O'REILLY®

Hands-On RAG for Production

Design, Develop, and Deploy Production-Ready RAG Applications

Ofer Mendelevitch & Forrest Sheng Bao

Forewords by Sharon Zhou & Jim Dowling

Praise for *Hands-On RAG for Production*

"Hands-On RAG for Production doesn't skip the unglamorous parts. Ofer and Forrest give document parsing, tables, and ingestion the serious treatment they deserve—which is exactly where most real-world RAG systems live or die."

—Jerry Liu, CEO, LlamaIndex

"We are entering an era where software is no longer just a tool we use, but an intelligence we collaborate with. Hands-On RAG for Production correctly identifies that the future of the 'corporate brain' relies on the unification of fragmented institutional knowledge through robust RAG pipelines. For teams building these living, breathing engines of insight, this book serves as the definitive guide to mastering the complex interplay between vector stores, agents, and evaluators."

—Bob Van Luijt, cofounder and CEO, Weaviate

"RAG has quietly become the default architecture for grounding LLMs in enterprise data, but the engineering discipline around it is still catching up. Mendelevitch and Bao lay out a rigorous, end-to-end framework, from ingestion and retrieval design through production evaluation and agentic extensions, that most teams are still figuring out by trial and error."

—Jimmy Lin, ACM and ACL Fellow, David R. Cheriton Chair, University of Waterloo

"Hands-On RAG for Production makes the leap from vector search to GraphRAG feel practical, intuitive, and immediately useful. The knowledge graph chapter gives readers the foundation they need to build more precise, explainable, and trustworthy AI systems."

—Stephen Chin, author of *GraphRAG: The Definitive Guide*, VP of Developer Relations, Neo4j

“While anyone can hack together a toy RAG demo in an afternoon, the gap between a prototype and a production-grade system is vast. Hands-On RAG for Production masterfully explains the intricacies of achieving accuracy, low latency, and scale, while providing the crucial ‘build versus buy’ framework every architect needs. If you’re moving beyond the sandbox, this is a must read.”

—Tallat M. Shafaat, founder and CEO, Vectara

“Hands-On RAG for Production is the first book I’ve seen that treats RAG, agents, and evaluation with true production rigor. Every ML or platform engineer shipping LLM applications at scale should read this.”

—Shailja Gupta, product manager, AI Platforms

“If you’ve built a demo RAG app and now need reliability, metrics, and tooling that scale, this book closes the gap with concrete patterns and hands-on examples.”

—Diva Dugar, software engineer

Hands-On RAG for Production

Design, Develop, and Deploy Production-Ready RAG Applications

Ofer Mendelevitch and Forrest Sheng Bao
Forewords by Sharon Zhou and Jim Dowling

Hands-On RAG for Production

by Ofer Mendelevitch and Forrest Sheng Bao

Printed in the United States of America.

Published by O'Reilly Media, Inc., 141 Stony Circle, Suite 195, Santa Rosa, CA 95401.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Nicole Butterfield

Development Editor: Michele Cronin

Production Editor: Jonathon Owen

Copyeditor: Liz Wheeler

Proofreader: Andrea Schein

Indexer: Krsta Technology Solutions

Interior Designer: David Futato

Cover Designer: Karen Montgomery

Illustrator: Kate Dullea

May 2026: First Edition

Revision History for the First Edition

- 2026-05-27: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9798341621718* for release details.

979-8-341-62171-8

[LSI]

Foreword by Sharon Zhou

The first time I saw a RAG system fail in production, it was because someone had naively chunked their documents on fixed character boundaries and split a legal clause in half. Clause A was in one chunk with some of clause B, and the rest of clause B was in another. The problem was that the second chunk provided a useful, common exception. The RAG system retrieved the first chunk but not the second based on the user's question, so unfortunately, the model answered the user's question with the opposite of what the contract said. Just think: if you were given incomplete or faulty knowledge through a Google search, you'd also have trouble giving the right answer.

No one building the system had been thinking about chunking strategy, not critically. They had been busy debating about which LLM to use. That's why a book like this is so important for those building RAG with LLMs and agents in production.

RAG looks deceptively simple:

- Chunk your documents—easy, that's a string split.
- Embed your chunks—easy, that's a lightweight model API call in a for loop.
- Retrieve the relevant chunks—easy, that's just using search, which has been around a lot longer than modern AI, so in a way, it should have best practices baked in already.
- Hand those chunks to an LLM—easy, that's just appending strings to another string to form a prompt.

You can build a working prototype by one-shotting a language model. But… you can also spend the next year working through parsing, chunking, embedding model choice, retrieval strategy, reranking, evaluation, guardrails, and a dozen other quiet yet important decisions. These are

decisions that can result in something like that legal assistant or in a production-ready system that's robust to what your users ask.

Mendelevitch and Bao have written a book that either catches you before you make those mistakes or helps you recover if (or when) you do. The goal is to save you time and headache so you can leverage more of the intelligence of language models, with RAG systematically helping, rather than hurting, production performance.

If you're building RAG systems that real people will depend on, this book is a great place to get started and download some of that knowledge.

Sharon Zhou, PhD
Corporate VP of Engineering & AI
AI Chief of Staff to the CEO
AMD

Foreword by Jim Dowling

Ilya Sutskever, the leading figure in the development of large language models (LLMs), claimed that because LLMs can accurately predict the next token, they understand the underlying reality that led to the creation of that token. In other words, LLMs have an internal model of the world based on language. The LLM's internal model can reason about anything in the world, provided it is first transformed into language the LLM was trained on. The LLM has encyclopedic knowledge of the world and can answer queries using the huge volume of knowledge it acquired from the vast number of documents it was trained on.

But if you want an LLM to provide insights on anything that happened after its training cutoff date, you need to include all the relevant information (known as *context*) in your prompt to the LLM so that it can answer the question. LLMs can even learn and generalize from the context you provide, in what is known as *in-context learning*. But if you converse directly with an LLM (not via a chatbot), it will be like talking to Leonard Shelby from *Memento,* who tragically could form no new long-term memories. Chatbots give you the illusion that the LLM has memory as they provide the full conversation as context in every prompt.

In a way, the LLMs are like computers that only have ROM but no RAM to make new memories. Just as Leonard Shelby got creative by using his body to store memories as tattoos (mementos), LLMs can use external systems as memory that can later be retrieved when needed.

LLMs also cannot act in the world. They are like the brain-in-a-jar from Steve Martin's *The Man with Two Brains*. They know stuff, and they can reason, but they are not connected to the world. While LLMs cannot perform actions in the world themselves, they can respond in a language (JSON) that enables clients to execute functions on their behalf. And functions are the building blocks of computer systems—they can trigger actions in either the virtual or physical world.

In recent times, agents have emerged as the body, or harness, that enables LLMs to act in the world. Agents can perform a task on behalf of an LLM by

calling functions (in the *agentic loop*). The loop exits when the LLM has deemed the task to have been completed. But calling a function is relatively easy for agents. What is not easy is providing the LLM with the correct context so it can decide on what function to call and with which parameters. You cannot just dump all available context into your prompt when querying an LLM. The LLM's context window has a fixed size, measured in number of tokens (not words), that limits how much context you can provide in a single prompt. A significant challenge when building agents is discovering and adding the right context information to a prompt so that an LLM provides a better answer than if there was no context.

The solution to this challenge has come to be known as *retrieval-augmented generation* (RAG).

To summarize the current state of affairs, our AI revolution is being built on an amnesiac brain-in-a-jar, and we have much engineering work to do to build useful systems by providing LLMs with the right context using RAG. How you implement a RAG solution is key to the success of your agentic systems and to generating value from AI. The first RAG solutions extracted the text from the user's query and retrieved context from a vector database using approximate nearest neighbor search. But as industry adopts AI to build production systems, so grows the need for consistent retrieval of context, more complete context, and the presence of guardrails when retrieving context. There is also the need to handle multimodal RAG, real-time context retrieval, and the integration of knowledge graphs. Each of these areas introduces its own challenges for production-grade context retrieval.

Ofer and Forrest have written an important book the industry needs right now to help developers move from prompt engineering to building production AI systems using RAG. The next decade of agentic software will be built by those who can connect the reasoning power of the LLM to the body of living data in organizations. These systems need to be reliable, secure, and trustworthy, and they need to work at scale. If you are a developer, architect, or a leader tasked with bringing LLMs into the real world, this is your manual.

Welcome to the era of production RAG.

Jim Dowling
CEO of Hopsworks

Preface

You've seen the "easy RAG" demo: a few lines of Python, a vector database, and an API key. In ten minutes, the chatbot is answering questions grounded in a few company PDF files. It feels like magic.

Perhaps you've even taken the next step at your company: built a retrieval-augmented generation (RAG) application, hosted it on your favorite cloud platform, and scaled your knowledge base to several hundred documents. It looks and feels like a "real" application.

Then comes "Day 2."

As users begin to ask more complex questions, the initial "magic" starts to fray. The cracks appear when your RAG application confidently hallucinates a nonexistent regulatory policy or fumbles a troubleshooting task by citing a generic marketing brochure instead of a specific engineering schematic. Tension rises as stakeholders weigh in: your CIO demands answers on security and data privacy, while R&D reports that the system remains "blind" to the vital flowcharts and diagrams buried within their PDF files.

You soon discover that the retrieval precision that held firm for a thousand documents dissolves into "semantic noise" at ten or a hundred times that volume. As the system expands, accuracy degrades, while latency spikes under the weight of production traffic. When the inevitable demand for a reliability audit arrives, you're forced to confront a sobering reality: you lack the repeatable, metrics-driven evaluation framework necessary to diagnose which specific component in your pipeline is actually breaking.

This is the *production wall*. It is the chasm between the initial RAG proof of concept (POC) and a resilient, enterprise-grade AI application. Crossing it requires more than just better prompts; it requires a fundamental shift in perspective.

The transition from a demo to an enterprise-scale application makes finding a needle in a haystack look simple. It requires solving for multimodal complexity, rigorous statistical reliability, and the operational overhead of a distributed AI system. Most projects fail because they underestimate this scope, treating RAG as a simple “plug-and-play” feature rather than an evolving engineering discipline.

What This Book Is About

Many developers hit the production wall and assume the technology is flawed. It isn't. The problem is that the techniques used to build a demo are fundamentally different from those required to build an enterprise-scale product.

This book is the bridge across that chasm. We tackle the unique operational challenges of RAG in production. By the end of this journey, you will be equipped to do the following:

Implement high-precision retrieval

> Move beyond simple vector search to leverage hybrid search, relevance reranking, or knowledge graphs, ensuring accuracy for complex questions at enterprise scale.

Eliminate hallucinations

> Diagnose and reduce large language model (LLM) "hallucinations" using retrieval-aware guardrails, while ensuring your RAG system has the most up-to-date enterprise data for grounding its responses.

Integrate multimodal content

> Expand your system's capabilities to accurately interpret tables, images, diagrams, and videos, and integrate their information content into the RAG responses.

Establish rigorous evaluation

Move away from “vibe-based” testing—the habit of asking the chatbot three questions and assuming it works because the answers “look” right—toward repeatable, automated metrics that provide a statistical guarantee of reliability.

Optimize for the real world

Make informed build-versus-buy decisions and deploy systems that survive real-user latency constraints and deep observability requirements.

Our focus is *RAG-specific resiliency*: turning a brittle demo into a hardened enterprise asset. While we respect the foundations of general systems engineering, this book isn’t a generic primer on continuous integration and continuous delivery (CI/CD) or cloud infrastructure. Instead, we provide the blueprints to solve for the unique failure modes of RAG—from low-latency, high-accuracy retrieval optimization to deep observability—focusing on the design and implementation of a system that is visible, measurable, and reliable under the weight of production traffic and the messiness of enterprise data.

Who This Book Is For

This book is for the builders in the trenches of the AI era—the software engineers, machine learning engineers, and data architects who know that the distance between a successful `pip install` and a reliable production system is measured in sleepless nights.

You are likely responsible for putting RAG systems on the critical path: the systems that customers, employees, and leadership now depend on. You aren't looking for another tutorial on prompt engineering; you are tasked with the structural heavy lifting. Whether you are designing document pipelines that don't choke on complex PDF files, implementing guardrails to kill hallucinations, or building the evaluation frameworks that prove your system actually works, this book is your guide.

While this is primarily an engineering text, it serves as a reality check for technical product managers and architects. If you define requirements, you need to understand the mechanical limits of RAG systems, and the role each component plays in the RAG stack. This book provides the technical intuition to distinguish between a realistic latency budget and a fantasy, ensuring you don't promise features that physics and compute costs can't deliver.

Who This Book Is Not For

To ensure this book is the right fit for your current journey, it is important to note that we skip the introductory basics. This is an advanced engineering guide, not a foundational Python course. We assume a level of comfort with Python's core structures and basic programming patterns; if you are still distinguishing between lists and dictionaries, you will likely find the technical depth of our implementations more frustrating than helpful.

Furthermore, our lens is strictly focused on applied AI rather than academic theory. While we dive deep into the orchestration and optimization of RAG systems, we don't spend time on the underlying calculus of neural networks, or the mathematical proofs behind transformer architectures.

Finally, this is a "hands-on" book in the literal sense—the code snippets throughout the book and the associated GitHub repository (which includes full code samples) are important to gain full understanding of the material. It is not intended for "no-code" enthusiasts or casual consumers. If your goal is to assemble RAG applications without engaging directly with code, system design, and debugging, this book will likely feel misaligned with your expectations.

Using Code Examples

We provide extensive code examples in this book, which are all open-source and available online at *https://github.com/ofermend/hands-on-rag*, in the form of Jupyter notebooks. These are interactive coding documents containing text and executable code snippets in Python.

The easiest and quickest way to get started is to run these notebooks locally or using Google Colab (a free service that allows you to run any Jupyter notebook directly online without having to install anything on your machine).

Readers are encouraged to modify the notebooks, swap models, adjust chunking strategies, and intentionally break components to observe failure modes—many of the most important lessons in this book only become obvious when things go wrong.

Installation details and environment setup instructions are provided in the GitHub repository.

If you have a technical question or a problem using the code examples, please send email to *support@oreilly.com*.

Prerequisites

This is a hands-on, "code-first" book. To get the most out of the chapters ahead, you should be comfortable navigating a Python-based development environment. Here is the specific toolkit we expect you to bring to the table:

Intermediate Python

You should have a solid grasp of Python fundamentals (classes, functions, and decorators). Because production RAG often involves high-concurrency tasks, familiarity with asynchronous programming will be highly beneficial. We make use of data orchestration and cleaning libraries like pandas. While you don't need to be a data scientist, you should know how to slice a DataFrame and handle vectorized operations.

Web services and APIs

Since RAG relies on communicating with LLM providers and external services, you should be comfortable working with Representational State Transfer (REST) APIs. Experience with libraries like httpx or Requests is essential for handling timeouts, retries, and authentication.

LLM fundamentals

We assume you understand what a large language model is and have experience interacting with LLMs via APIs (like OpenAI, Anthropic, or local providers). While we don't require you to know the underlying calculus of the Transformer architecture, you should be comfortable using them and understand basic concepts like tokenization, context windows, and the basics of how a prompt influences a model's output.

Visualization

We use Matplotlib and seaborn to visualize retrieval accuracy and evaluation metrics. Being able to interpret a distribution plot or a heatmap will help you "see" how your model is performing.