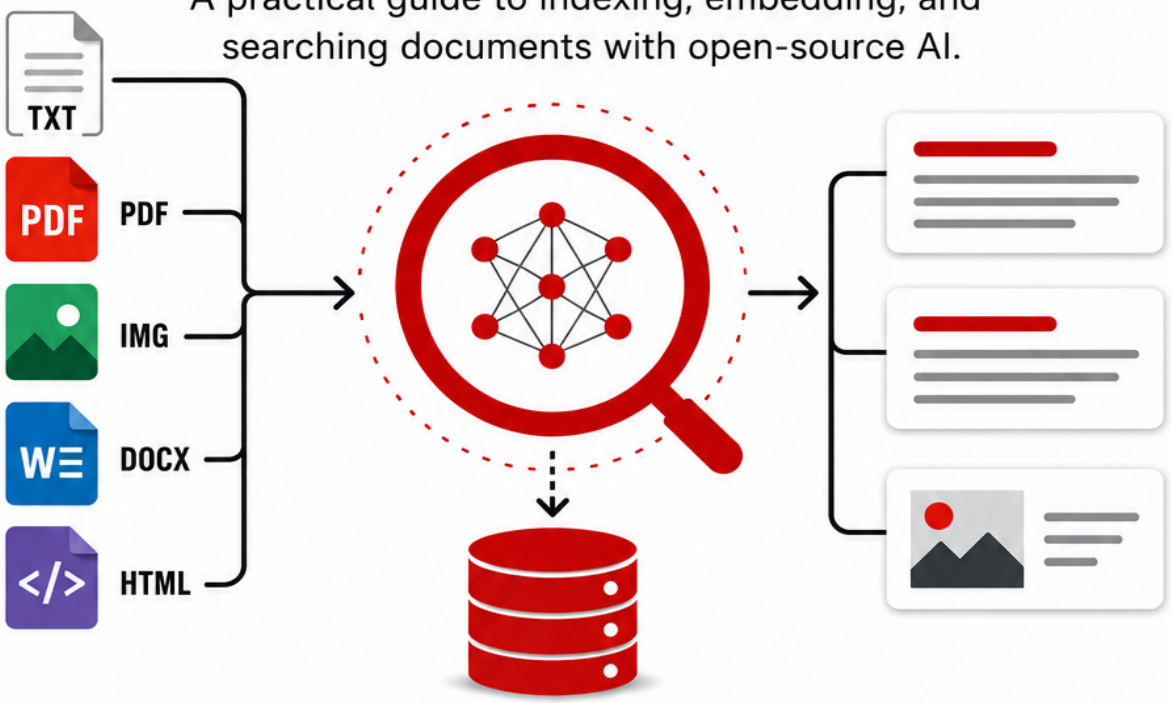
LOCAL AI SEARCH ENGINE

BUILDING INTELLIGENT DOCUMENT RETRIEVAL SYSTEMS FROM SCRATCH

A practical guide to indexing, embedding, and searching documents with open-source AI.



OPEN SOURCE | HYBRID SEARCH | LOCAL RAG | OCR & MULTIMODAL | SECURE & PRIVATE | CROSS-PLATFORM DEPLOYMENT

JAMES RUFUS

Local AI Search Engine

Building Intelligent Document Retrieval Systems from Scratch

Steve Publications

This book is available at https://leanpub.com/localaisearchengine

This version was published on 2026-07-20

Contents

Building Intelligent Document Retrieval Systems from Scratch

A practical guide to indexing, embedding, and searching documents with open-source AI.

This book teaches you how to build a complete, production-grade local search engine that indexes diverse document types, performs hybrid keyword and semantic retrieval, powers retrieval-augmented generation with local language models, and runs entirely on your own hardware. You will move from a minimal working prototype to a feature-rich system with OCR, multimodal embeddings, incremental indexing, caching, security, and cross-platform deployment. Every code example is complete, runnable Python built on current stable releases of open-source libraries including FastAPI, FAISS, Qdrant, Chroma, sentence-transformers, Ollama, and more.

Introduction: Why Build Your Own Search Engine?

You have documents. Hundreds, thousands, maybe tens of thousands of them. PDF reports, Word contracts, spreadsheet exports, presentation decks, scanned invoices, screenshots, code files, and notes scattered across directories on your machine or stored in a shared drive. You need to find things inside these documents quickly and accurately.

The obvious answer is Google, right? Or Microsoft Copilot, or Perplexity, or one of the many AI search engines that have proliferated over the past few years. But there is a reason none of those options work for your internal company documents, your legal case files, your research notes, or your family's scanned photographs. Those services require you to ship your data to someone else's servers, trust someone else's privacy policy, and pay someone else's pricing model.

The local AI search engine solves this problem. It runs entirely on your own hardware, never sends your documents across the network, and gives you complete control over every layer of the stack from document parsing to result ranking. While it requires upfront hardware investment and ongoing engineering attention, it eliminates per-query API costs and vendor lock-in. This book shows you how to build one from scratch.

The Problem with Cloud Search

Cloud AI search engines have made remarkable progress, but they come with fundamental limitations for anyone who needs to search their own private documents:

Privacy and data residency. Every time you upload a document to a cloud search service, that document leaves your control. Even if the provider promises not to use your data for training, you are trusting a third party's security posture and legal compliance. For healthcare, legal, financial, or government work, this trust is often insufficient or even prohibited by regulation.

API costs at scale. Embedding one million documents through OpenAI's text-embedding-3-large model costs approximately $0.13 per million tokens for embedding plus ongoing storage costs around $50 per month. For a growing corpus, these costs compound quickly, and you are locked into the provider's pricing changes.

Vendor lock-in. When your search infrastructure lives in someone else's platform, migrating becomes painful. Your embeddings, your index structure, and your query patterns are all tied to their proprietary formats.

Latency. Even a fast cloud API adds network round-trip time to every query. For interactive search experiences, this latency is noticeable and frustrating.

What Changed in 2024-2026?

Three developments converged to make local AI search practical for the first time:

Open embedding models reached production quality. Models like BGE-M3, Nomic Embed, and E5 now score competitively on the Massive Text Embedding Benchmark (MTEB), matching or exceeding commercial alternatives. The BGE-M3 model achieves an MTEB score of 63.0 with only 568 million parameters and a permissive MIT license, running comfortably on consumer hardware.

Local LLM inference became fast and accessible. Frameworks like Ollama, llama.cpp, and vLLM make it trivial to run capable language models locally. A model like Llama 3.2 with 8 billion parameters runs on a machine with as little as 8 GB of RAM, while more capable models benefit from GPU acceleration without requiring datacenter-class hardware.

The tooling ecosystem matured. Libraries for document parsing (PyMuPDF, pdfplumber, Unstructured), vector storage (FAISS, Qdrant, Chroma), OCR (Tesseract, PaddleOCR, EasyOCR), and API development (FastAPI) all reached production stability. You no longer need to build these components yourself.

What We Are Building

This book guides you through building a complete local search engine with the following capabilities:

- **Multi-format document ingestion** for PDFs, Word documents, spreadsheets, presentations, images, text files, code, and more
- **OCR and multimodal processing** for scanned documents and images
- **Intelligent chunking strategies** that preserve semantic boundaries
- **Local embedding generation** with GPU acceleration
- **Vector database indexing** with multiple backend options
- **Hybrid search** combining keyword (BM25) and semantic retrieval
- **Cross-encoder reranking** for precision refinement
- **Retrieval-augmented generation** powered by local LLMs
- **Incremental indexing** with file system monitoring
- **Query understanding and expansion** including HyDE techniques
- **Performance caching** at multiple layers
- **Security and access control** with JWT authentication
- **A REST API** built with FastAPI
- **Cross-platform deployment** on Windows, macOS, and Linux

How to Read This Book

This book is organized as a building process. Each chapter adds meaningful capability to the system. You can read it sequentially, following along with the code examples, or you can jump to specific chapters that address your immediate needs. The code is designed to be modular: each component can stand alone or integrate with the others.

You should be comfortable with Python, understand basic machine learning concepts (vectors, similarity, neural networks), and have access to a development machine. A GPU is helpful but not required for most of the book; we discuss CPU-only alternatives throughout.

By the end, you will have built a search engine that rivals commercial solutions in capability while giving you complete ownership of your data, your infrastructure, and your code.

Chapter 1: The Case for Local AI Search

Why Cloud Search Falls Short

The argument for local search is not that cloud solutions are bad. They excel at web-scale discovery, multilingual coverage, and convenience. But when you are searching your own documents, the value proposition flips entirely.

Consider a law firm with 50,000 case files spanning two decades. Every document contains privileged client information protected by attorney-client privilege and subject to strict confidentiality obligations. Uploading these files to a cloud search API means trusting that provider's security team, their incident response procedures, and their legal compliance across multiple jurisdictions. Even a minor breach notification can trigger regulatory consequences that far exceed any convenience gained.

Now consider a research lab working on an unpublished breakthrough. Their experimental notes, preliminary results, and internal analyses are exactly the kind of documents they want to search efficiently. But uploading them to a cloud service creates an unnecessary attack surface and complicates intellectual property protection. The answer is not to avoid search; it is to run search locally.

The cost argument is equally compelling. Embedding one million documents through a commercial API costs approximately $120 to $200 for the initial embedding pass, plus ongoing storage fees of roughly $50 per month. For a growing corpus, these costs compound. A local system that runs on existing hardware has no marginal cost per document or per query beyond electricity. The break-even point typically arrives within months, not years.

Vendor lock-in presents a subtler but more persistent problem. When your search infrastructure lives in a proprietary platform, every upgrade, pricing change, or feature deprecation is someone else's decision. Your embeddings are in their format, your index structure follows their schema, and your query patterns depend on their API contract. Migrating away becomes a multi-month engineering project rather than a configuration change.

The Local AI Revolution

Three technological shifts converged to make local AI search practical for individual developers and small teams:

Open embedding models achieved production quality. In early 2024, the gap between commercial and open embedding models was significant. By mid-2025, models like BGE-M3, Nomic Embed v1.5, and E5-mistral-7b-instruct were scoring competitively on the MTEB leaderboard, with some exceeding the performance of OpenAI's text-embedding-3-large on specific retrieval tasks. The key insight is that embedding quality is now good enough that the difference between models matters less than the difference between having search and not having it.

Local LLM inference became trivial. Ollama reduced model deployment to a single command: `ollama run llama3.2`. Under the hood, it handles model downloading, quantization, GPU detection, and a REST API for inference. For developers who want more control, llama.cpp provides Python bindings with support for CPU inference, GPU offloading, and the GGUF model format. vLLM offers high-throughput serving with PagedAttention for teams that need to handle concurrent queries efficiently.

The supporting ecosystem matured. Every component needed for a production search engine now has at least one well-maintained open-source option:

- Document parsing: PyMuPDF, pdfplumber, Unstructured, python-docx, openpyxl
- OCR: Tesseract, PaddleOCR, EasyOCR
- Vector storage: FAISS, Qdrant, Chroma
- Full-text search: Elasticsearch, OpenSearch, Whoosh
- API framework: FastAPI
- Embedding generation: sentence-transformers

What We Are Building

The system we will build has these design goals:

Privacy-first. No document content ever leaves the local machine. All embeddings, all indexing, all inference happens locally. The API can be exposed over a network, but the data stays on-premise.

Format-agnostic. The system handles PDFs, Word documents, Excel spreadsheets, PowerPoint presentations, images (with OCR), plain text, code files, Markdown, HTML, JSON, YAML, and CSV. Each format gets specialized parsing to extract maximum information.

Hybrid retrieval. Keyword search (BM25) and semantic search (vector embeddings) run in parallel and fuse their results using Reciprocal Rank Fusion. This gives the precision of exact matching with the flexibility of semantic understanding.

RAG-ready. Retrieved documents feed directly into a local LLM for answer generation, summarization, and question answering. The system supports streaming responses and citation tracking.

Incremental. New documents are indexed as they arrive. Modified documents trigger re-indexing. Deleted documents are removed from the index. No full rebuilds required.

Observable. Query latency, throughput, cache hit rates, and retrieval quality metrics are tracked and exposed for monitoring.

Hardware and Operational Trade-offs

Before diving into code, it is important to be honest about what "local" actually costs. The claim that local AI search "costs nothing beyond electricity" is incomplete. Real total cost of ownership includes hardware procurement, electricity, cooling, engineering time for setup and maintenance, dependency management, and the opportunity cost of downtime. Understanding these trade-offs helps you decide whether local search is the right choice for your specific situation.

Minimum hardware requirements. The system described in this book runs on a wide range of hardware, but performance varies dramatically:

Component	Minimum (CPU-only)	Recommended (GPU)	Optimal
CPU	4-core modern x86/ARM	8-core+	16-core+ for high throughput
RAM	8 GB (small corpus)	16-32 GB	32-64 GB for large corpora
GPU	Not required	RTX 3060 (12 GB VRAM)	RTX 4090 (24 GB VRAM) or better
Storage	50 GB SSD	100+ GB NVMe SSD	200+ GB NVMe for large indexes
Embedding model	all-MiniLM-L6-v2 (384 dim, 22M params)	BGE-M3 (1024 dim, 568M params)	BGE-M3 or Nomic Embed v1.5

BGE-M3 requires approximately 1.1 GB of VRAM at FP16 precision for model weights alone. With batch activation memory, expect 2 to 4 GB depending on batch size and sequence length. On an RTX 3060 with 12 GB VRAM, you can comfortably run BGE-M3 alongside a quantized LLM (e.g., Llama 3.2 8B at Q4_K_M uses approximately 5 GB). An RTX 4090 with 24 GB VRAM provides headroom for larger models and higher batch sizes. On Apple Silicon, MPS acceleration supports PyTorch embedding inference, though some operations (certain FAISS algorithms) fall back to CPU automatically [38].

Realistic cost comparison. A 2026 analysis of local versus cloud AI total cost of ownership found that the break-even point between local hardware and OpenAI's text-embedding-3-large API (priced at $0.13 per million tokens as of mid-2025 [40]) arrives at approximately 2 to 3 million tokens per day for consumer-grade hardware. Below this threshold, cloud APIs are cheaper when you factor in hardware depreciation, electricity, and engineering overhead. Above it, local infrastructure delivers better economics over a 36-month horizon [41].

The cloud API cost for embedding one million documents (assuming an average of 500 tokens per chunk and 10 chunks per document) is approximately $6.50 through OpenAI's text-embedding-3-large model. A local system has no

marginal cost per document beyond electricity, but requires upfront hardware investment ($1,500 to $5,000 for a capable consumer workstation) and ongoing engineering time for setup, monitoring, and maintenance.

Engineering overhead. Operating local infrastructure requires engineer hours that cloud APIs eliminate: initial setup (4 to 40 hours depending on complexity), monthly monitoring (2 to 10 hours per month), quarterly upgrades (4 to 8 hours per quarter), and variable debugging time. At a fully-loaded engineer cost of $75 to $200 per hour, these operational costs can match or exceed the raw compute savings [41]. For a solo developer or small team, this overhead is manageable. For an organization without dedicated MLOps staff, it may tip the balance toward cloud solutions.

When cloud search remains superior. Local-only architectures have genuine limitations:

- **Sporadic usage:** If you search documents a few times per week, paying for cloud APIs on demand is cheaper than maintaining hardware that sits idle 95 percent of the time.
- **Extreme scale without dedicated infrastructure:** Indexing and searching billions of document chunks requires distributed systems expertise and datacenter-grade hardware that most organizations do not have in-house.
- **Multilingual coverage at scale:** Commercial embedding models trained on massive multilingual corpora (OpenAI, Cohere, Jina) still lead on certain low-resource languages. Open models are closing the gap rapidly but have not fully caught up for all language pairs.
- **Frontier model access:** Cloud APIs provide immediate access to the latest proprietary models. Local deployment lags by 3 to 6 months as open-weight models are released, evaluated, and optimized.

The hybrid approach. Many production systems use a hybrid strategy: local embedding and vector search for privacy-sensitive documents, supplemented by cloud APIs for non-sensitive enrichment tasks (translation, summarization of public content). This gives you the best of both worlds but adds operational complexity.

With these trade-offs in mind, let us proceed to building the system. Every code example in this book is designed to run on consumer hardware with a CPU-only fallback, but GPU acceleration provides substantial speedups for embedding generation and LLM inference.

Hardware Requirements and Operational Trade-offs

Before diving into code, it is important to establish realistic expectations about hardware. Local AI search does not "cost nothing beyond electricity." The total cost of ownership includes hardware purchase or depreciation, electricity, storage, and engineering time. The following specifications are based on actual model memory requirements and benchmark measurements as of mid-2026.

Minimum specifications (CPU-only, development use):

- CPU: 8-core modern processor (Intel i7/Ryzen 7 or equivalent, 2020 or newer)
- RAM: 16 GB system memory
- Storage: 50 GB SSD for models, indexes, and data
- GPU: None required (CPU inference only)

This configuration handles embedding generation for small to medium corpora (up to approximately 100,000 chunks) at roughly 50 to 200 chunks per minute using all-MiniLM-L6-v2. Query latency is typically 100 to 500 ms for vector search. Suitable for development, prototyping, and light production use.

Recommended specifications (GPU-accelerated, production use):

- CPU: 12-core or higher
- RAM: 32 GB system memory
- GPU: NVIDIA RTX 3060 12GB ($300), RTX 4060 Ti 16GB ($450), or RTX 4090 24GB ($1,600)
- Storage: 200 GB NVMe SSD for fast index access
- Network: Gigabit Ethernet for API serving

This configuration handles embedding generation at 500 to 2,000 chunks per minute with BGE-M3, sub-50 ms query latency, and supports running a local LLM (Llama 3.2 8B) for RAG. Suitable for production deployments serving multiple users.

VRAM requirements by component:

Component	Model	FP16 VRAM	INT4 Quantized	Notes
Embedding (small)	all-MiniLM-L6-v2	~90 MB	~50 MB	Fits on any GPU
Embedding (balanced)	BGE-M3	~1.1 GB weights + 2-4 GB batch memory	~0.6 GB	Batch size 32 needs ~4 GB total VRAM [38]
Embedding (quality)	E5-mistral-7b-instruct	~14 GB	~4.5 GB	Requires 16+ GB GPU at FP16; INT4 fits on 8 GB cards
Reranker	ms-marco-MiniLM-L-6-v2	~150 MB	~80 MB	Negligible overhead
LLM (small)	Llama 3.2 1B	~2 GB	~1 GB	CPU inference viable
LLM (balanced)	Llama 3.2 8B	~16 GB	~5 GB	Needs 8+ GB GPU at INT4; CPU inference is slow but works with 16+ GB RAM
LLM (large)	Llama 3.1 70B	~140 GB	~40 GB	Requires dual-GPU or cloud for practical use

Source: VRAM estimates from Hugging Face model cards, WillItRunAI compatibility data, and NVIDIA NIM documentation [38, 39].

Realistic cost comparison (annualized):

Cost Component	Local Deployment	Cloud API Alternative
Hardware (amortized over 3 years)	$500 to $1,700/year	$0
Electricity (24/7, 300W average)	$250 to $400/year	Included in API cost
Storage (local SSD depreciation)	$100 to $200/year	$50 to $200/month for vector DB hosting
Embedding API costs (1M docs, 500 tokens/chunk, 10 chunks/doc)	$0	~$6.50 per indexing pass via OpenAI text-embedding-3-large [40]
LLM API costs (1,000 RAG queries/day, 2K context + 500 output)	$0	~$150 to $500/month depending on model
Engineering overhead (setup, monitoring, upgrades)	$3,000 to $10,000/year at $75-200/hr [41]	Near-zero for managed services
Total annualized	**$3,850 to $12,300/year**	**$2,100 to $9,000/year** (API + minimal ops)

The local approach pays for itself when you have:

- A corpus larger than approximately 100,000 documents (embedding costs compound)
- High query volume (thousands of searches per day)
- Privacy or regulatory requirements that mandate data residency
- An existing engineer who can absorb the operational overhead without additional headcount

For organizations with fewer than 50,000 documents, sporadic search usage, and no dedicated MLOps staff, cloud APIs may remain the more economical choice despite the privacy trade-off.

When cloud search remains superior. Local-only architectures have genuine limitations beyond cost:

- **Sporadic usage:** If you search documents a few times per week, paying for cloud APIs on demand is cheaper than maintaining hardware that sits idle 95 percent of the time.
- **Extreme scale without dedicated infrastructure:** Indexing and searching billions of document chunks requires distributed systems expertise and datacenter-grade hardware that most organizations do not have in-house.
- **Multilingual coverage at scale:** Commercial embedding models trained on massive multilingual corpora (OpenAI, Cohere, Jina) still lead on certain low-resource languages. Open models are closing the gap rapidly but have not fully caught up for all language pairs.
- **Frontier model access:** Cloud APIs provide immediate access to the latest proprietary models. Local deployment lags by 3 to 6 months as open-weight models are released, evaluated, and optimized.

The hybrid approach. Many production systems use a hybrid strategy: local embedding and vector search for privacy-sensitive documents, supplemented by cloud APIs for non-sensitive enrichment tasks (translation, summarization of public content). This gives you the best of both worlds but adds operational complexity.

With these trade-offs in mind, let us proceed to building the system. Every code example in this book is designed to run on consumer hardware with a CPU-only fallback, but GPU acceleration provides substantial speedups for embedding generation and LLM inference.

Prerequisites and Environment Setup

You need Python 3.10 or later. We recommend Python 3.12 for its performance improvements and long-term support status. Create a virtual environment to isolate dependencies:

```
# Create a virtual environment
python -m venv search-engine
source search-engine/bin/activate  # On Linux/macOS
# search-engine\Scripts\activate   # On Windows

# Install core dependencies
pip install fastapi uvicorn pydantic
pip install pymupdf pdfplumber python-docx openpyxl python-pptx
pip install sentence-transformers faiss-cpu chromadb
pip install qdrant-client rank-bm25
pip install watchdog tesseract-ocr pytesseract
pip install pillow numpy torch
```

For GPU acceleration with embedding models, install the CUDA-enabled PyTorch version:

```
# NVIDIA GPU with CUDA 12.4
pip install torch torchvision torchaudio --index-url
↪  https://download.pytorch.org/whl/cu124

# Or for CPU-only operation (slower but universally compatible)
pip install torch torchvision torchaudio --index-url
↪  https://download.pytorch.org/whl/cpu
```

A GPU is recommended but not required. The embedding models we use run on CPU with acceptable performance for most development workloads. A modern multi-core CPU can process hundreds of documents per minute; a mid-range GPU (GTX 1660 or better) accelerates this by 5 to 10 times.

Your First Search Engine in Ten Lines

Before we dive into architecture, let us see how little code it takes to build something that works. This minimal prototype demonstrates the core concept: embed documents, embed a query, find the nearest match.

```python
import numpy as np
from sentence_transformers import SentenceTransformer

# Load a lightweight embedding model (downloads on first run)
model = SentenceTransformer('all-MiniLM-L6-v2')

# Your documents
documents = [
    "The company reported revenue of $4.2 billion in Q3 2025.",
    "Python is a high-level programming language known for readability.",
    "Machine learning models require large datasets for training.",
    "The quarterly earnings exceeded analyst expectations by 12%.",
]

# Embed all documents at once (batch processing)
doc_embeddings = model.encode(documents, normalize_embeddings=True)

# The user's query
query = "What were the financial results?"
query_embedding = model.encode(query, normalize_embeddings=True)

# Compute cosine similarity (dot product of normalized vectors)
similarities = doc_embeddings @ query_embedding

# Return the top result
best_idx = np.argmax(similarities)
print(f"Best match: {documents[best_idx]}")
print(f"Similarity score: {similarities[best_idx]:.4f}")
```

This outputs:

```
Best match: The company reported revenue of $4.2 billion in Q3 2025.
Similarity score: 0.6847
```

Ten lines of code, and you have semantic search working. The query "What were the financial results?" correctly matched the revenue document despite sharing no exact keywords, because the embedding model understands that "financial results" and "revenue" are semantically related concepts.

This prototype has many limitations: it loads everything into memory, handles only plain text, provides no keyword search, has no persistence, and offers no API. The rest of this book addresses each limitation systematically, building toward a production-ready system. But the core idea is simple and powerful: convert text to vectors, compare vectors with math, return the closest matches. Everything else is engineering around this center.

The Evolving Project Structure

This book builds a single, cohesive codebase that grows incrementally from the ten-line prototype above to a feature-rich desktop search application. Each chapter adds real capability to the same repository. You can follow along sequentially or jump to specific chapters, but the project structure remains consistent throughout.

The base repository tree looks like this:

```
local-search-engine/
├── src/
│   ├── __init__.py
│   ├── config.py              # Centralized configuration
│   ├── ingestion/
│   │   ├── __init__.py
│   │   ├── watcher.py         # File system monitoring (Ch 2)
│   │   ├── pipeline.py        # Ingestion pipeline orchestrator (Ch 2)
│   │   └── parsers/
│   │       ├── __init__.py
│   │       ├── pdf_parser.py  # PDF extraction (Ch 3)
│   │       ├── office_parser.py # Word/Excel/PPT (Ch 3)
│   │       ├── ocr_engine.py  # OCR pipeline (Ch 3)
│   │       └── html_parser.py # Web content (Ch 3)
│   ├── chunking/
│   │   ├── __init__.py
│   │   ├── base.py            # Chunker interface (Ch 4)
│   │   ├── fixed.py           # Fixed-size with overlap (Ch 4)
│   │   ├── semantic.py        # Semantic boundary detection (Ch 4)
│   │   └── hierarchical.py    # Parent-child hierarchies (Ch 4)
│   ├── embedding/
│   │   ├── __init__.py
│   │   ├── generator.py       # Embedding pipeline (Ch 5)
│   │   └── multimodal.py      # CLIP image embeddings (Ch 5)
│   ├── index/
│   │   ├── __init__.py
│   │   ├── base.py            # VectorIndex ABC (Ch 6)
│   │   ├── faiss_index.py     # FAISS backend (Ch 6)
│   │   ├── qdrant_index.py    # Qdrant backend (Ch 6)
│   │   ├── chroma_index.py    # Chroma backend (Ch 6)
│   │   └── bm25.py            # BM25 keyword search (Ch 7)
│   ├── search/
│   │   ├── __init__.py
│   │   ├── hybrid.py          # Hybrid search engine (Ch 7)
│   │   ├── fusion.py          # RRF and weighted fusion (Ch 7)
│   │   └── reranker.py        # Cross-encoder reranking (Ch 8)
│   ├── rag/
```

```
│   │   ├── __init__.py
│   │   ├── pipeline.py         # RAG orchestration (Ch 8)
│   │   ├── prompts.py          # Prompt templates (Ch 8)
│   │   └── llm_client.py       # Ollama/local LLM client (Ch 8)
│   ├── query/
│   │   ├── __init__.py
│   │   ├── classifier.py       # Query intent detection (Ch 9)
│   │   └── hyde.py             # HyDE expansion (Ch 9)
│   ├── cache/
│   │   ├── __init__.py
│   │   └── search_cache.py     # Multi-layer caching (Ch 10)
│   └── api/
│       ├── __init__.py
│       ├── server.py           # FastAPI application (Ch 13)
│       └── models.py           # Pydantic request/response models (Ch 13)
├── tests/
│   ├── test_parsers.py
│   ├── test_chunkers.py
│   ├── test_search.py
│   └── test_rag.py
├── documents/                   # Sample documents for testing
├── data/                        # Indexes, embeddings, cache
├── config.yaml                  # Application configuration
├── requirements.txt
├── docker-compose.yml           # Multi-service deployment (Ch 14)
└── README.md
```

How components wire together. The system follows a dependency-injection pattern. The `config.py` module reads settings from `config.yaml` and instantiates the appropriate backends:

```python
# config.py -- simplified version shown here; full version in Chapter 13
from src.embedding.generator import EmbeddingGenerator
from src.index.factory import IndexFactory
from src.search.hybrid import HybridSearchEngine

def create_search_engine(config: dict) -> HybridSearchEngine:
    """Wire together all components from configuration."""
    embedder = EmbeddingGenerator(
        model_name=config["embedding"]["model"],
        batch_size=config["embedding"]["batch_size"],
    )
    vector_index = IndexFactory.create(
        index_type=config["index"]["backend"],  # "faiss", "qdrant", "chroma"
        dimensions=embedder.dimensions,
        **config["index"]["options"],
    )
    return HybridSearchEngine(
```

```
        vector_index=vector_index,
        embedding_generator=embedder,
        chunk_store_size=config.get("cache", {}).get("max_chunks", 100_000),
    )
```

This wiring function is the integration point that every chapter builds toward. When you read Chapter 6 about vector databases, the `IndexFactory.create()` call lets you swap FAISS for Qdrant by changing one config line. When you read Chapter 5 about embedding models, the `EmbeddingGenerator` class handles model loading, GPU detection, and batch processing transparently.

Incremental build path. Here is how the project evolves across chapters:

Phase	Chapters	What Gets Added
Foundation	Ch 1	Ten-line prototype, environment setup, project tree
Ingestion	Ch 2-3	File watcher, format detection, parsers for all document types
Processing	Ch 4-5	Chunking strategies, embedding generation, multimodal support
Indexing	Ch 6	Vector backends (FAISS, Qdrant, Chroma, pgvector, OpenSearch)
Search	Ch 7	BM25 keyword search, hybrid fusion, RRF
Intelligence	Ch 8-9	RAG pipeline, local LLM integration, query understanding
Production	Ch 10-12	Caching, incremental indexing, security, access control
Interface	Ch 13	REST API, desktop application (Tauri/PyQt6)
Deployment	Ch 14-16	Docker, cross-platform packaging, scalability, case studies

By the end of Chapter 7, you have a working hybrid search engine. By the end of Chapter 8, it generates answers via RAG. By the end of Chapter 13, it runs as a desktop application. Each phase is independently testable and deployable.

Chapter 2: Document Ingestion Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

File System Monitoring and Discovery

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Format Detection and Routing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

The Ingestion Pipeline Design

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Building the Document Abstraction Layer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Production-Ready File Watcher Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Chapter 3: Parsing Every Document Type

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

PDF Parsing Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Microsoft Office Documents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Spreadsheet Processing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Image and Scan Handling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Web Pages and HTML

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Code Files and Structured Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Debugging and Pitfalls: Document Parsing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Chapter 4: Text Processing and Chunking Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Why Chunking Matters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Fixed-Size Chunking with Overlap

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Semantic Chunking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Hierarchical Chunking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Format-Aware Chunking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Chunking Evaluation and Tuning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Debugging and Pitfalls: Chunking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Chapter 5: Embedding Models and Vector Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

How Embeddings Work

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Choosing Local Embedding Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Implementing the Embedding Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Multimodal Embeddings for Images

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Embedding Performance Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Debugging and Pitfalls: Embedding Models

Chapter 6: Vector Databases and Indexing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Vector Database Landscape

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

FAISS for High-Performance Indexing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Qdrant for Production Search

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Chroma for Simplicity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

PostgreSQL with pgvector for Relational Vector Search

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Elasticsearch and OpenSearch for Enterprise Hybrid Search

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Debugging and Pitfalls: Vector Databases

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Chapter 7: Hybrid Search Engine Design

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

The Keyword vs Semantic Tradeoff

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Implementing BM25 from Scratch

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Fusion Strategies: Reciprocal Rank Fusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Building the Hybrid Search Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Incremental BM25 Indexing for Production

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Chapter 8: Retrieval-Augmented Generation (RAG)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

RAG Architecture Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Local LLM Integration with Ollama

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Prompt Engineering for Search

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Streaming Responses and Real-Time Feedback

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Cross-Encoder Reranking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Debugging and Pitfalls: RAG Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Advanced RAG Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

RAG Evaluation Frameworks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Chapter 9: Query Processing and Understanding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Query Classification and Intent Detection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

HyDE: Hypothetical Document Embeddings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Chapter 10: Performance, Caching, and Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Caching Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Memory Management for Large Corpora

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Benchmarking Methodology

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Chapter 11: Incremental Indexing and Data Freshness

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Change Detection Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Incremental Index Updates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Chapter 12: 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/localaisearchengine.

Threat Model for Local Search

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Authentication with FastAPI and JWT

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Data Isolation and Multi-Tenancy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Chapter 13: API Design and Frontend Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

RESTful API Design with FastAPI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Building a Desktop Application with Tauri

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Debugging and Pitfalls: Desktop Application Development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Chapter 14: Deployment and Cross-Platform Operations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Containerization with Docker

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Cross-Platform Build Challenges

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Monitoring and Observability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Chapter 15: Evaluation, Testing, and Quality Assurance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Search Quality Metrics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Building a Test Corpus

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Automated Evaluation Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Chapter 16: Scalability, Case Studies, and Production Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Scaling Beyond a Single Machine

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Index Sharding Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Handling Million-Document Corpora

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Illustrative Case Studies: Archetypal Deployments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Archetype 1: Legal Contract Search

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Archetype 2: Codebase Indexing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Archetype 3: Research Lab Document Archive

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Load Balancing and High Availability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Profiling and Optimization Workflow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Architecture Decision Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Conclusion: The Future of Local AI Search

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

What We Built

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Emerging Technologies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Scaling Beyond Local

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

Your Next Steps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/localaisearchengine.