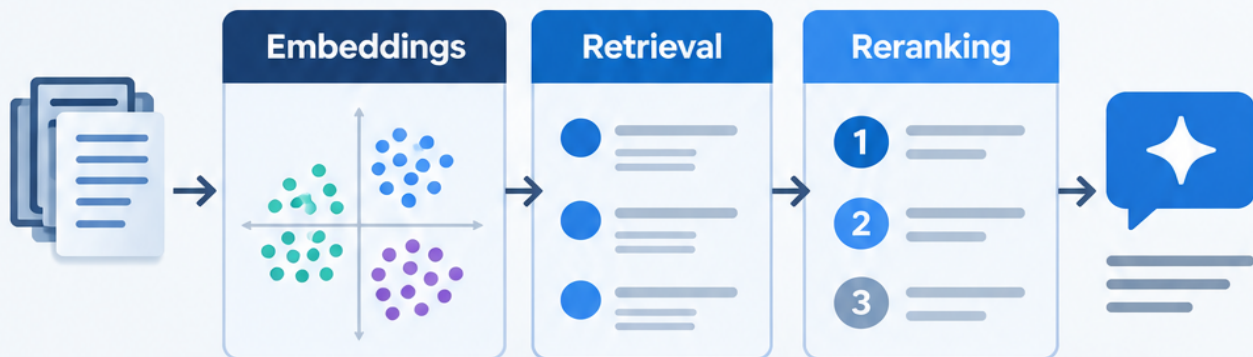


Embeddings and Reranking in Retrieval-Augmented Generation

Building Production-Grade Search Systems with Dense Retrieval and Neural Reranking



TOBIAS RICHTER

Embeddings and Reranking in Retrieval-Augmented Generation

Building Production-Grade Search Systems with Dense
Retrieval and Neural Reranking

Steve Publications

This book is available at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>

This version was published on 2026-09-27



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2026 Steve Publications

Contents

Building Production-Grade Search Systems with Dense Retrieval and Neural Reranking	1
Introduction: Why RAG Needs Better Retrieval	2
Chapter 1: The RAG Paradigm	5
The Retrieval-Then-Generate Architecture	5
Why Retrieval Quality Determines Output Quality	6
The Complete RAG Data Flow	7
RAG vs Finetuning vs Full Pretraining	9
Common Pitfalls and Misconceptions	11
What This Book Will Build	13
Chapter 2: Vector Representations and Similarity	15
From Tokens to Vectors: The Basic Idea	15
Embedding Spaces and Semantic Geometry	16
Cosine Similarity, Dot Product, and Euclidean Distance	18
Normalization and Its Effects on Similarity	20
High-Dimensional Geometry: What Goes Wrong at Large Dimensions	21
Choosing a Similarity Metric for Your Use Case	23
Chapter 3: Embedding Model Architectures	25
Transformer Encoders as Embedding Backbones	25
Bi-Encoder vs Cross-Encoder Architectures	25
Late-Interaction Models and ColBERT	25
Training Objectives: Contrastive Learning, Triplets, and Hard Negatives	25
Mean-Pooling, CLS Tokens, and Output Strategies	25
Instruction-Aware and Prompt-Based Embeddings	26
Chapter 4: Dense Retrieval vs Sparse Retrieval	27
TF-IDF and BM25: Sparse Retrieval Fundamentals	27

CONTENTS

How Dense Retrieval Works Under the Hood	27
The Lexical vs Semantic Gap	27
When Sparse Beats Dense (and Vice Versa)	27
Hybrid Search: Combining Dense and Sparse Signals	27
Reciprocal Rank Fusion and Other Fusion Strategies	28
Chapter 5: Document Preprocessing and Text Normalization	29
Document Parsing: PDFs, HTML, Office Files, and Code	29
Text Cleaning and Normalization Strategies	29
Handling Special Characters, Numbers, and Dates	29
Language Detection and Multilingual Preprocessing	29
Metadata Extraction and Structuring	29
Common Preprocessing Mistakes That Hurt Retrieval	30
Chapter 6: Context Chunking Strategies – Part I: Foundational Methods 31	31
Why Chunking Is The Most Underappreciated RAG Decision	31
Fixed-Size Chunking: The Baseline Everyone Starts With	31
Sentence-Based and Paragraph-Based Chunking	31
Sliding-Window Chunking and Overlap Strategies	31
Recursive Character Text Splitting	32
Choosing Chunk Size and Overlap: Evidence-Based Guidance	32
Chapter 7: Context Chunking Strategies – Part II: Advanced Methods . 33	33
Semantic Chunking: Cluster-Based and Boundary Detection Methods 33	33
Hierarchical Chunking and Parent-Document Retrieval	33
Structure-Aware Chunking for Markdown, HTML, and Code	33
Query-Aware and Dynamic Chunking	33
Long-Context Chunking for Large Language Models	33
Evaluating Chunking Quality: Metrics and Methods	34
Chapter 8: Vector Indexing and Approximate Nearest Neighbor Search 35	35
Exact Nearest Neighbor vs Approximate Nearest Neighbor	35
HNSW: The Workhorse of Modern Vector Search	35
IVF, Product Quantization, and Other Indexing Strategies	35
Index Build Time vs Query Latency Trade-Offs	35
Choosing m, ef_construction, ef_search: Practical Tuning	35
Vector Database Selection: Pinecone, Weaviate, Qdrant, Milvus, and More	36
Chapter 9: Building the Ingestion Pipeline	37

Pipeline Architecture: Batch vs Streaming vs Hybrid	37
Document Loading and Format Handling	37
Chunking, Embedding, and Indexing as Distinct Stages	37
Error Handling and Dead Letter Queues	37
Incremental Updates and Document Versioning	37
Scalable Ingestion Patterns for Large Corpora	38
Chapter 10: Query Processing and Expansion	39
Understanding the Query: Intent, Ambiguity, and Reformulation . . .	39
Query Embedding with the Same or Different Models	39
Query Expansion: HyDE, StepBack, and Similar Techniques	39
Multi-Query Retrieval and Ensemble Queries	39
Query Classification and Routing	39
Handling Multi-Turn Conversational Queries	40
Chapter 11: Candidate Retrieval Strategies	41
Single-Stage Dense Retrieval	41
Hybrid Retrieval: Dense Plus Sparse Combined	41
Multi-Vector Retrieval and ColBERT-Style Approaches	41
Metadata Filtering and Faceted Search	41
Semantic Search with Numerical and Temporal Filters	41
Controlling Recall: How Many Candidates to Retrieve	42
Chapter 12: Reranking Models: Theory and Architectures	43
Why Reranking Improves Retrieval Quality	43
Cross-Encoder Reranking: Architecture and Mechanics	43
Bi-Encoder vs Cross-Encoder vs Late-Interaction for Reranking	43
Training Rerankers: Pairwise, Listwise, and Pointwise Methods	43
Domain Adaptation and Fine-Tuning Rerankers	43
Latency vs Accuracy: When Is Reranking Worth It	44
Chapter 13: Reranking Model Implementations	45
Using Off-the-Shelf Rerankers (BGE, Cohere, Jina, E5)	45
Batch vs Streaming Reranking Architectures	45
Prompt-Based Reranking and LLM Reranking	45
Adaptive and Conditional Reranking	45
Cascaded and Multi-Stage Reranking Pipelines	45
Cost Optimization for Reranking at Scale	46
Chapter 14: Context Assembly and Prompt Construction	47

CONTENTS

- Selecting Which Chunks to Include 47
- Ordering Chunks: Recency, Relevance, and Narrative Flow 47
- Context Window Budgeting and Truncation Strategies 47
- Adding Source Attribution and Chunk Metadata 47
- Dynamic Context Selection Based on Query Complexity 47
- Prompt Templates That Make Retrieved Context Work 48
- Chapter 15: End-to-End RAG System Design 49**
 - The Complete RAG Architecture: All Components Together 49
 - Service Boundaries and API Design 49
 - Synchronous vs Asynchronous Request Flow 49
 - Caching Strategies for Embeddings and Retrieval Results 49
 - Rate Limiting and Backpressure Handling 49
 - Multi-Tenant and Multi-Index Architectures 50
- Chapter 16: Evaluating Embedding and Reranking Models 51**
 - Retrieval Metrics: Recall@K, Precision@K, MRR, MAP, NDCG 51
 - Building Evaluation Datasets: Queries and Ground Truth 51
 - Model Selection Through Systematic Benchmarking 51
 - Evaluating Without Ground Truth: Heuristics and Proxies 51
 - A/B Testing Retrieval Quality in Production 51
 - Ablation Studies and Controlled Experiments 52
- Chapter 17: Evaluating End-to-End RAG Performance 53**
 - Answer Correctness and Factual Accuracy 53
 - Faithfulness and Groundedness: Did the Answer Use the Context? . . 53
 - Context Utilization and Relevance Scores 53
 - Latency, Throughput, and Cost as Quality Metrics 53
 - Automated Evaluation Frameworks (RAGAS, DeepEval, etc.) 53
 - Human Evaluation Design and Execution 54
- Chapter 18: Advanced Optimization Techniques 55**
 - Optimizing Embedding Dimensionality and Model Size 55
 - Quantization and Compression for Embeddings 55
 - GPU Acceleration and Batch Inference Patterns 55
 - Distributed Vector Indexing and Shard Management 55
 - Memory-Optimized Retrieval Architectures 55
 - Throughput and Latency Optimization Checklist 56
- Chapter 19: Debugging and Troubleshooting RAG Systems 57**

CONTENTS

The RAG Debugging Playbook: Where to Start	57
Diagnosing Bad Retrieval: Query, Index, or Model?	57
Identifying Chunking Failures and Context Gaps	57
Spotting Reranking Degradation and Cascading Errors	57
Hallucination Diagnosis and Mitigation	57
Logging, Tracing, and Observability Patterns	58
Chapter 20: Production Deployment and Operations	59
Infrastructure and Containerization	59
Model Serving: Triton, vLLM, and Custom Endpoints	59
Monitoring: Metrics, Alerts, and Dashboards	59
Versioning Models, Embeddings, and Chunking Logic	59
Rollback Strategies and Canary Releases	59
Disaster Recovery and Index Backups	60
Chapter 21: Security, Privacy, and Access Control	61
Access Control and Document-Level Permissions	61
PII Detection and Redaction in RAG Pipelines	61
Prompt Injection and Retrieval Poisoning Attacks	61
Data Isolation in Multi-Tenant Systems	61
Audit Logging and Compliance Requirements	61
Securing Embeddings Against Reconstruction Attacks	62
Chapter 22: Case Studies and Industry Patterns	63
Enterprise Knowledge Base Search	63
Code Documentation and Code Search	63
Customer Support and Helpdesk Automation	63
Legal Document Retrieval and Case Research	63
Medical and Scientific Literature Search	63
E-Commerce Product Search and Recommendations	64
Chapter 23: Future Directions and Open Problems	65
Long-Context Models and Their Impact on RAG	65
Multimodal RAG: Images, Audio, and Video in the Pipeline	65
Self-Correcting RAG and Agentic Retrieval Patterns	65
Learned Chunking and Neural Document Structure	65
The Future of Reranking: Are Cross-Encoders Obsolete?	65
Open Questions and Research Frontiers	66
Conclusion: Building RAG Systems That Work	67

The Retrieval Stack in One Picture	67
Decision Framework for Your RAG Architecture	67
Common Anti-Patterns to Avoid	67
How to Stay Current in a Fast-Moving Field	67
Final Thoughts on Engineering Excellence in RAG	67
References	69
Foundational Papers	69
Chunking and Document Processing	69
Query Processing and Expansion	69
Embedding and Reranking Models	69
Benchmarks and Evaluation	69
Infrastructure and Optimization	70
Security and Access Control	70
Technical Articles and Guides	70

Building Production-Grade Search Systems with Dense Retrieval and Neural Reranking

This book teaches you how to design, implement, and operate retrieval-augmented generation systems that work reliably in production. You will learn how embedding models represent semantic meaning as vectors, how different retrieval strategies exploit those representations, how reranking models refine candidate sets into high-quality context, and how to assemble these components into systems that are accurate, fast, and maintainable. The material assumes working knowledge of Python and basic machine learning concepts, and provides end-to-end code examples using real libraries, real models, and real architectural patterns.

Introduction: Why RAG Needs Better Retrieval

The first time you tried retrieval-augmented generation, it probably worked impressively. You uploaded a document, asked a question, and got a grounded answer that cited sources. Then you tried it on real data at real scale, and something broke. Answers became evasive. Citations went stale. Retrieval returned irrelevant chunks that pulled the generator off course. Users complained that the system knew the answer but could not find it.

This book exists to fix that gap between the demo and production.

RAG has become the standard pattern for giving large language models access to external knowledge. The idea is simple: retrieve relevant context, feed it to the model, generate an answer. But retrieval quality is the bottleneck. The generator can only work with what you give it. If retrieval returns three irrelevant chunks and one weakly relevant chunk, no prompt engineering trick will save you. The system will either hallucinate or refuse to answer.

Most RAG guides stop at the pipeline diagram. They show you how to load a document, split it into chunks, embed it, and query a vector database. They do not tell you why your chunks are too large, why your similarity metric is the wrong choice, why hybrid search matters for your use case, or why reranking would fix 80 percent of your retrieval failures. They treat retrieval as a commodity instead of the core engineering challenge it is.

This book treats retrieval as first-class infrastructure.

The central thesis is this: building effective RAG systems is not about choosing a single model. It is about understanding how embedding spaces represent semantic similarity, how different retrieval strategies exploit those representations, and how reranking refines candidate sets into high-quality context. The details matter. The size of your chunks matters. The parameters of your HNSW index matter. The order in which you assemble context into the prompt matters. Each decision compounds.

You will learn the mechanisms, not just the APIs.

We will start with the mathematics of similarity. You cannot design a retrieval system if you do not understand how vectors encode meaning and how distance functions measure relevance. We will walk through how cosine similarity, dot product, and Euclidean distance relate, when they diverge, and what each one implies for your retrieval quality.

We will dissect embedding architectures. You will see why bi-encoders are fast for indexing but lose fine-grained signal, why cross-encoders are accurate but too slow for candidate generation, and why late-interaction models like ColBERT bridge the gap. You will understand the training objectives: contrastive learning, triplet loss, hard negative mining. You will know what instruction-aware embeddings are and when they help.

We will spend more time on chunking than most books. Chunking is the most underappreciated decision in RAG. You will learn fixed-size chunking, sentence-based chunking, recursive splitting, sliding windows, semantic chunking, hierarchical parent-child strategies, structure-aware chunking, and query-aware chunking. For each, you will see how it works, where it fails, and what to measure when deciding. Recent empirical studies have overturned some assumptions about semantic chunking; we will cover those results honestly.

We will cover dense retrieval, sparse retrieval, and hybrid search. BM25 is not dead. Keyword search still beats semantic search for exact matches, rare terms, and identifiers. The best systems combine both and fuse the results with reciprocal rank fusion or weighted scoring. You will implement hybrid retrieval end to end.

We will give reranking its due. Reranking is not a nice-to-have. It is the cheapest way to dramatically improve retrieval quality. Cross-encoder rerankers evaluate query-document pairs jointly and recover relevance that bi-encoders discard during compression. We will cover off-the-shelf rerankers from Cohere, Jina, and BAAI, how to run them efficiently at scale, when to use prompt-based reranking, and how to fine-tune domain-specific rerankers.

We will build the full pipeline. You will see how documents flow from ingestion through parsing, normalization, chunking, embedding, indexing, retrieval, reranking, context assembly, prompt construction, and response generation. You will see how to design asynchronous pipelines, how to cache embeddings, how to handle incremental updates, how to enforce access control, and how to monitor everything in production.

We will measure rigorously. You will learn retrieval metrics: recall at K, precision at K, mean reciprocal rank, mean average precision, normalized discounted cumulative gain. You will learn RAG-specific metrics: faithfulness, groundedness, context precision, and answer relevance. We will cover automated evaluation with RAGAS, human evaluation design, and how to run A/B tests on retrieval quality.

We will not ignore the hard parts. We will address prompt injection attacks, retrieval poisoning, PII leakage, multi-tenant isolation, and the limitations of every technique we cover. We will show you where things go wrong and how to debug them systematically.

This book is written for practitioners. Every chapter includes production-ready Python code using established libraries: LangChain for pipeline orchestration, LlamaIndex for advanced retrieval patterns, sentence-transformers for embedding models, FAISS or Qdrant for vector indexing, rank_bm25 for sparse retrieval, and open-source rerankers from Hugging Face. The code is complete, documented, and designed to be extended rather than copied blindly.

You do not need to read this book linearly. Chapters on chunking, embedding models, and reranking are largely independent. If you are already comfortable with one area, skip ahead. The reference sections and the curated bibliography at the end will help you navigate to specific topics.

By the end, you will understand how to design retrieval that actually works: retrieval that finds what the generator needs, fast enough, cheap enough, reliably enough, and securely enough for production. Let us begin.

Chapter 1: The RAG Paradigm

The Retrieval-Then-Generate Architecture

The core idea of retrieval-augmented generation is to combine two distinct capabilities: the ability to look up relevant information from external sources, and the ability to generate coherent text conditioned on that information. Neither component is new. Information retrieval has been studied for decades. Sequence-to-sequence generation has been the foundation of modern language models. What is new is the systematic integration of both into a single pipeline, made practical by the emergence of powerful embedding models, efficient approximate nearest neighbor search, and large language models that can reliably use provided context.

The original formulation of RAG appeared in the 2020 paper by Patrick Lewis, Ethan Perez, and colleagues at Facebook AI Research and University College London [1]. Their work introduced a neural architecture that jointly trains a dense retriever and a seq2seq generator on knowledge-intensive tasks. The retriever maps queries to Wikipedia passages stored in a dense vector index. The generator conditions on both the query and the retrieved passages to produce answers. On open-domain question answering tasks, their RAG model established new state-of-the-art results and demonstrated that non-parametric memory (the retrieved documents) and parametric memory (the weights of the generator) are complementary rather than redundant.

The original RAG architecture was specific to open-domain question answering. Modern RAG, as used in enterprise knowledge bases, code search, customer support systems, and document assistants, has evolved substantially. The generator is no longer jointly trained with the retriever but is instead a pre-trained large language model accessed via API or self-hosted. The retriever is no longer limited to Wikipedia but operates on arbitrary corpora: internal documentation, databases, emails, chat logs, research papers, or product catalogs. The pipeline is modular rather than monolithic. Components are replaced independently as better models appear.

Despite these changes, the fundamental architecture remains unchanged. A RAG system takes a user query, retrieves a set of candidate documents or

document chunks, assembles those chunks into a prompt, feeds the prompt to a language model, and returns the model output. This retrieval-then-generate flow is the backbone of every production RAG system. Every design decision in this book revolves around improving one or more steps of this flow.

The beauty of the paradigm is its separation of concerns. The generator does not need to memorize facts. It only needs to understand context and produce coherent responses. The retriever does not need to generate text. It only needs to find relevant information quickly. This separation enables continuous updates to the knowledge base without retraining the generator. It enables domain-specific adaptation through curated documents rather than expensive fine-tuning. It enables control over what the system knows and does not know.

The weakness of the paradigm is that every component is necessary and every failure mode is fatal. If retrieval returns irrelevant context, the generator either hallucinates or refuses to answer. If the generator ignores the context, the system behaves like a regular chatbot despite all the retrieval machinery. If the context is truncated, malformed, or too large, the generator is overloaded and degrades. RAG systems are pipelines, and pipelines fail at their weakest link.

Understanding this architecture means understanding what each component promises and what it cannot deliver. The following sections map out the full data flow, contrast RAG with alternative approaches, and surface the pitfalls that engineers encounter when moving from prototypes to production.

Why Retrieval Quality Determines Output Quality

The generator is not omniscient. It has no independent access to the corpus. Its output is conditioned entirely on the prompt you give it. The prompt contains the user query, any system instructions, and the retrieved context. If the retrieved context does not contain the answer, the generator has two choices: guess based on parametric memory or say it does not know. Neither outcome is acceptable in production systems where accuracy matters.

This means retrieval quality is the primary determinant of answer quality. Better retrieval leads to better answers. Worse retrieval leads to worse answers. No prompt template, no chain-of-thought instruction, no temperature

adjustment can compensate for a retrieval system that consistently returns the wrong chunks.

Consider a concrete example. A customer asks, “What is the SLA for enterprise support?” A well-functioning RAG system retrieves the relevant support policy document, which states that enterprise customers receive 99.99 percent uptime guarantees and two-hour response times for critical incidents. The generator produces an accurate, sourced answer. A poorly-functioning system retrieves three generic marketing pages about product features and one outdated FAQ. The generator either fabricates an SLA or produces a vague response that frustrates the customer.

This example illustrates the basic dependency. It also illustrates subtler issues. What if the relevant document exists but is split across chunks in a way that no single chunk contains the complete answer? What if the query uses different terminology than the document (“SLA” versus “service level agreement” versus “uptime guarantee”)? What if the document is in a different language than the query? What if multiple documents contain conflicting information? All of these are retrieval problems, not generation problems.

The relationship between retrieval quality and output quality is not linear. There are diminishing returns at the top. Moving from a system that retrieves zero relevant chunks to one that retrieves one relevant chunk is transformative. Moving from three relevant chunks to four may have negligible impact. But below a certain threshold, the system fails entirely. The generator cannot extract truth from noise.

This threshold behavior explains why many RAG improvements appear to do nothing until they suddenly do. Adding reranking to a system that already retrieves good chunks may not change the answer. Adding reranking to a system that retrieves mediocre chunks often changes everything. The difference lies in whether the improvement crosses the threshold that lets the generator find signal.

You will see this pattern throughout the book. Chunking strategies, embedding models, similarity metrics, index parameters, hybrid search, query expansion, metadata filtering, reranking: each technique matters most when the baseline retrieval is poor enough that relevant chunks are present in the index but not consistently returned in the candidate set. Your job is to diagnose where retrieval is failing and apply the right intervention.

The Complete RAG Data Flow

A complete RAG system operates in two distinct phases: ingestion and query time. The ingestion phase transforms raw documents into a searchable index. The query phase retrieves from that index and generates answers. Both phases are critical, but they have very different characteristics and optimization goals.

The ingestion pipeline begins with document loading. Documents arrive in many formats: PDFs, Word files, HTML pages, plain text, code repositories, database dumps, structured JSON, emails, chat logs. Each format requires different parsing logic. PDFs may contain columns, headers, footers, tables, and images that must be disentangled. HTML may contain navigation elements and scripts that should be stripped. Code repositories may contain test files and build artifacts that should be excluded. The loading step extracts raw text and metadata from each document.

The next step is text normalization. The raw text is cleaned to remove artifacts, normalize whitespace, handle special characters, and standardize formatting. Aggressive normalization can destroy information (such as version numbers or identifiers). Insufficient normalization can introduce noise (such as excessive line breaks or encoding errors). The right level of normalization depends on the document type and the query patterns.

After normalization comes chunking. The document is split into smaller units that can be embedded and retrieved independently. The goal is to create chunks that are self-contained enough to be relevant on their own, small enough to fit within context windows alongside other chunks, and aligned with the semantic structure of the document. Chunking decisions are made once during ingestion but affect every query. They are among the most consequential and underexamined choices in RAG system design.

Each chunk is then embedded. An embedding model converts the chunk text into a dense vector representation. The vector encodes semantic meaning such that similar chunks produce similar vectors. The embedding model is typically a pre-trained transformer fine-tuned on similarity or retrieval tasks. It operates on a fixed maximum sequence length, which constrains the maximum chunk size. The output is a floating-point vector with hundreds or thousands of dimensions.

The vectors are stored in a vector database or index. The index supports approximate nearest neighbor search: given a query vector, return the stored

vectors most similar to it. The index is optimized for fast retrieval over millions or billions of vectors. Common algorithms include HNSW (Hierarchical Navigable Small World graphs), IVF (Inverted File with clustering), and PQ (Product Quantization). The index may also store metadata alongside each vector to support filtering and faceting.

At query time, the user submits a natural language query. The query is preprocessed: normalized, possibly expanded or reformulated. The query is then embedded using the same or a compatible embedding model. The query vector is searched against the index, returning a ranked list of candidate chunks. The candidates are scored by similarity and potentially filtered by metadata (such as document type, date, or access permissions).

In many systems, the initial retrieval is followed by reranking. A reranking model evaluates each candidate chunk jointly with the query and produces a more accurate relevance score. Reranking is computationally more expensive than vector search because it requires joint encoding rather than precomputed vectors. It is applied to a smaller set of candidates (typically 50 to 200) to keep latency manageable. The reranker reorders the candidates, promoting truly relevant chunks and demoting false positives.

The top-K reranked chunks are assembled into a prompt. The prompt includes system instructions, the user query, and the retrieved context with clear source attribution. Context window constraints may require truncation, summarization, or selective exclusion of some chunks. The prompt is sent to the generator, which produces the final answer.

The answer may be post-processed: citations are added, formatting is applied, sensitive information is redacted, and confidence scores are computed. The answer is returned to the user. The entire interaction is logged for monitoring, evaluation, and debugging.

This data flow is a simplification. Real systems add complexity: multi-stage retrieval with multiple indices, query classification for routing, caching for frequently asked questions, conversation history for context, and feedback loops for continuous improvement. But the core flow remains: load, normalize, chunk, embed, index, retrieve, rerank, assemble, generate.

RAG vs Finetuning vs Full Pretraining

RAG is one of three major approaches for giving models access to task-specific or domain-specific knowledge. The others are fine-tuning and full pre-training. Each approach has different strengths, costs, and limitations. Understanding the trade-offs helps you choose the right architecture for your use case.

Full pre-training is the process of training a model from random initialization on a massive corpus of text. It builds the model's parametric memory: its internalized knowledge of language, facts, reasoning patterns, and world models. Pre-training is extremely expensive. It requires vast datasets, thousands of GPUs, and weeks or months of computation. It produces general-purpose models that are then adapted for specific tasks. RAG, fine-tuning, and all downstream applications depend on models that have already undergone pre-training.

Fine-tuning is the process of continuing training on a smaller, domain-specific dataset to adapt the model's behavior. A model pre-trained on general web text can be fine-tuned on medical literature to answer medical questions, on code to generate code, or on customer support transcripts to handle support tickets. Fine-tuning modifies the model's parameters, changing what the model knows and how it reasons. It is cheaper than pre-training but still non-trivial. It requires labeled data, compute resources, and careful validation to avoid overfitting or catastrophic forgetting.

RAG, by contrast, does not modify model parameters. It gives the model access to external knowledge at inference time through retrieval. The knowledge base can be updated instantly by adding or removing documents. Different users can see different knowledge bases through access control. The system can cite sources and show its reasoning. RAG does not require labeled training data or compute-intensive training runs. It requires engineering effort in the retrieval pipeline.

The key trade-off is between implicit knowledge (stored in model weights) and explicit knowledge (stored in documents). Fine-tuning is better when the desired behavior is subtle, contextual, and hard to articulate in documents. It is ideal for style transfer, instruction following, domain-specific reasoning, and tasks where the training data reflects patterns rather than facts. RAG is better when the knowledge is explicit, factual, frequently changing, or requires cita-

tions. It is ideal for knowledge bases, document search, regulations, product documentation, and any domain where accuracy and traceability matter.

Many production systems combine both approaches. A model is fine-tuned on domain data to learn how to reason about that domain and follow domain-specific instructions. It is then augmented with RAG to access the latest documents and provide sourced answers. The fine-tuned model handles the reasoning and formatting; the retrieval pipeline supplies the facts. This hybrid approach is often more effective than either alone.

Full pre-training is rarely a practical option for individual organizations. It is done by well-funded labs and companies to produce the base models that everyone else uses. Fine-tuning and RAG are the primary tools available to practitioners. Choosing between them depends on the nature of the knowledge you want the system to use, how frequently that knowledge changes, whether you need citations, and your available resources.

A common mistake is to use fine-tuning when RAG is more appropriate, or vice versa. Organizations fine-tune models on documents that are updated weekly, incurring retraining costs every time the documents change. Organizations use RAG for tasks where retrieval is unnecessary and the model already knows the answer from pre-training. The rule of thumb: if the knowledge is static, implicit, or behavioral, fine-tune. If the knowledge is explicit, dynamic, or sourced, use RAG. If you are unsure, start with RAG because it is cheaper to change your retrieval system than your model weights.

Common Pitfalls and Misconceptions

Early-stage RAG prototypes rarely survive production without significant rework. The gap between prototype and production is not a technology gap. It is an engineering gap. The same fundamental approach works; the implementation must be hardened.

The most common pitfall is ignoring chunking. Engineers default to whatever chunking strategy their framework provides, often fixed-size chunks of 512 or 1024 tokens with no overlap. They never evaluate whether chunks align with semantic boundaries. They never test whether queries can find the relevant information within individual chunks. They assume chunking is a preprocessing detail rather than a retrieval-critical decision. Chapter 6 and 7 cover this in depth.

Another pitfall is treating all embedding models as interchangeable. Engineers download a popular model from Hugging Face and never question its suitability. They do not benchmark it on their data. They do not consider multilingual requirements, domain-specific vocabulary, or instruction-aware training. They expect the model to work because it worked on MTEB or BEIR. Chapter 3 and Chapter 16 address model selection.

Many systems neglect reranking entirely. Engineers build vector search and assume similarity is enough. They do not realize that bi-encoders discard fine-grained relevance signal during compression. They do not measure the precision of their top-5 results. When users complain about irrelevant answers, they adjust the prompt instead of improving retrieval. Chapter 12 and 13 explain reranking and how to integrate it.

Hybrid search is frequently overlooked. Engineers implement vector search and assume it handles all queries. They are surprised when exact matches, rare terms, and identifiers fail to retrieve. They do not know that BM25 often beats dense retrieval on lexical queries. They do not implement reciprocal rank fusion or weighted score combination. Chapter 4 covers hybrid search comprehensively.

Evaluation is often treated as optional. Engineers validate their RAG system with hand-crafted queries that they know work. They do not build systematic evaluation datasets with ground truth. They do not measure recall, precision, or faithfulness. They do not run A/B tests when they change a model or a chunking strategy. They ship blindly. Chapter 16 and 17 cover evaluation rigorously.

Production concerns are left for later. Engineers prototype with everything in a single process: embedding, indexing, retrieval, generation all running in one Python script. They do not consider caching, rate limiting, access control, monitoring, or error handling. They do not design for incremental updates. When traffic scales, the system fails. Chapter 15 and 20 cover production architecture.

Finally, many engineers over-rely on frameworks. They use LangChain or LlamaIndex without understanding what the abstractions are doing. They cannot debug a retrieval failure because they do not know how their chunking, embedding, or indexing configuration works at the implementation level. Frameworks accelerate development but they must not obscure understanding.

This book is structured to help you avoid these pitfalls. Each chapter explains the mechanisms beneath the abstractions. Each code example shows complete implementations, not just framework calls. Each design recommendation includes trade-offs and failure modes. The goal is not to teach you how to use a tool but how to design a system that works.

What This Book Will Build

Throughout this book, you will see working code that builds real RAG systems. The code uses established libraries and models that are available today. You will be able to run it, modify it, and extend it for your own use cases.

We will build a basic RAG system: load documents from disk, chunk them with recursive text splitting, embed them with a BGE model, index them in FAISS or Qdrant, retrieve candidates with cosine similarity, rerank them with a cross-encoder, and generate answers with an LLM. This basic system will serve as a foundation.

We will extend it with hybrid search: adding BM25 retrieval alongside vector search and combining results with reciprocal rank fusion. We will add metadata filtering to support access control and faceted search. We will implement multi-query retrieval and query expansion techniques like HyDE and StepBack prompting. We will add caching layers and asynchronous processing for production throughput.

We will implement advanced chunking strategies: hierarchical parent-child parsing, semantic chunking with boundary detection, structure-aware chunking for code and markdown, and sentence window retrieval. We will benchmark these strategies and discuss when each is appropriate.

We will evaluate the system rigorously: building evaluation datasets, computing retrieval metrics, measuring faithfulness and groundedness with RAGAS, and running A/B tests on model and chunking changes. We will show how to set up automated evaluation as part of your CI/CD pipeline.

We will harden the system for production: implementing error handling, observability, rate limiting, versioning, and rollback strategies. We will address security: PII detection, prompt injection defense, and document-level access control with RBAC. We will show deployment patterns: containerization, model serving with Triton or vLLM, and scalable architectures for high-traffic applications.

You do not need to build every variant we discuss. The goal is to give you a toolbox of patterns and the understanding to choose the right one for your domain. By the end of this book, you will know how to design a RAG system that is not just functional but excellent.

Chapter 2: Vector Representations and Similarity

From Tokens to Vectors: The Basic Idea

Natural language is discrete and symbolic. Words are symbols drawn from a vocabulary, sentences are sequences of symbols, and documents are sequences of sentences. Language models operate on this symbolic structure. Embeddings change the representation from symbolic to geometric.

An embedding is a mapping from discrete tokens or sequences to continuous vectors in a high-dimensional space. Each token or sequence is assigned a vector with hundreds or thousands of dimensions. The dimensions have no intrinsic meaning like height or weight. They are learned coordinates that capture statistical relationships in the training data. The geometry of the embedding space encodes semantic information: similar concepts have similar vectors, and dissimilar concepts have dissimilar vectors.

The intuition is simple but powerful. If you can represent text as vectors, you can use geometric operations to measure similarity. If the embedding model has learned meaningful representations, then vectors that are close together in the space correspond to text that is semantically similar. Retrieval becomes a nearest neighbor search: given a query vector, find the document vectors closest to it.

Modern embedding models are built on transformer encoders. The encoder takes tokenized text as input and produces a representation for each token. A pooling operation then aggregates token representations into a single sequence-level vector. The choice of encoder architecture and pooling strategy affects the quality and characteristics of the embeddings, but the basic pipeline is consistent across most models.

Consider a concrete example. Suppose you have three document chunks:

- Chunk A: “The quarterly earnings report showed a 12 percent revenue increase.”

- Chunk B: “Sales grew by 12 percent in the latest fiscal quarter.”
- Chunk C: “The company announced a new marketing strategy for social media.”

A good embedding model will produce vectors such that A and B are close together (similar meaning about revenue growth) while C is farther away (different topic). A query like “How much did sales increase last quarter?” will produce a query vector close to A and B. Retrieval returns A and B as the top results.

This works because the embedding model has learned that “quarterly earnings report,” “fiscal quarter,” “revenue increase,” and “sales grew” are related concepts. It has learned that “marketing strategy” and “social media” are a different conceptual cluster. The learning happens during training on large datasets of text, where the model is optimized to produce representations that satisfy a similarity objective.

The embedding model does not understand language in the human sense. It has not learned that revenue and sales are synonyms through linguistic rules. It has learned them as statistical regularities: documents containing “revenue” often contain related terms, documents containing “sales” contain similar terms, and queries about financial performance are semantically closer to documents about financial performance than to documents about marketing.

This statistical learning is sufficient for most retrieval tasks. It is also the source of limitations. Embeddings can fail on rare terms, domain-specific jargon, exact matches, and queries where the difference between relevant and irrelevant documents is subtle. Understanding both the strengths and limitations of this approach is essential for designing retrieval systems that work.

Embedding Spaces and Semantic Geometry

An embedding space is a vector space where each point corresponds to a text representation. The space is high-dimensional: common embedding models produce vectors with 384 to 1536 dimensions for smaller models, and up to 4096 dimensions for larger models. The dimensions are continuous floating-point numbers. The space is learned: the coordinates are not hand-designed but emerge from training.

The fundamental assumption of embedding-based retrieval is that semantic similarity is approximated by geometric proximity. If two texts have similar meaning, their vectors should be close together. This assumption is not universally true, but it is approximately true enough to be useful across a wide range of tasks. The quality of the approximation depends on the embedding model and the task domain.

Visualizing high-dimensional spaces is impossible for humans, but we can reason about their properties through projections and analogies. In a well-structured embedding space, you might observe clusters of vectors corresponding to topics. Financial documents cluster together. Medical documents cluster together. Code snippets in the same programming language cluster together. Within these topic clusters, finer-grained sub-clusters may appear: Python code vs JavaScript code, cardiology vs oncology, accounting vs investing.

The geometry is not perfectly clean. There is noise, overlap, and outliers. Two documents about the same topic may be far apart if they use different terminology or writing styles. Two documents about different topics may be close if they share vocabulary. The embedding model smooths over these variations based on patterns in its training data, but it cannot eliminate them entirely.

One important property of most modern embedding spaces is anisotropy: the embeddings do not use the full space uniformly. Research on sentence embeddings has shown that many models produce vectors that lie in a narrow cone of the space rather than being distributed uniformly across all directions [2]. This anisotropy can degrade similarity estimation because most vectors point in roughly the same direction, reducing the discriminative power of cosine similarity. Some training techniques explicitly regularize embeddings to be more isotropic, spreading them more evenly across the space.

Another property is dimensionality. High-dimensional spaces have counterintuitive geometric properties. In high dimensions, most points are far from each other, and the difference between the nearest neighbor and the farthest neighbor shrinks as a fraction of the total range. This is sometimes called the “curse of dimensionality” for similarity search. In practice, embeddings with hundreds of dimensions work well because the meaningful signal is concentrated in a lower-dimensional manifold within the high-dimensional space. Approximate nearest neighbor algorithms like HNSW are designed to navigate this geometry efficiently.

The embedding space also inherits biases and gaps from the training data. If the training data is predominantly English, English concepts are better represented than concepts in other languages. If the training data is from the web, popular topics are better represented than niche topics. If the training data is from before a certain date, recent events are not well represented. These biases affect retrieval quality in domain-specific applications.

Understanding the embedding space means understanding what it can and cannot represent. It can represent broad semantic similarity very well. It can represent topic clustering reasonably well. It is less good at representing exact lexical matches, rare terms, fine-grained distinctions within a narrow domain, and temporal freshness. Hybrid retrieval, reranking, and domain-specific fine-tuning are all techniques that compensate for these limitations.

Cosine Similarity, Dot Product, and Euclidean Distance

Once texts are embedded as vectors, retrieval requires measuring similarity between the query vector and each document vector. Three similarity metrics dominate: cosine similarity, dot product, and Euclidean distance. Each has different mathematical properties, different computational costs, and different implications for retrieval quality.

Cosine similarity measures the cosine of the angle between two vectors. It is defined as the dot product of the vectors divided by the product of their magnitudes:

$$1 \quad \text{cosine_similarity}(a, b) = (a \cdot b) / (||a|| * ||b||)$$

The result ranges from -1 (opposite directions) to +1 (same direction). Cosine similarity is magnitude-invariant: it only considers direction, not how large the vectors are. This is desirable for text embeddings because the magnitude of an embedding vector does not carry meaningful semantic information. A longer document does not produce a more meaningful embedding simply because its vector has larger magnitude.

Dot product is the sum of element-wise products:

```
1 dot_product(a, b) = sum(a_i * b_i for all i)
```

Dot product is not magnitude-invariant. Two vectors in the same direction but different magnitudes produce different dot products. This can be problematic for text embeddings where magnitude is arbitrary. However, if vectors are normalized to unit length (magnitude equals 1), dot product becomes identical to cosine similarity. Many modern embedding models output normalized vectors specifically to allow dot product as the similarity metric because it is computationally cheaper: it avoids the square root and division operations required for cosine similarity.

Euclidean distance measures the straight-line distance between two vectors:

```
1 euclidean_distance(a, b) = sqrt(sum((a_i - b_i)^2 for all i))
```

Euclidean distance is a distance metric, not a similarity metric: lower values indicate greater similarity. It is sensitive to both direction and magnitude. Two vectors with similar direction but different magnitudes have larger Euclidean distance than two vectors with similar direction and similar magnitudes. For text embeddings, this sensitivity to magnitude is usually undesirable, so Euclidean distance is less commonly used.

The practical choice between these metrics depends on your embedding model. The rule is: use the metric the model was trained with. If the model was trained to optimize cosine similarity, evaluate with cosine similarity. If the model normalizes vectors during inference and was trained with dot product in mind, use dot product. Using the wrong metric degrades retrieval quality because the embedding space geometry is optimized for the training objective.

Most major embedding models normalize their outputs and are compatible with cosine similarity or dot product (they are equivalent when vectors are normalized). OpenAI's embedding models, for example, return normalized vectors and recommend cosine similarity. Sentence-transformers models typically normalize their outputs. Models that do not normalize often still work with cosine similarity because it is robust to magnitude differences.

Vector databases default to cosine similarity because it is the safest choice across models. Pinecone, Qdrant, Weaviate, and Chroma all use cosine as the default distance metric. Some databases allow you to choose: pgvector

supports separate index operators for L2 distance, dot product, and cosine similarity.

The computational difference between these metrics matters at scale. Dot product is the cheapest: it requires only multiplication and addition. Cosine similarity adds magnitude computation (square root of sum of squares) and division. Euclidean distance adds subtraction and square root. When comparing millions of vectors per query, even small per-vector cost differences compound. If your vectors are normalized, use dot product. If not, use cosine similarity.

A common mistake is to mix metrics unintentionally. An embedding model trained with cosine similarity is used with Euclidean distance because that was the vector database default. Retrieval quality drops silently. Always verify that your similarity metric matches your model's training objective.

Normalization and Its Effects on Similarity

Vector normalization is the process of scaling each vector so that its magnitude (L2 norm) equals 1. A normalized vector lies on the surface of a unit sphere. Normalization changes the relationship between different similarity metrics and affects how the embedding space behaves.

When vectors are normalized, dot product equals cosine similarity. This is not an approximation. It is an identity. The normalization eliminates the magnitude term in the cosine formula, leaving only the dot product. This equivalence is exploited by most modern systems because dot product is faster to compute.

Normalization also affects Euclidean distance. For normalized vectors, Euclidean distance is monotonically related to cosine similarity. Specifically:

```
1 euclidean_distance(a, b) = sqrt(2 - 2 * cosine_similarity(a, b))
```

This means that for normalized vectors, ranking by Euclidean distance produces the same order as ranking by cosine similarity. The ranking is what matters for retrieval, so the three metrics become effectively equivalent for normalized vectors.

Many embedding models normalize their outputs during inference. Sentence-transformers models apply L2 normalization by default. OpenAI's

models normalize internally. BGE models normalize their embeddings. When using these models, you can safely use dot product, cosine similarity, or Euclidean distance for ranking. They will produce the same ranked order.

However, normalization is not free. It constrains the embedding space. When every vector lies on a unit sphere, the model cannot use magnitude to encode information. Some models might want to use magnitude to represent confidence, length, or importance, but normalization discards that signal. For most retrieval tasks, this trade-off is acceptable because direction is more informative than magnitude for text similarity.

Normalization also interacts with training objectives. A model trained with contrastive loss on normalized embeddings learns to push similar pairs closer in angular distance and dissimilar pairs farther apart on the sphere. If you then evaluate with unnormalized vectors and a magnitude-sensitive metric, the learned geometry may not translate well. Consistency between training, normalization, and evaluation is essential.

Vector databases often store normalized vectors. Qdrant and Pinecone automatically normalize vectors on insertion if you select cosine or dot product distance. FAISS has built-in normalization utilities. Some systems normalize at query time rather than storage time to save disk space, normalizing the query vector before searching an unnormalized index.

A subtle point: normalization can sometimes hurt performance. In rare cases where magnitude carries meaningful signal (such as in some image embeddings or specialized models), forcing all vectors onto the unit sphere discards useful information. Always check the model documentation and benchmark with and without normalization for your specific use case.

For text embeddings in RAG systems, normalization is standard practice. It simplifies the choice of similarity metric, improves stability, and aligns with how most embedding models are trained. Unless you have a specific reason to do otherwise, normalize your vectors.

High-Dimensional Geometry: What Goes Wrong at Large Dimensions

Embedding spaces have hundreds or thousands of dimensions. Human intuition about geometry is calibrated for two or three dimensions. In high

dimensions, geometry behaves strangely. Understanding these effects helps explain why approximate nearest neighbor search is necessary and why retrieval quality degrades under certain conditions.

The first counterintuitive effect is concentration of distances. In high-dimensional spaces, the ratio between the nearest and farthest distances to a query point tends toward 1 as dimensionality increases. This means that all points become roughly equidistant, and the concept of “nearest” loses discriminative power. This is the essence of the curse of dimensionality.

In practice, this effect is mitigated because embeddings do not uniformly fill the space. They lie on lower-dimensional manifolds where meaningful variation exists. The effective dimensionality of the embedding space is often much lower than the nominal dimensionality. A 768-dimensional embedding might have an effective dimensionality of 100 to 200 because most variation is captured along a smaller number of directions.

The second effect is sparsity. High-dimensional spaces are vast. Even millions of vectors occupy a tiny fraction of the available volume. Most of the space is empty. This sparsity makes exact nearest neighbor search expensive because you must examine many vectors to find the nearest one. It also makes approximate nearest neighbor algorithms like HNSW effective: the graph structure allows you to navigate through the empty space and converge on dense regions where neighbors are likely to be.

The third effect is angular concentration. In high dimensions, random vectors are nearly orthogonal. The expected cosine similarity between two random high-dimensional vectors approaches zero. This means that embeddings with cosine similarity close to zero are not necessarily dissimilar in meaning; they might just be random. Meaningful semantic similarity requires cosine similarity significantly above zero for most models. A cosine similarity of 0.8 between two text embeddings is genuinely high. A similarity of 0.3 might indicate some overlap but is not strong evidence of relevance.

Understanding this helps with result thresholding. If you retrieve candidates with cosine similarity below some threshold, you are likely getting noise. Many systems apply a minimum similarity threshold before passing chunks to the generator. The threshold depends on the model and task but is typically in the range of 0.5 to 0.7 for cosine similarity. Below that threshold, the retrieved chunks are unlikely to be useful.

The fourth effect is the difficulty of distance-based clustering. In high

dimensions, distance-based clustering algorithms struggle because distances become less discriminative. Semantic chunking methods that rely on clustering or distance thresholds between consecutive chunks must be carefully calibrated. This is one reason why fixed-size chunking often performs as well as or better than semantic chunking on real-world documents.

High-dimensional geometry also affects index design. HNSW, IVF, and PQ are all designed to handle high-dimensional spaces efficiently. They make trade-offs between recall and speed based on assumptions about the geometry. Understanding these assumptions helps you tune parameters for your specific workload.

For RAG practitioners, the key takeaways are: high-dimensional spaces are sparse and distances concentrate, approximate search is necessary and correct, meaningful similarity requires thresholds significantly above random, and your intuition about geometry may be misleading. Trust measurements over intuition.

Choosing a Similarity Metric for Your Use Case

Selecting a similarity metric is rarely an open-ended choice. The embedding model constrains your options. The practical decision is: verify that your metric matches your model, then optimize for performance.

For most modern text embedding models, the answer is straightforward. Use the metric specified in the model documentation. If the model normalizes embeddings, dot product is safe and fastest. Cosine similarity is also safe and equivalent. If the model does not normalize, use cosine similarity because it is robust to magnitude differences.

If you are building a system from scratch and choosing both model and metric, follow this process:

1. Select an embedding model based on quality benchmarks (MTEB, BEIR), latency requirements, and deployment constraints.
2. Check the model documentation for the recommended similarity metric.
3. Verify the metric experimentally on your data. Compute retrieval metrics (recall@K, MRR) for cosine similarity, dot product, and Euclidean distance. Use the best-performing one.

4. Once selected, lock in the metric. Ensure the embedding model, vector database, and application code all use the same metric consistently.
5. If you change the embedding model, re-verify the metric.

Do not choose a metric based on mathematical elegance. Choose based on empirical performance on your data. The embedding model defines the space; the metric only measures it. If the model's geometry aligns with cosine similarity, no amount of metric engineering will make Euclidean distance perform better.

For vector databases, configure the distance metric at index creation time. Most databases require you to specify the metric when creating the index. Changing the metric later requires rebuilding the index. Plan ahead.

For hybrid search combining multiple retrieval methods, different components may use different scoring methods (BM25 scores, cosine similarity, reranker scores). These scores must be combined using a fusion strategy like reciprocal rank fusion or normalized weighted scoring. RRF is preferred because it operates on ranks rather than raw scores, avoiding scale mismatch issues. Chapter 4 covers fusion in detail.

The choice of similarity metric is foundational but not magical. It is one component of retrieval quality. Better embedding models, better chunking, better query processing, and reranking all have larger impacts than the choice between cosine similarity and dot product. Make the right choice, but do not obsess over it at the expense of more consequential decisions.

Chapter 3: Embedding Model Architectures

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Transformer Encoders as Embedding Backbones

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Bi-Encoder vs Cross-Encoder Architectures

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Late-Interaction Models and ColBERT

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Training Objectives: Contrastive Learning, Triplets, and Hard Negatives

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Mean-Pooling, CLS Tokens, and Output Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Instruction-Aware and Prompt-Based Embeddings

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chapter 4: Dense Retrieval vs Sparse Retrieval

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

TF-IDF and BM25: Sparse Retrieval Fundamentals

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

How Dense Retrieval Works Under the Hood

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

The Lexical vs Semantic Gap

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

When Sparse Beats Dense (and Vice Versa)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Hybrid Search: Combining Dense and Sparse Signals

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Reciprocal Rank Fusion and Other Fusion Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chapter 5: Document Preprocessing and Text Normalization

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Document Parsing: PDFs, HTML, Office Files, and Code

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Text Cleaning and Normalization Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Handling Special Characters, Numbers, and Dates

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Language Detection and Multilingual Preprocessing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Metadata Extraction and Structuring

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Common Preprocessing Mistakes That Hurt Retrieval

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chapter 6: Context Chunking Strategies — Part I: Foundational Methods

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Why Chunking Is The Most Underappreciated RAG Decision

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Fixed-Size Chunking: The Baseline Everyone Starts With

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Sentence-Based and Paragraph-Based Chunking

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Sliding-Window Chunking and Overlap Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Recursive Character Text Splitting

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Choosing Chunk Size and Overlap: Evidence-Based Guidance

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chapter 7: Context Chunking

Strategies — Part II: Advanced Methods

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Semantic Chunking: Cluster-Based and Boundary Detection Methods

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Hierarchical Chunking and Parent-Document Retrieval

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Structure-Aware Chunking for Markdown, HTML, and Code

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Query-Aware and Dynamic Chunking

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Long-Context Chunking for Large Language Models

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Evaluating Chunking Quality: Metrics and Methods

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chapter 8: Vector Indexing and Approximate Nearest Neighbor Search

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Exact Nearest Neighbor vs Approximate Nearest Neighbor

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

HNSW: The Workhorse of Modern Vector Search

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

IVF, Product Quantization, and Other Indexing Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Index Build Time vs Query Latency Trade-Offs

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Choosing m, ef_construction, ef_search: Practical Tuning

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Vector Database Selection: Pinecone, Weaviate, Qdrant, Milvus, and More

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chapter 9: Building the Ingestion Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Pipeline Architecture: Batch vs Streaming vs Hybrid

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Document Loading and Format Handling

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chunking, Embedding, and Indexing as Distinct Stages

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Error Handling and Dead Letter Queues

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Incremental Updates and Document Versioning

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Scalable Ingestion Patterns for Large Corpora

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chapter 10: Query Processing and Expansion

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Understanding the Query: Intent, Ambiguity, and Reformulation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Query Embedding with the Same or Different Models

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Query Expansion: HyDE, StepBack, and Similar Techniques

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Multi-Query Retrieval and Ensemble Queries

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Query Classification and Routing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Handling Multi-Turn Conversational Queries

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chapter 11: Candidate Retrieval Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Single-Stage Dense Retrieval

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Hybrid Retrieval: Dense Plus Sparse Combined

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Multi-Vector Retrieval and ColBERT-Style Approaches

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Metadata Filtering and Faceted Search

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Semantic Search with Numerical and Temporal Filters

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Controlling Recall: How Many Candidates to Retrieve

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chapter 12: Reranking Models: Theory and Architectures

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Why Reranking Improves Retrieval Quality

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Cross-Encoder Reranking: Architecture and Mechanics

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Bi-Encoder vs Cross-Encoder vs Late-Interaction for Reranking

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Training Rerankers: Pairwise, Listwise, and Pointwise Methods

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Domain Adaptation and Fine-Tuning Rerankers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Latency vs Accuracy: When Is Reranking Worth It

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chapter 13: Reranking Model Implementations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Using Off-the-Shelf Rerankers (BGE, Cohere, Jina, E5)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Batch vs Streaming Reranking Architectures

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Prompt-Based Reranking and LLM Reranking

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Adaptive and Conditional Reranking

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Cascaded and Multi-Stage Reranking Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Cost Optimization for Reranking at Scale

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chapter 14: Context Assembly and Prompt Construction

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Selecting Which Chunks to Include

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Ordering Chunks: Recency, Relevance, and Narrative Flow

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Context Window Budgeting and Truncation Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Adding Source Attribution and Chunk Metadata

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Dynamic Context Selection Based on Query Complexity

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Prompt Templates That Make Retrieved Context Work

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chapter 15: End-to-End RAG System Design

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

The Complete RAG Architecture: All Components Together

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Service Boundaries and API Design

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Synchronous vs Asynchronous Request Flow

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Caching Strategies for Embeddings and Retrieval Results

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Rate Limiting and Backpressure Handling

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Multi-Tenant and Multi-Index Architectures

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chapter 16: Evaluating Embedding and Reranking Models

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandreringinretrieval-augmentedgeneration>.

Retrieval Metrics: Recall@K, Precision@K, MRR, MAP, NDCG

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandreringinretrieval-augmentedgeneration>.

Building Evaluation Datasets: Queries and Ground Truth

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandreringinretrieval-augmentedgeneration>.

Model Selection Through Systematic Benchmarking

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandreringinretrieval-augmentedgeneration>.

Evaluating Without Ground Truth: Heuristics and Proxies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandreringinretrieval-augmentedgeneration>.

A/B Testing Retrieval Quality in Production

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandreringinretrieval-augmentedgeneration>.

Ablation Studies and Controlled Experiments

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandreringinretrieval-augmentedgeneration>.

Chapter 17: Evaluating End-to-End RAG Performance

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Answer Correctness and Factual Accuracy

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Faithfulness and Groundedness: Did the Answer Use the Context?

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Context Utilization and Relevance Scores

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Latency, Throughput, and Cost as Quality Metrics

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Automated Evaluation Frameworks (RAGAS, DeepEval, etc.)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Human Evaluation Design and Execution

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chapter 18: Advanced Optimization Techniques

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Optimizing Embedding Dimensionality and Model Size

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Quantization and Compression for Embeddings

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

GPU Acceleration and Batch Inference Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Distributed Vector Indexing and Shard Management

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Memory-Optimized Retrieval Architectures

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Throughput and Latency Optimization Checklist

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chapter 19: Debugging and Troubleshooting RAG Systems

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

The RAG Debugging Playbook: Where to Start

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Diagnosing Bad Retrieval: Query, Index, or Model?

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Identifying Chunking Failures and Context Gaps

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Spotting Reranking Degradation and Cascading Errors

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Hallucination Diagnosis and Mitigation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Logging, Tracing, and Observability Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chapter 20: Production Deployment and Operations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Infrastructure and Containerization

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Model Serving: Triton, vLLM, and Custom Endpoints

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Monitoring: Metrics, Alerts, and Dashboards

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Versioning Models, Embeddings, and Chunking Logic

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Rollback Strategies and Canary Releases

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Disaster Recovery and Index Backups

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chapter 21: Security, Privacy, and Access Control

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Access Control and Document-Level Permissions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

PII Detection and Redaction in RAG Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Prompt Injection and Retrieval Poisoning Attacks

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Data Isolation in Multi-Tenant Systems

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Audit Logging and Compliance Requirements

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Securing Embeddings Against Reconstruction Attacks

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chapter 22: Case Studies and Industry Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Enterprise Knowledge Base Search

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Code Documentation and Code Search

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Customer Support and Helpdesk Automation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Legal Document Retrieval and Case Research

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Medical and Scientific Literature Search

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

E-Commerce Product Search and Recommendations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chapter 23: Future Directions and Open Problems

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Long-Context Models and Their Impact on RAG

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Multimodal RAG: Images, Audio, and Video in the Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Self-Correcting RAG and Agentic Retrieval Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Learned Chunking and Neural Document Structure

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

The Future of Reranking: Are Cross-Encoders Obsolete?

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandreringinretrieval-augmentedgeneration>.

Open Questions and Research Frontiers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandreringinretrieval-augmentedgeneration>.

Conclusion: Building RAG Systems That Work

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

The Retrieval Stack in One Picture

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Decision Framework for Your RAG Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Common Anti-Patterns to Avoid

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

How to Stay Current in a Fast-Moving Field

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Final Thoughts on Engineering Excellence in RAG

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

References

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Foundational Papers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Chunking and Document Processing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Query Processing and Expansion

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Embedding and Reranking Models

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Benchmarks and Evaluation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Infrastructure and Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Security and Access Control

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.

Technical Articles and Guides

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/embeddingsandrerankinginretrieval-augmentedgeneration>.