# **RAG Pipeline - Production Ready Implementation**

A production-ready Retrieval-Augmented Generation (RAG) pipeline for document question-answering, built with LangChain and best software engineering practices.

## **🚀 Features**

* **Modular Architecture**: Clean separation of concerns with abstract base classes
* **Production Logging**: Structured JSON logging with rotation and multiple handlers
* **Comprehensive Testing**: Unit and integration tests with pytest
* **Configuration Management**: Environment-based configuration with Pydantic
* **Error Handling**: Custom exceptions for better debugging
* **Performance Monitoring**: Metrics tracking and evaluation with Ragas
* **Docker Support**: Containerized deployment ready
* **CI/CD Ready**: Pre-commit hooks, linting, and formatting

## **📋 Prerequisites**

* Python 3.8+
* OpenAI API Key
* (Optional) LangSmith API Key for observability

## **🛠️ Installation**

### **Development Setup**

# Clone the repository

git clone <repository-url>

cd rag-pipeline

# Create virtual environment

python -m venv venv

source venv/bin/activate # On Windows: venv\Scripts\activate

# Install dependencies

make install-dev

# Copy environment variables

cp .env.example .env

# Edit .env with your API keys

### **Production Setup**

# Install only production dependencies

make install

# Or using Docker

make docker-build

## **🔧 Configuration**

Create a .env file based on .env.example:

# Required

OPENAI\_API\_KEY=your-openai-api-key

# Optional

LANGSMITH\_API\_KEY=your-langsmith-api-key

ENVIRONMENT=production

LOG\_LEVEL=INFO

# Model Configuration

LLM\_MODEL=gpt-3.5-turbo

EMBEDDING\_MODEL=text-embedding-ada-002

# Chunking Configuration

CHUNK\_SIZE=1000

CHUNK\_OVERLAP=200

## **📖 Usage**

### **Command Line Interface**

# Index documents

python main.py --mode index --documents path/to/doc1.pdf path/to/doc2.pdf

# Query the system

python main.py --mode query --question "Your question here" --load-existing

# Run evaluation

python main.py --mode evaluate --evaluation-file path/to/eval\_data.json --load-existing

# Run demo

python main.py --mode demo

### **Using Make Commands**

# Index documents

make run-index

# Query with a question

make run-query QUESTION="How do I install a SIM card?"

# Run evaluation

make run-evaluate

# Run demo

make run-demo

### **Programmatic Usage**

from config.settings import settings

from src.data\_loader.pdf\_loader import PDFDocumentLoader

from src.text\_processing.chunker import DocumentChunker

from src.embeddings.openai\_embeddings import OpenAIEmbeddingModel

from src.vector\_store.chroma\_store import ChromaVectorStore

from src.llm.openai\_llm import OpenAILLM

from src.retrieval.qa\_chain import QAChain

# Initialize components

pdf\_loader = PDFDocumentLoader()

chunker = DocumentChunker(chunk\_size=1000, chunk\_overlap=200)

embedding\_model = OpenAIEmbeddingModel(api\_key=settings.openai\_api\_key)

vector\_store = ChromaVectorStore(embedding\_model=embedding\_model)

llm = OpenAILLM(api\_key=settings.openai\_api\_key)

# Load and process documents

documents = pdf\_loader.load\_multiple([Path("doc1.pdf"), Path("doc2.pdf")])

chunks = chunker.process(documents)

vector\_store.add\_documents(chunks)

# Create QA chain and answer questions

qa\_chain = QAChain(llm=llm, vector\_store=vector\_store)

response = qa\_chain.run("Your question here")

print(response["answer"])

## **🧪 Testing**

# Run all tests

make test

# Run only unit tests

make test-unit

# Run only integration tests

make test-integration

# Run tests with coverage report

pytest --cov=src --cov-report=html

## **🔍 Code Quality**

# Run linting

make lint

# Format code

make format

# Run all quality checks

pre-commit run --all-files

## **📊 Monitoring and Logging**

The application uses structured JSON logging with the following features:

* **Log Rotation**: Automatic rotation when log files reach 10MB
* **Multiple Handlers**: Console (human-readable) and file (JSON) outputs
* **Contextual Logging**: Request IDs and additional metadata
* **Log Levels**: Configurable via environment variables

Example log output:

{

"timestamp": "2024-01-15T10:30:45.123Z",

"level": "INFO",

"logger": "src.retrieval.qa\_chain",

"message": "Running QA chain for query: 'How do I install...'",

"module": "qa\_chain",

"function": "run",

"line": 85,

"query\_length": 28,

"processing\_time\_seconds": 1.234

}

## **🏗️ Architecture**

src/

├── data\_loader/ # Document loading functionality

├── text\_processing/ # Text chunking and processing

├── embeddings/ # Embedding model implementations

├── vector\_store/ # Vector database operations

├── llm/ # Language model wrappers

├── retrieval/ # RAG chain implementation

├── evaluation/ # Evaluation metrics and tools

└── utils/ # Logging, exceptions, helpers

### **Design Patterns Used**

* **Abstract Factory**: Base classes for all major components
* **Strategy Pattern**: Swappable implementations for embeddings, LLMs, etc.
* **Dependency Injection**: Configuration-driven component initialization
* **Repository Pattern**: Vector store abstraction

## **🐳 Docker Deployment**

# Build image

docker build -t rag-pipeline:latest .

# Run container

docker run -it --rm \

-v $(pwd)/data:/app/data \

-v $(pwd)/logs:/app/logs \

-e OPENAI\_API\_KEY=$OPENAI\_API\_KEY \

rag-pipeline:latest

## **📈 Performance Optimization**

* **Batch Processing**: Process multiple documents simultaneously
* **Caching**: Reuse computed embeddings when possible
* **Async Operations**: Asynchronous API calls where applicable
* **Connection Pooling**: Efficient database connections

## **🔐 Security Considerations**

* API keys stored in environment variables
* Input validation on all user inputs
* Sanitization of file paths
* Rate limiting on API calls
* Secure logging (no sensitive data in logs)

## **🤝 Contributing**

1. Fork the repository
2. Create a feature branch (git checkout -b feature/amazing-feature)
3. Commit your changes (git commit -m 'Add amazing feature')
4. Push to the branch (git push origin feature/amazing-feature)
5. Open a Pull Request

## **📝 License**

This project is licensed under the MIT License - see the LICENSE file for details.

## **🙏 Acknowledgments**

* LangChain community for the excellent framework
* OpenAI for the powerful language models
* Chroma for the vector database
* All contributors and maintainers