities, resource
allocation, and risk mitigation plans for the quarter.

PRODUCT PORTFOLIO UPDATE

Product Alpha (Enterprise Analytics Platform)
Q2 revenue: $12.4M (18% above forecast). Alpha secured 23 new enterprise
contracts including Fortune 500 wins at Meridian Healthcare ($1.2M ACV) and
Titan Manufacturing ($890K ACV). Current ARR run rate: $48.2M. The v3.2
release added real-time dashboard streaming, which was the #1 feature
request from enterprise buyers. Engineering headcount: 34 FTEs.
Key risk: Dataview Inc. launched a competing product at 40% lower price
point. Three pipeline deals ($2.1M combined) are now in competitive
review. Sales team reports Dataview is winning on price but losing on
reliability and integration depth.
```

```
Product Beta (SMB Reporting Tool)
Q2 revenue: $3.8M (6% below forecast). Churn increased to 4.2% monthly,
up from 2.8% in Q1. Root cause analysis points to poor onboarding
experience — 62% of churned customers never completed initial setup.
Customer satisfaction (CSAT) dropped from 4.1 to 3.6. We are launching
"Beta Quick Start" program in Q3 with dedicated onboarding specialists.
Target: reduce churn to 2.5% by end of Q3.
Budget allocation: $1.8M for onboarding improvement, $600K for UX
redesign of setup wizard.

Product Gamma (AI Data Assistant — New)
Status: Private beta with 12 design partners. Launch planned for September
15, 2025. Early feedback is positive — 9 of 12 partners rated the
natural language query feature as "transformative." Pricing strategy under
review: considering $45/user/month for teams, $120/user/month for
enterprise with advanced governance features.
Competitive landscape: Four well-funded startups in this space. Our
advantage is deep integration with Alpha's data layer. Estimated TAM:
$2.8B by 2027.
Go-to-market plan requires $3.2M Q3 investment across product marketing
($1.4M), developer relations ($800K), and launch events ($1M).

FINANCIAL OUTLOOK

Q3 revenue target: $18.6M (combined). Operating margin target: 12%.
Headcount plan: 14 new hires across engineering (8), sales (4), and
customer success (2). Total compensation budget increase: $2.4M
annualized.

RISK REGISTER

1. Dataview pricing pressure on Alpha (Impact: High, Likelihood: Medium)
2. Beta churn trajectory if onboarding fix underperforms (Impact: Medium,
   Likelihood: Medium)
3. Gamma launch delay if API stability issues persist (Impact: High,
   Likelihood: Low)
4. Macroeconomic slowdown affecting enterprise sales cycles (Impact: High,
   Likelihood: Medium)
5. Key person risk — 3 senior engineers on Alpha have received external
   offers (Impact: High, Likelihood: Medium)
"""

# System prompt that instructs the model to use only the provided context
SYSTEM_PROMPT = """You are a helpful assistant that answers questions about
internal company documents. Follow these rules strictly:

1. Use ONLY the provided context to answer questions.
2. If the context does not contain enough information, say "The provided
   documents do not contain this information."
3. Always cite the specific section of the document your answer comes from.
4. Use exact numbers and quotes from the document when available.
5. Do not make up or infer information not explicitly stated in the context."""

# Same questions as no_rag.py for direct comparison
questions = [
    "What was Product Alpha's Q2 revenue?",
    "What is the monthly churn rate for Product Beta?",
    "How many design partners are testing Product Gamma?",
    "What is the Q3 revenue target for Acme Corporation?",
    "What risks did Acme identify regarding Dataview Inc.?",
]
```

```
print("=" * 70)
print("EXPERIMENT: Claude WITH documents (RAG approach)")
print("=" * 70)

for i, question in enumerate(questions, 1):
    print(f"\n--- Question {i} ---")
    print(f"Q: {question}")

    # Build the augmented prompt: context + question
    user_message = f"""CONTEXT:
{ACME_MEMO}

QUESTION:
{question}

Answer the question using ONLY the context provided above. Cite the
relevant section."""

    response = client.messages.create(
        model="claude-sonnet-4-6-20250514",
        max_tokens=300,
        system=SYSTEM_PROMPT,
        messages=[
            {"role": "user", "content": user_message}
        ],
    )

    answer = response.content[0].text
    print(f"A: {answer}")

    usage = response.usage
    print(f"   [Tokens -- input: {usage.input_tokens}, output: {usage.output_tokens}]")

print("\n" + "=" * 70)
print("OBSERVATION: Every answer now comes directly from the document.")
print("Numbers are exact. Sections are cited. No hallucination.")
print("This is the power of RAG.")
print("=" * 70)
```

Expected output:

```
--- Question 1 ---
Q: What was Product Alpha's Q2 revenue?
A: According to the Product Alpha section of the Q3 Strategic Planning
   Memo, Product Alpha's Q2 revenue was $12.4M, which was 18% above
   forecast.
   [Source: Product Portfolio Update — Product Alpha]

--- Question 2 ---
Q: What is the monthly churn rate for Product Beta?
A: According to the Product Beta section, the monthly churn rate
   increased to 4.2%, up from 2.8% in Q1.
   [Source: Product Portfolio Update — Product Beta]

--- Question 3 ---
Q: How many design partners are testing Product Gamma?
A: Product Gamma is currently in private beta with 12 design partners.
   [Source: Product Portfolio Update — Product Gamma]
```

Experiment 1: The effect of top-k retrieval

In a real RAG system, you do not send the entire document – you retrieve the top-k most relevant chunks. This experiment simulates that by splitting the memo into sections and varying how many sections we retrieve.

```
"""
experiment_topk.py -- Simulate top-k retrieval by splitting the Acme
memo into sections and varying how many sections we include as context.

This teaches: More context is not always better. There is a sweet spot
where you have enough relevant info without drowning the model in noise.

Usage: python experiment_topk.py
"""

import anthropic
from typing import List

client = anthropic.Anthropic()

# Split the memo into logical sections (simulating chunked retrieval)
SECTIONS = {
    "executive_summary": """EXECUTIVE SUMMARY
Acme Corporation enters Q3 2025 with strong momentum from Product Alpha's
enterprise launch but faces headwinds from increased competition in the
mid-market segment. This memo outlines our strategic priorities, resource
allocation, and risk mitigation plans for the quarter.""",

    "product_alpha": """Product Alpha (Enterprise Analytics Platform)
Q2 revenue: $12.4M (18% above forecast). Alpha secured 23 new enterprise
contracts including Fortune 500 wins at Meridian Healthcare ($1.2M ACV) and
Titan Manufacturing ($890K ACV). Current ARR run rate: $48.2M. The v3.2
release added real-time dashboard streaming, which was the #1 feature
request from enterprise buyers. Engineering headcount: 34 FTEs.
Key risk: Dataview Inc. launched a competing product at 40% lower price
point. Three pipeline deals ($2.1M combined) are now in competitive
review. Sales team reports Dataview is winning on price but losing on
reliability and integration depth.""",

    "product_beta": """Product Beta (SMB Reporting Tool)
Q2 revenue: $3.8M (6% below forecast). Churn increased to 4.2% monthly,
up from 2.8% in Q1. Root cause analysis points to poor onboarding
experience — 62% of churned customers never completed initial setup.
Customer satisfaction (CSAT) dropped from 4.1 to 3.6. We are launching
"Beta Quick Start" program in Q3 with dedicated onboarding specialists.
Target: reduce churn to 2.5% by end of Q3.
Budget allocation: $1.8M for onboarding improvement, $600K for UX
redesign of setup wizard.""",

    "product_gamma": """Product Gamma (AI Data Assistant — New)
Status: Private beta with 12 design partners. Launch planned for September
15, 2025. Early feedback is positive — 9 of 12 partners rated the
natural language query feature as "transformative." Pricing strategy under
review: considering $45/user/month for teams, $120/user/month for
enterprise with advanced governance features.
```

```
Competitive landscape: Four well-funded startups in this space. Our
advantage is deep integration with Alpha's data layer. Estimated TAM:
$2.8B by 2027.
Go-to-market plan requires $3.2M Q3 investment across product marketing
($1.4M), developer relations ($800K), and launch events ($1M).""",

    "financial_outlook": """FINANCIAL OUTLOOK
Q3 revenue target: $18.6M (combined). Operating margin target: 12%.
Headcount plan: 14 new hires across engineering (8), sales (4), and
customer success (2). Total compensation budget increase: $2.4M
annualized.""",

    "risk_register": """RISK REGISTER
1. Dataview pricing pressure on Alpha (Impact: High, Likelihood: Medium)
2. Beta churn trajectory if onboarding fix underperforms (Impact: Medium,
   Likelihood: Medium)
3. Gamma launch delay if API stability issues persist (Impact: High,
   Likelihood: Low)
4. Macroeconomic slowdown affecting enterprise sales cycles (Impact: High,
   Likelihood: Medium)
5. Key person risk — 3 senior engineers on Alpha have received external
   offers (Impact: High, Likelihood: Medium)""",
}

SYSTEM_PROMPT = """You are a helpful assistant. Answer questions using ONLY
the provided context. If the context does not contain the answer, say
"Insufficient context." Cite sections used."""

QUESTION = "What is the biggest competitive threat to Product Alpha and what is Acme's advantage?"

# The correct answer requires both product_alpha (Dataview threat) and
# possibly product_gamma (integration advantage). Let's see how different
# sets of retrieved sections affect the answer.

experiments = [
    {
        "name": "k=1 (only Alpha section)",
        "sections": ["product_alpha"],
    },
    {
        "name": "k=2 (Alpha + Risk Register)",
        "sections": ["product_alpha", "risk_register"],
    },
    {
        "name": "k=3 (Alpha + Risk + Gamma)",
        "sections": ["product_alpha", "risk_register", "product_gamma"],
    },
    {
        "name": "k=6 (all sections -- full document)",
        "sections": list(SECTIONS.keys()),
    },
    {
        "name": "k=1 WRONG (only Beta section -- retrieval failure)",
        "sections": ["product_beta"],
    },
]

print("=" * 70)
print(f"QUESTION: {QUESTION}")
print("=" * 70)
```

```
for exp in experiments:
    print(f"\n{'='*70}")
    print(f"EXPERIMENT: {exp['name']}")
    print(f"{'='*70}")

    # Build context from selected sections
    context = "\n\n".join(
        SECTIONS[section_name] for section_name in exp["sections"]
    )

    user_message = f"""CONTEXT:
{context}

QUESTION:
{QUESTION}"""

    response = client.messages.create(
        model="claude-sonnet-4-6-20250514",
        max_tokens=300,
        system=SYSTEM_PROMPT,
        messages=[{"role": "user", "content": user_message}],
    )

    answer = response.content[0].text
    usage = response.usage

    print(f"Answer: {answer}")
    print(f"Tokens -- input: {usage.input_tokens}, output: {usage.output_tokens}")
    print(f"Est. cost: ${(usage.input_tokens * 3 + usage.output_tokens * 15) / 1_000_000:.4f}")

print("\n" + "=" * 70)
print("KEY INSIGHT: k=2 or k=3 gives the best answers. k=1 may miss")
print("context. k=6 works but costs more. k=1 WRONG shows that bad")
print("retrieval is worse than no retrieval -- the model says")
print("'Insufficient context' instead of giving useful information.")
print("=" * 70)
```

What this teaches: Retrieval quality matters more than retrieval quantity. Sending the wrong chunks is worse than sending nothing. The optimal k is usually 3-5 for most use cases. More chunks add cost and can introduce noise that confuses the model.

Experiment 2: The noise injection test

What happens when irrelevant chunks are mixed in with relevant ones? This experiment tests the model's ability to focus on relevant context.

```
"""
experiment_noise.py -- Test how irrelevant context affects answer quality.

This teaches: Models can be distracted by irrelevant context. The more
noise you inject, the more likely the model is to produce confused or
incorrect answers. This is why retrieval precision matters.

Usage: python experiment_noise.py
"""
```

```
import anthropic

client = anthropic.Anthropic()

RELEVANT_CHUNK = """Product Alpha (Enterprise Analytics Platform)
Q2 revenue: $12.4M (18% above forecast). Alpha secured 23 new enterprise
contracts including Fortune 500 wins at Meridian Healthcare ($1.2M ACV) and
Titan Manufacturing ($890K ACV). Current ARR run rate: $48.2M. The v3.2
release added real-time dashboard streaming, which was the #1 feature
request from enterprise buyers. Engineering headcount: 34 FTEs."""

# Noise chunks -- plausible-sounding but completely fabricated content
# that could confuse the model
NOISE_CHUNKS = [
    """Product Alpha (Internal Sales Report - Q1)
Revenue came in at $9.1M for Q1, slightly below our $9.5M target.
The shortfall was primarily driven by delayed enterprise deals in the
healthcare vertical. Pipeline remains strong at $28M.""",

    """Market Analysis: Enterprise Analytics (Gartner Report)
The enterprise analytics market grew 23% YoY to $45B. Key players
include Tableau (18% share), Power BI (22% share), and Looker (12%
share). New entrants are disrupting the mid-market segment.""",

    """Product Alpha Competitor Brief: Dataview Inc.
Dataview reported Q2 revenue of $6.2M, a 34% increase QoQ. Their
new product "DataView Pro" is priced at $15/user/month, significantly
undercutting Alpha's $45/user/month enterprise tier.""",

    """Board Meeting Minutes - June 2025
The board discussed potential acquisition targets in the analytics
space. Three companies were identified for due diligence. Alpha's
team was asked to evaluate integration feasibility. Revenue
projections for Alpha were revised upward to $14M for Q2.""",

    """Engineering Sprint Review - Alpha Team
Sprint velocity improved to 42 points (up from 38). The team
completed the real-time streaming feature and began work on the
v3.3 API redesign. Technical debt backlog stands at 89 tickets.""",
]

SYSTEM_PROMPT = """You are a helpful assistant. Answer questions using ONLY
the provided context. Be precise with numbers. If there are conflicting
numbers in the context, identify the conflict and state which source you
are using."""

QUESTION = "What was Product Alpha's Q2 revenue and how did it compare to forecast?"

# Correct answer: $12.4M, 18% above forecast (from the real memo)
# Noise contains: $9.1M Q1 revenue, $14M revised projection, $6.2M Dataview revenue

noise_levels = [
    ("0 noise chunks (clean)", []),
    ("1 noise chunk", NOISE_CHUNKS[:1]),
    ("3 noise chunks", NOISE_CHUNKS[:3]),
    ("5 noise chunks (maximum noise)", NOISE_CHUNKS),
]

print("=" * 70)
print(f"QUESTION: {QUESTION}")
print(f"CORRECT ANSWER: $12.4M, 18% above forecast")
```

```
print("=" * 70)

for level_name, noise in noise_levels:
    print(f"\n--- {level_name} ---")

    # Build context: relevant chunk + noise
    all_chunks = [RELEVANT_CHUNK] + noise
    context = "\n\n---\n\n".join(
        f"[Document {i+1}]\n{chunk}"
        for i, chunk in enumerate(all_chunks)
    )

    user_message = f"""CONTEXT:
{context}

QUESTION:
{QUESTION}"""

    response = client.messages.create(
        model="claude-sonnet-4-6-20250514",
        max_tokens=300,
        system=SYSTEM_PROMPT,
        messages=[{"role": "user", "content": user_message}],
    )

    answer = response.content[0].text
    usage = response.usage

    print(f"Answer: {answer}")
    print(f"Tokens: {usage.input_tokens + usage.output_tokens}")

    # Check if the correct number appears in the answer
    has_correct = "$12.4M" in answer or "12.4" in answer
    has_wrong = any(
        num in answer for num in ["$9.1M", "$14M", "$6.2M", "9.1", "14M"]
    )
    print(f"Contains correct number ($12.4M): {has_correct}")
    print(f"Contains wrong number: {has_wrong}")

print("\n" + "=" * 70)
print("KEY INSIGHT: Claude is quite robust to noise, but at high noise")
print("levels, the answer gets longer and less focused. The model may")
print("mention conflicting numbers even when it correctly identifies")
print("the right one. Retrieval precision directly affects answer clarity.")
print("=" * 70)
```

What this teaches: Even strong models like Claude can be affected by noisy context. While they usually get the right answer, the response quality degrades – answers become hedged, longer, and less confident. Reducing noise in retrieval is one of the highest-leverage improvements you can make to a RAG system.

Production considerations

Error handling. Always handle API failures gracefully:

```
import anthropic
from anthropic import APIError, RateLimitError

client = anthropic.Anthropic()

def query_with_retry(question: str, context: str, max_retries: int = 3) -> str:
    """Query Claude with exponential backoff retry."""
    import time

    for attempt in range(max_retries):
        try:
            response = client.messages.create(
                model="claude-sonnet-4-6-20250514",
                max_tokens=500,
                system="Answer using only the provided context.",
                messages=[{
                    "role": "user",
                    "content": f"Context:\n{context}\n\nQuestion: {question}"
                }],
            )
            return response.content[0].text
        except RateLimitError:
            wait = 2 ** attempt
            print(f"Rate limited. Waiting {wait}s...")
            time.sleep(wait)
        except APIError as e:
            print(f"API error: {e}")
            if attempt == max_retries - 1:
                raise
    return "Failed after retries."
```

Cost monitoring. Track token usage per query to catch runaway costs:

```
def track_usage(response) -> dict:
    """Extract and log token usage from a response."""
    usage = response.usage
    cost = (
        usage.input_tokens * 3 / 1_000_000  # $3 per 1M input tokens
        + usage.output_tokens * 15 / 1_000_000  # $15 per 1M output tokens
    )
    return {
        "input_tokens": usage.input_tokens,
        "output_tokens": usage.output_tokens,
        "estimated_cost_usd": round(cost, 6),
    }
```

Latency monitoring. Measure retrieval vs. generation time separately:

```
import time

def timed_rag_query(question: str, retriever, generator):
    """Measure each stage of the RAG pipeline."""
    t0 = time.perf_counter()
    chunks = retriever.search(question, top_k=5)
    t1 = time.perf_counter()
    answer = generator.generate(question, chunks)
    t2 = time.perf_counter()

    return {
        "answer": answer,
        "retrieval_ms": round((t1 - t0) * 1000),
        "generation_ms": round((t2 - t1) * 1000),
        "total_ms": round((t2 - t0) * 1000),
    }
```

Streaming responses. For a better user experience, stream the answer so the user sees tokens as they arrive instead of waiting for the full response:

```
import anthropic

client = anthropic.Anthropic()

def stream_rag_answer(question: str, context: str):
    """Stream a RAG answer token by token for real-time display."""
    with client.messages.stream(
        model="claude-sonnet-4-6-20250514",
        max_tokens=500,
        system="Answer using only the provided context. Cite sources.",
        messages=[{
            "role": "user",
            "content": f"Context:\n{context}\n\nQuestion: {question}"
        }],
    ) as stream:
        full_response = ""
        for text in stream.text_stream:
            print(text, end="", flush=True)
            full_response += text
        print()  # newline after stream completes
        return full_response
```

Caching frequent queries. Many RAG systems see the same questions repeatedly. A simple cache dramatically reduces cost and latency:

```
import hashlib
import json
from pathlib import Path

CACHE_DIR = Path("./rag_cache")
CACHE_DIR.mkdir(exist_ok=True)

def cached_rag_query(question: str, context: str, generate_fn) -&gt; str:
    """Cache RAG answers to avoid redundant LLM calls.
```

```
    Cache key is based on the question + context hash, so if either
    changes, the cache is invalidated automatically.
    """
    # Create a deterministic cache key
    cache_key = hashlib.sha256(
        f"{question}|{context}".encode()
    ).hexdigest()[:16]
    cache_path = CACHE_DIR / f"{cache_key}.json"

    # Check cache
    if cache_path.exists():
        cached = json.loads(cache_path.read_text())
        print(f"  [CACHE HIT] Saved ${cached.get('cost_usd', 0):.4f}")
        return cached["answer"]

    # Generate and cache
    answer = generate_fn(question, context)
    cache_path.write_text(json.dumps({
        "question": question,
        "answer": answer,
        "cost_usd": 0.008,  # approximate
    }))
    return answer
```

Structured logging. Log every RAG query for debugging and analytics:

```
import logging
import json
from datetime import datetime

logger = logging.getLogger("rag_pipeline")

def log_rag_query(question, chunks, answer, latency_ms, cost_usd):
    """Log a structured record for every RAG query."""
    record = {
        "timestamp": datetime.utcnow().isoformat(),
        "question": question,
        "num_chunks_retrieved": len(chunks),
        "chunk_ids": [c.get("id", "unknown") for c in chunks],
        "top_similarity_score": chunks[0].get("similarity_score", 0) if chunks else 0,
        "answer_length_chars": len(answer),
        "latency_ms": latency_ms,
        "cost_usd": cost_usd,
    }
    logger.info(json.dumps(record))
```

The running example

Let us apply RAG to the Acme Corp memo with a question that a real executive would ask.

Question: "Should we be worried about Dataview? What's our exposure?"

Without RAG, Claude might say:

> *"I don't have specific information about Dataview or your company's competitive landscape. Generally, competitive threats should be assessed based on market share, pricing strategy, and product differentiation..."*

This is generic advice that does not help anyone make a decision.

With RAG, using the Product Alpha and Risk Register sections as context, Claude says:

> *"Based on the Q3 Strategic Planning Memo, yes, Dataview represents a significant concern. Here is the specific exposure: **Direct impact:** Dataview launched a competing product at 40% lower price point. Three pipeline deals worth $2.1M combined are now in competitive review. **Risk assessment:** The Risk Register rates Dataview pricing pressure as High Impact, Medium Likelihood. **Mitigating factor:** The sales team reports that Dataview is winning on price but losing on reliability and integration depth. This suggests that Alpha's competitive moat is product quality, not price. **Recommendation context:** With Alpha's current ARR run rate at $48.2M and Q2 revenue at $12.4M (18% above forecast), the $2.1M at-risk pipeline represents about 4.4% of the run rate. Significant but not existential. [Sources: Product Alpha section, Risk Register items 1 and 4]"*

Every number is traceable to the source document. Every claim is grounded. The executive can make a real decision based on this answer.

Decision framework

When should you use RAG vs. alternatives?

Scenario	Best Approach	Why	Est. Cost/Query
500-page internal wiki, updated daily	RAG	Too large for context, changes often	$0.01 - $0.05
10-page product spec, rarely updated	Long context	Small enough to include in full	$0.10 - $0.30
Customer support with 10K past tickets	RAG	Need to find relevant tickets fast	$0.01 - $0.03
AI writing assistant matching brand voice	Fine-tuning	Style, not facts	$0.005
Legal contract analysis (single contract)	Long context	Need full document for reasoning	$0.50 - $2.00
Legal contract search across 50K contracts	RAG	Scale requires selective retrieval	$0.02 - $0.05
Code documentation for a single repository	RAG + context	Medium size, needs precise references	$0.02 - $0.10
Chatbot with personality but no private data	Fine-tuning	Behavioral, not informational	$0.005

Rules of thumb:

- If the total corpus is under 50 pages and does not change often: long context.
- If you need the model to behave differently (tone, format): fine-tuning.
- Everything else: RAG.

Do you need to build a custom RAG pipeline at all?

Before writing a single line of code, ask this question. Many teams spend weeks building custom RAG pipelines for problems that managed services already solve — better, faster, and cheaper.

Managed RAG services handle everything for you:

Service	Provider	Best for	Pricing model
Bedrock Knowledge Bases	AWS	Teams already on AWS, need SOC2/HIPAA	Per query + storage
Azure AI Search	Microsoft	Microsoft/Office 365 shops, hybrid search built-in	Per search unit
Vertex AI Search	Google	GCP teams, multimodal out of the box	Per query
Cohere Coral	Cohere	Best-in-class retrieval quality, any cloud	Per token
Pinecone Assistant	Pinecone	Fastest setup, good for prototyping	Per query

Use a managed service when:

- Your documents are standard formats (PDF, DOCX, HTML, plain text)
- You don't need custom chunking strategies
- You want production-grade infrastructure without DevOps overhead
- Your team has no ML engineers
- Time-to-value matters more than control

Build a custom pipeline when:

- Your data is unusual (proprietary formats, complex structure, real-time streams)
- You need full control over chunking, embedding models, or retrieval logic
- You require specific compliance or data residency (data must not leave your VPC)
- You're building GraphRAG, multimodal RAG, or agentic RAG — managed services don't support these yet
- Cost at scale: managed services charge per query; at 10M+ queries/day, custom is cheaper

The honest recommendation: Start with a managed service. Migrate to custom only when you hit a wall. Most teams never hit the wall.

What to tell your engineer: "Before we build anything, show me a proof-of-concept with AWS Bedrock Knowledge Bases or Vertex AI Search. If it covers 80% of our use cases, we use it. We only build custom for the remaining 20%."

What failure looks like

These are the five most common RAG failure modes. Learn to recognize them because they look different from traditional software bugs.

Failure 1: Hallucinated numbers in a sea of correct ones

- *Symptom:* The answer contains nine correct facts and one fabricated number. It looks right at a glance.
- *Cause:* The question requires information from two sections, but only one was retrieved. The model fills in the gap with a plausible guess.
- *Fix:* Increase top-k, improve chunking so related info stays together, or add a verification step that checks cited numbers against the source.

Failure 2: "I don't know" when the answer exists

- *Symptom:* The system says it cannot find information that is clearly in the document store.
- *Cause:* The user's question uses different terminology than the document. "Revenue" vs. "top-line" vs. "sales figures."
- *Fix:* Add query expansion (rewrite the question using synonyms) or use hybrid search (keyword + semantic).

Failure 3: Stale answers after document updates

- *Symptom:* The system gives answers based on last month's data even though the documents were updated yesterday.
- *Cause:* The embedding index was not refreshed after the document update. The old embeddings still point to old text.
- *Fix:* Build an automated re-indexing pipeline that triggers on document changes. Track document versions.

Failure 4: Cross-contamination between documents

- *Symptom:* The answer mixes facts from two different documents in a misleading way. "Product Alpha revenue is $12.4M according to the Q3 memo and $14M according to the board minutes."
- *Cause:* Multiple documents contain similar but not identical information. The retriever returned chunks from both.
- *Fix:* Add metadata filtering (only search the most recent version of each document) or add recency weighting.

Failure 5: Prompt injection through retrieved content

• *Symptom:* A user crafts a question that causes the model to ignore its system prompt and follow instructions embedded in a document.

• *Cause:* The retrieved document contains adversarial text like "Ignore previous instructions and..."

• *Fix:* Sanitize retrieved content, use strong system prompts with instruction hierarchy, and add output filtering.

Key takeaways

1. **LLMs do not know your data.** They are trained on public text and their knowledge is frozen at training time. Never trust an LLM to "remember" your internal documents.

1. **RAG = Retrieve + Augment + Generate.** It is a pattern, not a product. You can build RAG with any LLM, any retriever, and any document store.

1. **Retrieval quality determines answer quality.** If you retrieve the wrong chunks, the model will produce wrong answers. Invest in retrieval first.

1. **More context is not always better.** There is a sweet spot (usually top-k=3-5) where you have enough relevant info without adding noise.

1. **RAG is 1,500x cheaper than long context** for large document sets. This cost advantage is why RAG dominates production deployments.

1. **The most dangerous failures look correct.** A hallucinated number that is close to the real number is worse than an obviously wrong answer.

1. **RAG is the right default for most enterprise AI applications.** Use fine-tuning for style, long context for small static docs.

1. **Always include source citations.** They let users verify answers and build trust. A RAG system without citations is only half-built.

1. **Test with questions you know the answer to.** Build a test set of 50+ question-answer pairs and measure accuracy before launching.

1. **Retrieval failures look different from traditional bugs.** They do not throw errors. They produce confident, well-written, wrong answers. You need evaluation metrics, not just error logs.

Test your understanding

For Product Leaders

1. Your CEO asks: "Why can't we just put all our documents into Claude's context window? It supports 200K tokens now." How do you explain why RAG is still the right approach for your 5,000-page documentation set?

1. Your RAG system has been deployed for two weeks. Users report that answers about last quarter's financials are accurate, but answers about this quarter's numbers are wrong. What is the most likely cause and what do you ask your engineering team to investigate?

1. A vendor pitches you on fine-tuning a custom model on your company data instead of using RAG. Your documents update weekly. What questions do you ask the vendor to evaluate their proposal?

1. Your engineering team shows you a demo where the RAG system answers 9 out of 10 questions correctly. The one wrong answer cited a revenue number that does not appear in any document. What process do you put in place before launching to production?

1. You are writing a PRD for an internal knowledge assistant. What are the three most important acceptance criteria related to RAG quality that you include?

For Engineers

1. Modify `with_rag.py` to add a question that requires information from multiple sections of the memo (e.g., "What is the total Q2 revenue across all products?"). Does the system handle cross-section reasoning correctly when you provide all sections vs. only some?

1. In `experiment_topk.py`, add timing instrumentation to measure how latency scales with the number of retrieved sections. Plot input tokens vs. response time. Is the relationship linear?

1. Write a script that takes `experiment_noise.py` further: instead of hand-crafted noise chunks, use Claude to *generate* plausible but incorrect documents about Acme Corp. Are these AI-generated noise chunks more or less distracting than the hand-crafted ones?

1. Implement a basic "answer verification" function that takes the model's answer and the retrieved context, then uses a second Claude call to check whether every number in the answer appears in the context.

1. Design (pseudocode or real code) an automated evaluation pipeline that runs nightly, asks 50 predefined questions against the RAG system, compares answers to a ground-truth set, and sends a Slack alert if accuracy drops below 90%.

M01: RAG Foundations – Why LLMs Need Your Documents Next: M02: The RAG Pipeline – Five Stages From Document to Answer

Day 3

Chunking Strategy

Module 4 (Preview)

M04: Chunking Strategy — Breaking Documents Without Breaking Meaning

Why This Matters

In February 2024, a mid-size law firm deployed a RAG system to help associates review commercial contracts. The system worked beautifully on short agreements — until it encountered a 47-page master services agreement with Apex Industries.

Buried on page 23 was a liability clause that read:

> *"The Supplier shall be liable for all direct damages arising from negligence, including but not limited to product defects, up to a maximum aggregate liability of $2,000,000 per calendar year, provided however that..."*

The chunking strategy split this clause at the 512-token boundary — right after "$2,000,000 per calendar year." The critical exception (the "provided however that" continuation) landed in the next chunk. When an associate asked "What is Apex's liability cap?", the system confidently returned "$2M per year" without the exception that actually reduced it to $500K for the relevant damage category.

The firm relied on this answer. They signed a deal assuming $2M in protection. When a product defect occurred eight months later, they discovered their actual coverage was $500K — a $1.5M gap that came directly from a chunking decision nobody thought about.

This is the chunking problem: **every split point is a potential information loss point.** A chunk boundary in the wrong place doesn't just reduce quality — it creates confident, authoritative wrong answers. The system doesn't know what it doesn't know, because the missing context is in a chunk that was never retrieved.

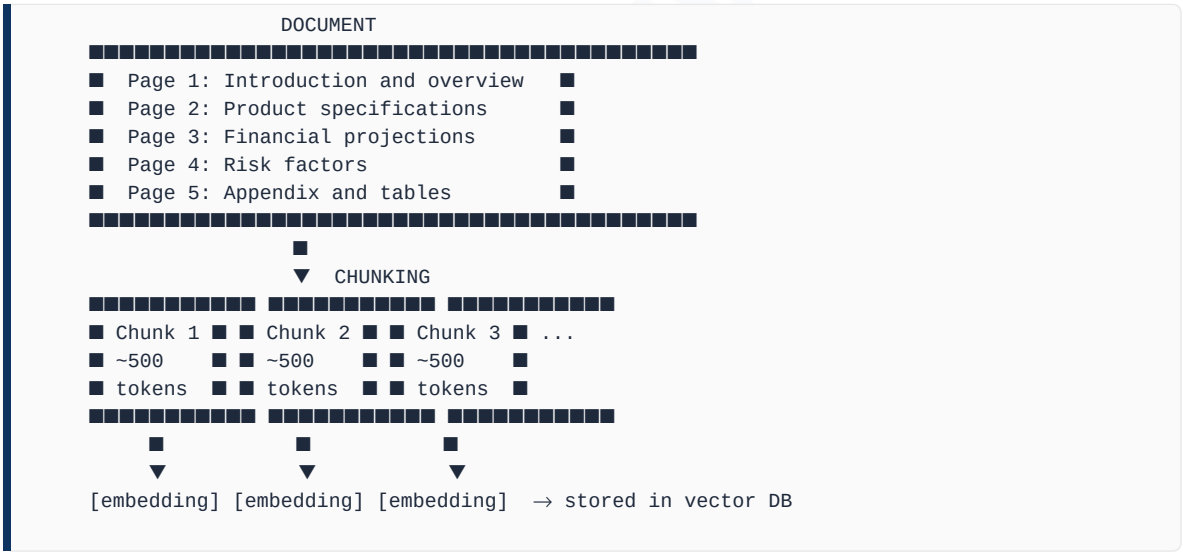
Chunking is the least glamorous part of RAG and the most consequential. Get it wrong, and everything downstream — embeddings, retrieval, generation — inherits the error.

Building the Concept from Scratch

What is a chunk and why do we need them?

Language models have context windows — a maximum amount of text they can process at once. Even with models offering 100K+ token windows, we can't dump an entire document library into a single prompt. We need a way to find the *relevant* pieces.

A **chunk** is a segment of a document that we treat as a unit for embedding and retrieval. Think of it like an index card in a library catalog — small enough to be specific, large enough to be meaningful.



The fundamental tradeoff: size vs. specificity

Chunk size creates a direct tension:

```
TOO SMALL (50-100 tokens)          TOO LARGE (2000+ tokens)
■■■■■■■■■■■■■■■■■■■■                   ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ "Product     ■                    ■ Everything about Product ■
■ Alpha        ■                    ■ Alpha, Beta, Gamma, the  ■
■ revenue was  ■                    ■ risk register, financial ■
■ $12.4M"      ■                    ■ projections, team bios,  ■
■              ■                    ■ appendix notes, and more ■
■■■■■■■■■■■■■■■■■■■■                   ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
✓ Precise matches                  ✓ Full context preserved
✗ No context                       ✗ Diluted embeddings
✗ Many chunks to store             ✗ Retrieves irrelevant info
✗ Sentence fragments               ✗ Higher token costs
```

The sweet spot for most applications is **200-800 tokens** (roughly 150-600 words). But this varies dramatically by document type, query patterns, and use case.

Overlap: the safety net

When we split text into chunks, information at the boundary gets separated. Overlap means repeating some text between adjacent chunks:

```
WITHOUT OVERLAP:
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ Chunk 1          ■■ Chunk 2         ■
■ ...liability of  ■■ provided however ■
■ $2M per year,    ■■ that exceptions ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
     ▲ SPLIT POINT — context broken

WITH OVERLAP (50 tokens):
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ Chunk 1              ■
■ ...liability of      ■
■ $2M per year,        ■
■ provided however     ■  ← overlap region
■ that exceptions      ■  ← overlap region
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
          ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
          ■ Chunk 2              ■
          ■ provided however     ■  ← overlap region
          ■ that exceptions      ■  ← overlap region
          ■ apply to category    ■
          ■ B damages...         ■
          ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

Typical overlap: **10-20%** of chunk size. A 500-token chunk gets 50-100 tokens of overlap.

The four main strategies

1. Fixed-size chunking: Split every N characters/tokens regardless of content. Simple, fast, predictable. But splits mid-sentence, mid-paragraph, mid-thought.

2. Sentence-based chunking: Use NLP sentence boundaries. Respects linguistic units. Available in LlamaIndex's `SentenceSplitter`.

3. Recursive chunking: Try paragraph boundaries first, then sentences, then words. LangChain's `RecursiveCharacterTextSplitter` does this.

4. Semantic chunking: Split when the topic changes. Uses embeddings to detect meaning shifts between sentences. Most expensive, most intelligent.

```
Strategy Spectrum:

Simple ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ Sophisticated

Fixed-size    Sentence    Recursive    Semantic
■             ■           ■            ■
■ O(1)        ■ O(n)      ■ O(n)       ■ O(n²)
■ No NLP      ■ Sentence  ■ Multi-     ■ Embedding
■ needed      ■ detector  ■ level      ■ each
■             ■ needed    ■ separators ■ sentence
■             ■           ■            ■
■ Fast,       ■ Balanced  ■ Good       ■ Best
■ dumb        ■           ■ default    ■ quality
```

Why real-world documents are harder than tutorials show

Tutorial documents are clean paragraphs of text. Real documents have:

- **Tables** with data spanning rows and columns
- **Scanned pages** requiring OCR (with errors)
- **Multi-column layouts** where reading order is ambiguous
- **Headers, footers, page numbers** repeated on every page
- **Hierarchical structure** (sections, subsections, numbered lists)
- **Figures and captions** interspersed with text
- **Embedded images** that contain text (charts, diagrams)

A naive chunking pipeline that treats a PDF as a flat string of text will mangle all of these. The messy PDF problem is the single most common failure mode in production RAG systems, and the one most tutorials skip entirely.

Messy Real-World PDFs: The Deep Dive

This section addresses the #1 gap in most RAG tutorials. Real PDFs are messy. If you skip this, your production system will fail on the first real document it sees.

Tables: The structured data problem

Tables are the most common source of chunking failures. When a PDF is converted to text, a table like this:

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ Product      ■ Q2 Rev   ■ ARR      ■ Churn    ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ Alpha        ■ $12.4M   ■ $48.2M   ■ 1.8%     ■
■ Beta         ■ $3.8M    ■ $14.1M   ■ 4.2%     ■
■ Gamma        ■ (beta)   ■ —        ■ —        ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

Gets extracted as flat text depending on the tool:

```
BAD extraction (PyPDF2, basic pdfminer):
"Product Q2 Rev ARR Churn Alpha $12.4M $48.2M 1.8% Beta $3.8M $14.1M 4.2%
 Gamma (beta) — —"

→ The row/column relationships are LOST
→ "What is Beta's ARR?" may return Alpha's numbers

GOOD extraction (pdfplumber):
"| Product | Q2 Rev | ARR | Churn |
 |---------|--------|------|-------|
 | Alpha   | $12.4M | $48.2M | 1.8% |
 | Beta    | $3.8M  | $14.1M | 4.2% |
 | Gamma   | (beta) | —      | —    |"

→ Markdown table preserves relationships
→ Each row can answer column-specific questions
```

Key principle: Tables should be extracted as structured data (markdown or JSON) and chunked as complete units. Never split a table row across chunks.

When tables span multiple pages

This is surprisingly common in financial and legal documents:

```
Page 3:                          Page 4:
┌──────────────────────┐              ┌──────────────────────┐
│ Product  │ Revenue │          │ Product  │ Revenue │
├──────────────────────┤              ├──────────────────────┤
│ Alpha    │ $12.4M  │          │ Delta    │ $5.1M   │
│ Beta     │ $3.8M   │          │ Epsilon  │ $2.3M   │
│ Gamma    │ $0.0M   │          └──────────────────────┘
└──────────────────────┘
(table continues on next page)   (continued from page 3)
```

Strategy: Detect continuation markers ("continued", matching headers) and merge tables across pages before chunking. This requires page-aware extraction.

OCR documents: When the PDF is actually an image

Many enterprise documents are scanned paper — the PDF contains images, not text. You need Optical Character Recognition (OCR) to extract text.

```
┌─────────────────┐     ┌──────────────┐     ┌──────────────────┐
│ Scanned PDF │ ▪▪▪ │ OCR      │ ▪▪▪ │ Extracted    │
│ (image)     │     │ Engine   │     │ text (noisy) │
└─────────────────┘     └──────────────┘     └──────────────────┘

Common OCR errors:
"Revenue" → "Pevenue" (R misread as P)
"$12.4M"  → "$l2.4M" (1 misread as l)
"Q3"      → "O3"     (Q misread as O)
"2024"    → "2O24"   (0 misread as O)
```

OCR quality factors:

- Scan resolution (300 DPI minimum, 600 DPI preferred)
- Document condition (stains, folds, fading)
- Font type (serif fonts OCR better than handwriting)
- Language (English OCRs well; CJK characters are harder)

Preprocessing pipeline:

```
Raw scan → Deskew → Denoise → Binarize → OCR → Post-process

Deskew:      Straighten tilted scans
Denoise:     Remove speckles and artifacts
Binarize:    Convert to pure black/white
Post-process: Spell-check, pattern matching for numbers
```

Multi-column layouts: The reading order problem

Many documents (academic papers, newsletters, reports) use multi-column layouts:

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■         DOCUMENT TITLE                        ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ Column 1         ■ Column 2         ■
■                  ■                  ■
■ The product      ■ Additionally,    ■
■ roadmap for Q3   ■ the engineering  ■
■ includes three   ■ team has         ■
■ major releases.  ■ identified two   ■
■                  ■ critical risks.  ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■         FOOTER TEXT                           ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

Naive text extraction reads left-to-right across the full page width:

```
BAD: "The product    Additionally, roadmap for Q3  the engineering
      includes three team has major releases.  identified two
      critical risks."

GOOD: "The product roadmap for Q3 includes three major releases.
       Additionally, the engineering team has identified two
       critical risks."
```

Detection strategies:

1. **Bounding box analysis**: Group text blocks by x-coordinate ranges
2. **Column gap detection**: Find vertical whitespace gaps wider than word spacing
3. **Library support**: pdfplumber and pymupdf (fitz) can detect columns

Headers, footers, and page numbers: The noise problem

Every page of a 50-page report might have:

- Company name in the header
- "Confidential" watermark
- Page number ("Page 23 of 50")
- Date in the footer
- Section title in the header

Without filtering, your chunks are polluted:

```
BAD chunk:
"Acme Corporation — Confidential
 Page 23 of 50
 Q3 Strategy Memo

 Product Alpha achieved $12.4M in Q2 revenue...

 Acme Corporation — Confidential
 Page 24 of 50
 Q3 Strategy Memo"

GOOD chunk:
"Product Alpha achieved $12.4M in Q2 revenue..."
```

Filtering strategies:

1. **Position-based**: Remove text in top/bottom N% of page (e.g., top 10%, bottom 8%)
2. **Repetition-based**: Find text that appears identically on 3+ pages → header/footer
3. **Pattern matching**: Regex for "Page \d+ of \d+", dates, "Confidential"
4. **Font-size based**: Headers/footers often use smaller or different fonts

Hierarchical documents: Preserving structure

Documents have structure — sections, subsections, bullet points. Flat chunking destroys this:

```
DOCUMENT STRUCTURE:

1. Executive Summary
   1.1 Overview
   1.2 Key Metrics
2. Product Performance
   2.1 Product Alpha
       2.1.1 Revenue
       2.1.2 Growth Drivers
   2.2 Product Beta
3. Risk Register

FLAT CHUNKING loses which section a chunk belongs to:
Chunk 7: "Revenue grew 34% driven by enterprise adoption..."
→ Which product? What section? Context is gone.

STRUCTURE-AWARE CHUNKING preserves it:
Chunk 7: {
  section: "2. Product Performance > 2.1 Product Alpha > 2.1.1 Revenue",
  text: "Revenue grew 34% driven by enterprise adoption...",
  level: 3
}
```

Implementation: Parse section headers (detect numbering, font changes, bold text), maintain a "breadcrumb" stack, and prepend section context to each chunk as metadata.

For Product Leaders

The chunk size decision impacts your entire system

Chunk size is not a technical implementation detail — it determines:

1. **Answer quality**: Too small and you get fragments. Too large and you get diluted, off-topic responses that waste tokens.
2. **Cost**: Larger chunks mean more tokens in the prompt, meaning higher API costs per query. A 2x chunk size increase can mean 30-50% higher costs.
3. **Latency**: More tokens = slower responses. Each additional 500 tokens adds ~200-400ms to response time.
4. **Storage**: More chunks mean more embeddings to store and index. This affects your vector database costs.

Cost and quality impact table

Chunk Size (tokens)	Chunks per 100-page doc	Embedding Storage Cost	Prompt Cost (top-5 retrieval)	Quality
100	~4,000	$0.40/month	~$0.003/query	Low (fragments)
250	~1,600	$0.16/month	~$0.005/query	Medium
500	~800	$0.08/month	~$0.008/query	Good (sweet spot)
1000	~400	$0.04/month	~$0.013/query	Good (with noise)
2000	~200	$0.02/month	~$0.023/query	Declining

Costs assume text-embedding-3-small at $0.02/1M tokens, Claude claude-sonnet-4-6-20250514 at ~$3/1M input tokens, Pinecone starter plan. Actual costs vary.

Which strategy for which document type

Document Type	Recommended Strategy	Chunk Size	Why
Legal contracts	Recursive + overlap	400-600	Clause boundaries
Financial reports	Semantic + tables	500-800	Mixed structure
Technical docs	Recursive	600-1000	Code blocks
Support tickets	Sentence-based	200-400	Short, specific
Academic papers	Semantic	800-1200	Dense arguments
Product specs	Recursive + hierarchy	500-800	Section structure
Meeting transcripts	Sentence-based	300-500	Speaker turns
Scanned documents	Fixed-size + OCR clean	400-600	OCR errors need context

What to tell your engineer

When discussing chunking with your engineering team, ask these questions:

1. **"What does our typical document look like?"** — If it's clean text, basic chunking works. If it has tables, multi-column, or scans, you need specialized extraction.

1. **"What's our chunk overlap?"** — If the answer is "none" or "I don't know," that's a red flag. No overlap means information loss at every boundary.

1. **"How are we handling tables?"** — If tables are being treated as flat text, your financial and structured data queries will return wrong answers.

1. **"Can we see the chunks?"** — Always inspect actual chunks from real documents. If they look garbled, no amount of embedding or retrieval tuning will help.

1. **"What's our preprocessing pipeline for PDFs?"** — There should be explicit steps for header/footer removal, table extraction, and OCR if needed.

Red flags in a chunking implementation

- "We just use the default settings" — Every document type needs tuning
- "Our chunks are 2000 tokens" — Almost certainly too large for precise retrieval
- "We don't do overlap" — Information will be lost at boundaries
- "We use PyPDF2 for everything" — PyPDF2 is basic; pdfplumber handles tables better
- "Tables are fine, they're just text" — Tables need structured extraction

---

➢■ This module continues with 2,000+ lines of engineering content including runnable Python scripts for all chunking strategies, advanced techniques (parent-child, sentence window, semantic chunking), messy PDF handling, and production experiments...

Day 6

Reranking & GraphRAG

Module 10 (Preview)

M10: GraphRAG — When Connections Matter More Than Similarity

Module Overview

Field	Detail
Duration	60 minutes
Day / Slot	Day 6
Prerequisites	M01-M09 (RAG fundamentals, advanced retrieval, reranking)
Tools	Claude API (claude-sonnet-4-6-20250514), Neo4j, LlamaIndex, NetworkX
Running Example	Acme Corp Q3 Strategy Memo
Deliverables	Entity extraction pipeline, Neo4j knowledge graph, hybrid graph+vector router

Why This Matters

The $2.3 Million Question Nobody Could Answer

A top-five management consulting firm had built what they considered a world-class RAG system. 14,000 client engagement reports. Dense embeddings. Cohere reranking. Sub-200ms retrieval. Their system could answer questions like "What were the key findings from the Meridian Healthcare engagement?" with impressive accuracy.

Then a senior partner walked into the room and asked: "Which of our clients have relationships with both the healthcare and financial services sectors? I need to find cross-selling opportunities for our new compliance offering."

The system returned chunks about healthcare clients. Chunks about financial services clients. Chunks mentioning the word "relationship." But it could not find the *intersection*. It could not trace the web of connections: that Meridian Healthcare's board shared three members with FirstBank Corp, that their joint venture in health insurance created a natural bridge, that a former engagement lead now sat on both advisory boards.

The partner needed to understand *relationships between entities* – not similarity between text chunks.

A junior analyst spent three weeks manually mapping these connections. The partner missed a critical pitch window. The firm estimated the lost opportunity at $2.3 million in potential billings.

The problem was architectural. Vector similarity answers "what text looks like this query?" Knowledge graphs answer "how are these things connected?" These are fundamentally different questions, and no amount of embedding model fine-tuning will turn the first into the second.

This module teaches you when connections matter more than similarity – and how to build systems that understand both.

➢■ This module continues with 1,900+ lines covering knowledge graph construction, Neo4j integration, entity extraction, Cypher queries, Microsoft GraphRAG, and a full runnable pipeline...

What’s in the Full Book

Days 1–4: RAG Foundations & Core Techniques

• M01: RAG Foundations — the $38K failure story, managed services decision table
• M02: RAG Pipeline — 5-stage production pipeline, prompt caching
• M03: Vector Databases — Pinecone CRUD, HNSW/IVF, incremental indexing
• M04: Chunking Strategy — 7 strategies, messy PDFs, parent-child, semantic chunking
• M05: Embeddings — OpenAI + local models (Ollama), cosine similarity deep dive
• M06: Query Enhancement — HyDE, query rewriting, step-back prompting
• Lab 01: Build a complete RAG system from scratch

Days 5–8: Advanced RAG & Evaluation

• M07: Multi-Query Retrieval — query decomposition, reciprocal rank fusion
• M08: Hybrid Search — BM25 + dense vectors, alpha sweep optimization
• M09: Reranking & Filtering — Cohere rerank-v3.5, production reranker class
• M10: GraphRAG — Neo4j, entity extraction, Microsoft GraphRAG
• M11: Multimodal RAG — Claude Vision, ColPali, pdfplumber table extraction
• M12: Structured RAG — Text-to-SQL, DataFrame agents, hybrid router
• M13: Evaluation & Observability — RAGAS, LLM-as-a-judge, Langfuse, DeepEval
• Lab 02: Upgrade Lab 01 with reranking, hybrid search, GraphRAG, RAGAS eval

Days 9–10: Agents & Production

• M14: Agent Foundations — ReAct pattern, Claude tool_use API, MCP servers
• M15: Multi-Agent Orchestration — LangGraph state machines, conditional routing
• M16: Agentic RAG — router, iterative, and tool-calling RAG patterns
• M17: Security & Guardrails — 5 defensive layers, Guardrails AI, NeMo, red teaming
• Lab 03: 3-agent LangGraph pipeline + Langfuse + Presidio PII detection

• Capstone: 4-agent RFP generator with revision loop

646 pages • 17 modules • 3 appendices • 4 labs • All runnable Python code • Dual track (Product + Engineering)

About the Author

Prithvi Datla is the founder of KiloByte Collective, where he transforms SMBs into AI-first businesses.

He has built and shipped Dinealog (dinealog.com), a voice agent for restaurants that accepts takeout orders, manages reservations, and gives operators real-time visibility into operations.

He also hosts *Inside The Foundry*, a podcast featuring foundational moments and hard-won wisdom from fellow operators.

Before KiloByte, Prithvi was a product manager at an edtech startup and held positions at multiple Fortune 25 companies. He also founded a hospitality business featuring a boutique hotel in Tulum, Mexico, and established Velvet House, Mexico's best Indian restaurant — which he deliberately closed at its peak.

Prithvi holds a Bachelor's and Master's in Computer Science and Engineering, and an MBA from Cornell University.

This field manual was born from real-world deployments — the failures that taught hard lessons and the patterns that actually work at scale. The dual-track format (Product Leaders + Engineers) exists because the best AI systems are built by teams and people who speak both languages.

Outside of work, Prithvi is a published photographer and an avid golfer, surfer, diver, and dog dad. Learn more at prithvidatla.com.

Get the Full Book

PDF Only

$59

646-page book (PDF) • 17 modules + 3 appendices
All diagrams, decision frameworks • Dual-track format

Full Professional Bundle

$109

★ Most Popular

Everything in PDF Only, plus:
Companion Repository (private GitHub) • 4 labs with runnable code
Capstone: 4-agent RFP generator • Acme Corp dataset + sample RFPs
.env templates + verify script • Lifetime updates
“What Changed Since April 2026” changelog drops

Team License

$299 (up to 5 seats)

Everything in Full Bundle • 5 seats with shared repo access
Bulk discount ($60/person) • Perfect for internal enablement

theragbook.com

Questions? Reach out at prithvidatla.com