O'REILLY®

RAG with Python Cookbook

Practical Recipes from Data Preprocessing to LLM Agents

Dominik Polzer

RAG with Python Cookbook

As businesses race to unlock the full potential of large language models (LLMs), a critical challenge has emerged: How do you connect these tools to real-time, external data to solve real-world problems? Retrieval-augmented generation (RAG) is the answer. By combining LLMs with information retrieval, RAG empowers you to build everything from intelligent chatbots to autonomous, task-solving agents.

Packed with over 70 practical recipes, this go-to guide tackles a wide range of GenAI applications through structured hands-on learning. Author Dominik Polzer provides the tools you need to design, implement, and optimize RAG systems for your unique use cases. Whether you're working with simple data retrieval or designing cutting-edge autonomous agents, this cookbook will help you stay ahead of the curve.

- **Learn core RAG components including embedding, retrieval, and generation techniques**
- **Understand advanced workflows like semantic-aware chunking and multiquery prompting**
- **Build custom solutions such as chatbots and autonomous agents for specific data challenges**
- **Continuously evaluate and optimize systems for accuracy, relevance, and performance**

Dominik Polzer is an experienced machine learning engineer who has worked at leading companies like Siemens. Specializing in forecasting, anomaly detection, and generative AI, he leads initiatives leveraging foundation models and RAG systems to automate business processes. He also shares ML insights on his popular Medium blog.

DATA

US $79.99 CAN $99.99
ISBN: 979-8-341-60056-0

9 798341 600560 57999

O'REILLY®

RAG with Python Cookbook

Practical Recipes from Data Preprocessing to LLM Agents

Dominik Polzer

O'REILLY®

RAG with Python Cookbook
by Dominik Polzer

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: Jeff Bleiel
Production Editor: Jonathon Owen
Copyeditor: Sharon Wilkey
Proofreader: Dave Awl
Indexer: BIM Creatives, LLC
Cover Designer: Susan Brown
Cover Illustrator: José Marzan Jr.
Interior Designer: David Futato
Illustrator: Kate Dullea

April 2026: First Edition

Revision History for the First Edition
2026-04-27: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9798341600560* for release details.

979-8-341-60056-0

[LSI]

Table of Contents

Preface

In 2017, Google researchers published the paper "Attention Is All You Need" and introduced the Transformer architecture, a breakthrough that reshaped modern AI. Over the following years, large foundation models demonstrated what happens when these ideas are scaled. The models began to write coherent text, answer complex questions, and generate working code. For the first time, software systems could interact with language in ways that felt broadly useful in real applications, not just impressive in research demos.

Yet these models had an important limitation. They were powerful but isolated, prone to hallucinating facts, lacking access to up-to-date information, and unable to work with private company data. Retrieval-augmented generation (RAG) addresses these gaps by coupling language models with external knowledge sources. I see RAG and agentic RAG as a key step toward AI systems that emulate human problem-solving. They actively gather new information, interpret it in context, and continuously plan their next steps based on their findings. By connecting foundation models to external knowledge sources, RAG grounds model outputs in verifiable data and enables systems to reason over trusted information when handling complex tasks.

This book is about building production-ready RAG systems. Each recipe focuses on a concrete engineering challenge that appears when moving from prototype to dependable application and explains the trade-offs behind key design decisions. You'll learn how to design pipelines, select retrieval strategies, evaluate system quality, and operate RAG systems at scale. The goal is to help you build solutions that are accurate, scalable, and maintainable—systems that perform reliably not only in demos but in real-world environments.

Who This Book Is For

This book is written for developers and data scientists who want to build practical generative AI applications. You should be comfortable with Python and familiar with APIs, data processing, and general software development. A deep machine learning

background is not required. The necessary concepts are introduced as they appear, but you should be ready to write, run, and debug code along the way.

What You'll Learn and How the Book Is Organized

The 11 chapters follow the natural progression of building a RAG system, from core concepts to production deployment:

- Chapters 1 and 2 cover foundations. You'll identify high-value use cases, set up your environment, work with prompts, and select appropriate foundation models.
- Chapters 3 and 4 focus on the data pipeline. You'll load data from documents, databases, images, audio, and video, then prepare it through cleaning, chunking, and metadata enrichment.
- Chapters 5 and 6 explain embeddings and storage. You'll learn how embeddings represent semantic meaning, choose embedding models, and select vector databases based on your requirements.
- Chapter 7 introduces advanced retrieval techniques such as metadata filtering, multiquery retrieval, HyDE, reranking, and query decomposition to improve accuracy.
- Chapters 8 and 9 explore intelligent systems. You'll build agentic workflows with tools and learn how knowledge graphs preserve relationships between entities.
- Chapters 10 and 11 address production readiness. You'll evaluate RAG systems with automated metrics and human judgment, generate synthetic test data, and deploy applications with Streamlit, Docker, and Amazon Web Services.

Each chapter contains practical recipes with working code that you can adapt and extend in your own projects.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

This element signifies a tip or suggestion.

This element signifies a general note.

This element indicates a warning or caution.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at *https://github.com/polzerdo55862/RAG-with-Python-Cookbook*.

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

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but generally do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*RAG with Python Cookbook* by Dominik Polzer (O'Reilly). Copyright 2026 O'Reilly Media, 979-8-341-60056-0."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

O'Reilly Online Learning

For more than 40 years, O'Reilly Media has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, visit *https://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
141 Stony Circle, Suite 195
Santa Rosa, CA 95401
800-889-8969 (in the United States or Canada)
707-827-7019 (international or local)
707-829-0104 (fax)
support@oreilly.com
https://oreilly.com/about/contact.html

We have a web page for this book, where we list errata and any additional information. You can access this page at *https://oreil.ly/rag-with-python-cookbook*.

For news and information about our books and courses, visit *https://oreilly.com*.

Find us on LinkedIn: *https://linkedin.com/company/oreilly*.

Watch us on YouTube: *https://youtube.com/oreillymedia*.

Acknowledgments

Writing this book has been one of the most challenging and rewarding projects I have taken on. Although I have enjoyed writing for years, shaping a complete manuscript taught me more than I expected. If this book helps you build better RAG systems or solve real problems, I would love to hear about it. Please feel free to reach out via LinkedIn or GitHub. Your feedback and stories are incredibly motivating.

I am deeply grateful to the technical reviewers who dedicated their time and expertise to improving this book. Vishwesh Ravi Shrimali, Leonie Monigatti, Laura Uzcategui,

Prashanth Josyula, Louis-François Bouchard, Nicolas Bièvre, and Susan Shu Chang provided thoughtful feedback, caught errors, clarified explanations, and helped ensure that the examples work as intended. Any remaining mistakes are my own.

This project would not exist without the outstanding team at O'Reilly. I am especially indebted to Jeff Bleiel and Jill Leonard for their editorial guidance, and to Nicole Butterfield for initiating and supporting this project from the beginning. Thanks also to the production team for turning the manuscript into a polished book.

I owe special thanks to Mirko Einert, who has been both a manager and a mentor. His drive to challenge the status quo helped evolve RAG systems from prototypes into core systems that transform our procurement processes at Siemens Energy.

Finally, I am deeply grateful to my girlfriend, Luisa. Writing this book required many late nights and weekends. She gave me the space to focus, encouraged me when progress felt slow, and reminded me to rest when needed. Her support made this book possible. Thank you, Luisa.

CHAPTER 1
Getting Started with RAG

Large language models (LLMs) have transformed how we approach complex cognitive tasks—from writing production code to analyzing financial reports to translating dozens of languages. Complementary foundation models now handle vision, speech synthesis and recognition, audio processing, image generation, and multimodal reasoning, all built on similar transformer architectures that can process and generate human-like content across multiple domains.

Despite these capabilities, these models have fundamental structural limitations. They don't have access to your private or confidential data unless you provide it. Their context windows limit how much information they can consider at a time, which makes long documents expensive or impossible to analyze in a single pass. And when these models lack the right information, they tend to hallucinate rather than admit uncertainty.

Retrieval augmented generation (RAG) addresses all three problems at once. It gives the model controlled access to external knowledge, lets it work with far more information than fits into a single prompt, and grounds its answers in retrieved evidence instead of guesswork. When a user asks a question, the system first retrieves relevant information from a knowledge source and then passes that context to the model to generate a response.

Figure 1-1 shows the simplest form of a RAG system. A retriever searches a knowledge source such as a vector store or database and returns relevant passages. Think of the vector store as a specialized search engine that indexes your specific documents and knowledge base, rather than the entire web like Google. A generator, typically an LLM, uses that retrieved material to produce a grounded answer.

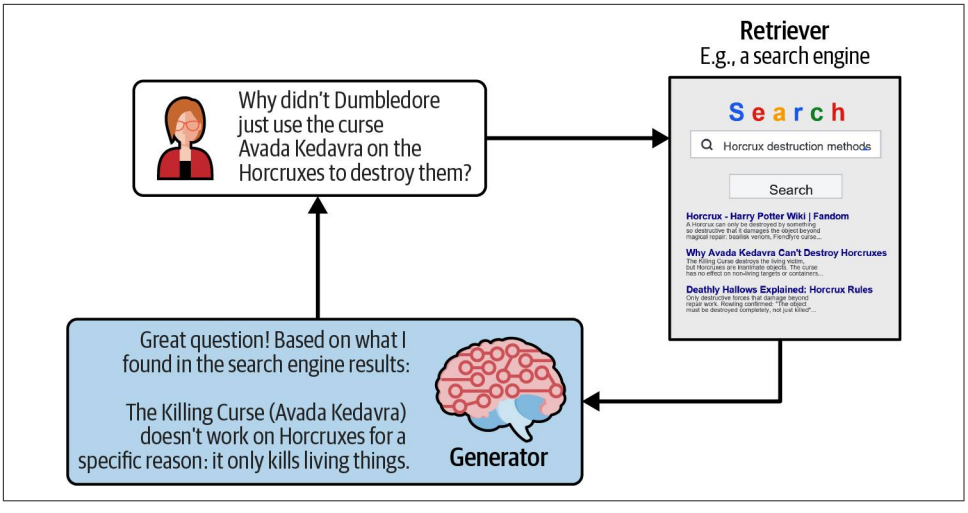

Figure 1-1. A RAG chatbot retrieves relevant information and generates contextual responses

RAG is most useful when knowledge is spread across many documents, when answers require synthesis rather than simple lookup, or when the data is too large or too diverse for a human to review. Modern systems often use multiple retrievers. One may search documents, another may query structured databases, and others may call external APIs. The model coordinates these tools to decide which information to use for each question.

Figure 1-2 shows exactly this approach—a RAG system with multiple API endpoints that can search a Notion database, search the web, perform calculations, and access both SQL and vector databases to search the knowledge base. The model decides which retriever to call based on the question and combines the retrieved information to generate an answer. This architecture allows the system to handle a wide variety of questions.

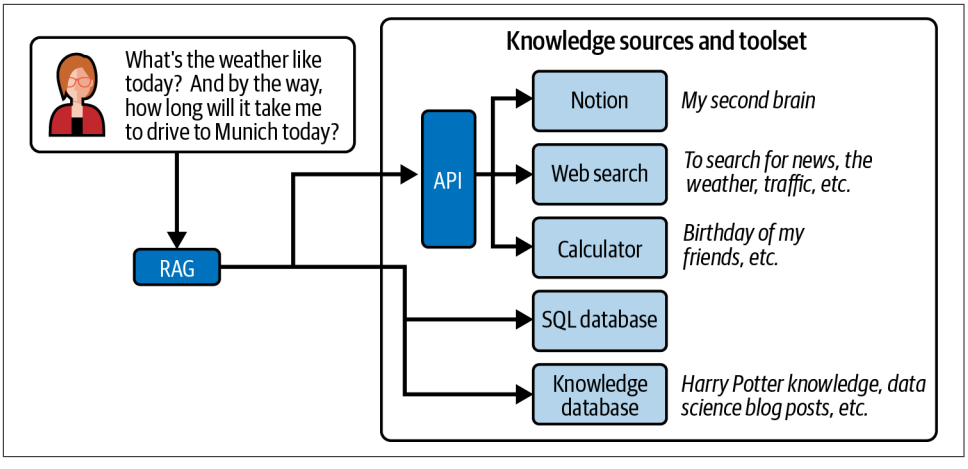

Figure 1-2. A RAG system with multiple retrievers answering questions about Harry Potter books

While many people associate RAG primarily with chatbots, chat interfaces aren't always the optimal solution. Chat works well for exploratory questioning and initial user interactions, but often the best approach combines multiple interfaces. A typical workflow might start with a conversational interface to understand user intent, then transition to a dashboard, form-based application, or automated report once the system understands the user's needs. Many RAG systems also run entirely in the background, delivering results through APIs, scheduled reports, or integration with existing business applications.

This chapter walks you through setting up your environment and building your first basic RAG application.

All code examples are available in the book's GitHub repository.

1.1 Identifying High-Value RAG Use Cases for Your Organization

Problem

Your organization is exploring RAG and needs to identify use cases that deliver the most value.

Solution

Start by identifying use cases that solve concrete problems. Many RAG pilot projects fail because they're too complex, too expensive, or don't address real user pain.

Focus on two high-value areas where RAG delivers measurable impact:

Data extraction
: Converting unstructured data (contracts, invoices, technical drawings, meeting notes) into structured, searchable information

Process automation
: Handling repetitive tasks that require decision making and adaptation

Table 1-1 provides some guidance on how to evaluate different scenarios in terms of RAG fit and potential value. A value of 1 indicates a RAG fit of no or little value, and 5 indicates the best fit.

Table 1-1. Evaluating RAG use cases

RAG fit (1–5)	Use case	Reasoning
1	"I want to chat with my quarterly report."	Single document, low volume. Reading it directly or analyzing with general apps (e.g., ChatGPT or Claude) can do this well.
2	"Summarize this one document for me."	General models (Claude, ChatGPT) already do this well. Custom RAG adds little value.
2	"Automate tasks that require high-stakes decisions."	Possible, but too risky without strong controls (human in the loop, auditing, fail-safes). LLMs still make mistakes.
4	"Meeting recordings pile up, but insights never make it into our knowledge base."	Converts inaccessible data (audio, videos, long unstructured notes) into searchable form. Strong ongoing value, but quality depends on transcription.
4	"You must check technical drawings against specification documents."	Multimodal comparison that humans find tedious. Automation saves time, but requires strong evaluation and exception handling.
5	"We have 10,000 contracts but don't know which ones have auto-renewal clauses."	High volume makes manual review impractical. Clear extraction task with verifiable output.
5	"Our customer support tickets contain product issues, but we can't aggregate them."	Cross-document pattern detection. Humans can't spot trends at scale.
5	"You get hundreds of customer inquiries daily and need to route them to the right team."	Repetitive classification/routing with clear success criteria and measurable return on investment.

Many use cases combine data extraction, analysis, and automation. Figure 1-3 shows an example from engineering and quality management. Manufacturers receive thousands of quality documents daily from suppliers. RAG applications extract key information, compare it against ISO standards in a vector database, check compliance, and generate reports. If information is missing or contains noncompliant parameters (like welding thickness in this example), the system can automatically retrieve the supplier's contact information and generate a draft follow-up email.

This automation saves hours of manual review and speeds up issue resolution, an example for a high-value RAG use case.

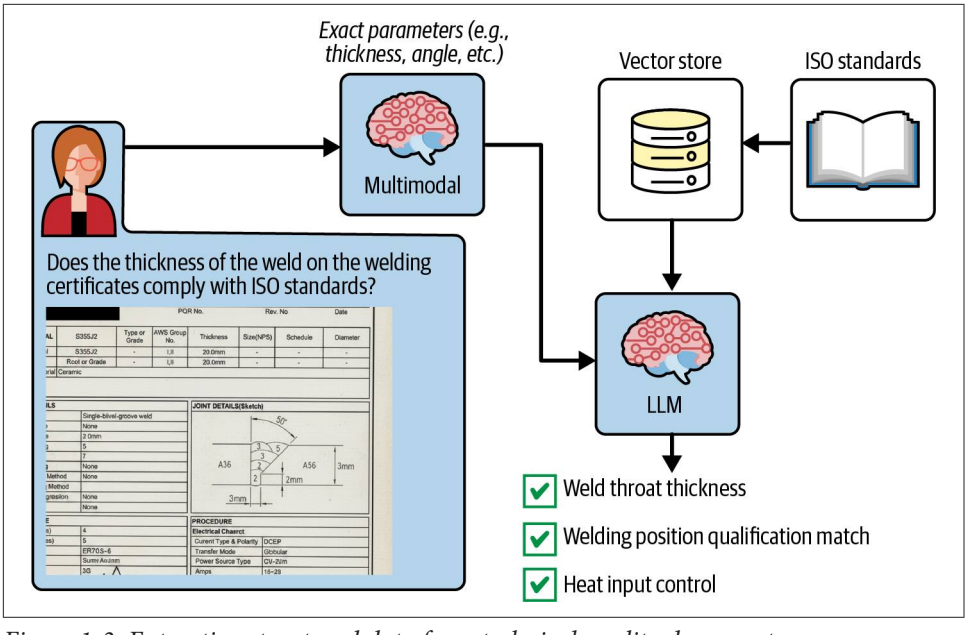

Figure 1-3. Extracting structured data from technical quality documents

Discussion

RAG works for automation because LLMs can interpret intent and context, not just match patterns. Traditional automation requires explicit rules for every scenario. When a supplier email says "delayed due to weather" versus "shipment rescheduled," rule-based systems need separate handlers for each phrase. RAG-based systems understand that both express the same operational impact.

This capability matters most when data is unstructured and variable. Use RAG when each input differs slightly but requires similar handling—customer emails, contract clauses, or incident reports. Don't use RAG for simple lookups, fixed-format data extraction, or tasks based on unchanging rules. If you can write a regular expression (regex) or SQL query that handles 95% of cases, RAG adds unnecessary complexity and cost.

RAG trades simplicity for flexibility. Traditional automation flows that rely on rule-based decision steps run faster, cost less per operation, and fail in predictable ways. LLM-powered RAG systems, on the other hand, handle edge cases better but require model hosting, prompt engineering, and evaluation frameworks. Factor in both token costs and engineering effort when comparing approaches.

Start with high-volume tasks where manual review creates bottlenecks. A system that processes 10 documents daily may not justify RAG development costs, but one that processes 1,000 documents daily does. Each implementation teaches you about

prompt patterns, evaluation metrics, and failure modes that apply to subsequent projects.

Avoid complex multisystem integrations as first projects. Success requires both RAG expertise and organizational buy-in. After demonstrating value on focused use cases, expand to scenarios requiring cross-functional coordination or multiple data sources.

See Also

- Dasha Maliugina's blog post "10 RAG Examples and Use Cases from Real Companies" provides an overview of practical RAG implementations from organizations such as LinkedIn, Harvard Business School, Vimeo, and Pinterest.
- The open source repository GenAI & LLM System Design: 500+ Production Case Studies collects real-world examples of production generative AI (GenAI) systems, including many architectures and deployment patterns based on RAG.

1.2 Choosing Your IDE and Coding Agent Setup

Problem

You want to set yourself up to quickly develop and share RAG use cases in Python.

Solution

Popular Python IDEs for RAG development include the following:

Visual Studio (VS) Code
: Lightweight, extensible, with strong Python support

PyCharm
: Full-featured Python IDE with advanced debugging

Jupyter Notebook/Lab
: Interactive development with inline visualization

Spyder
: Scientific computing focus

Vim/Neovim
: Terminal-based, highly customizable