Enterprise RAG System Documentation

Complete Technical Guide

Version: 1.0.0
Last Updated: December 2024
Author: Technical Documentation Team

Table of Contents

1. Executive Summary

What is RAG?

Retrieval-Augmented Generation (RAG) is an AI architecture that enhances Large Language Models (LLMs) by grounding their responses in actual documents. Instead of relying solely on

the model's training data, RAG systems:

1. **Retrieve** relevant document chunks based on the user's query
2. **Augment** the LLM prompt with this retrieved context
3. **Generate** answers strictly from the provided context

Why This System?

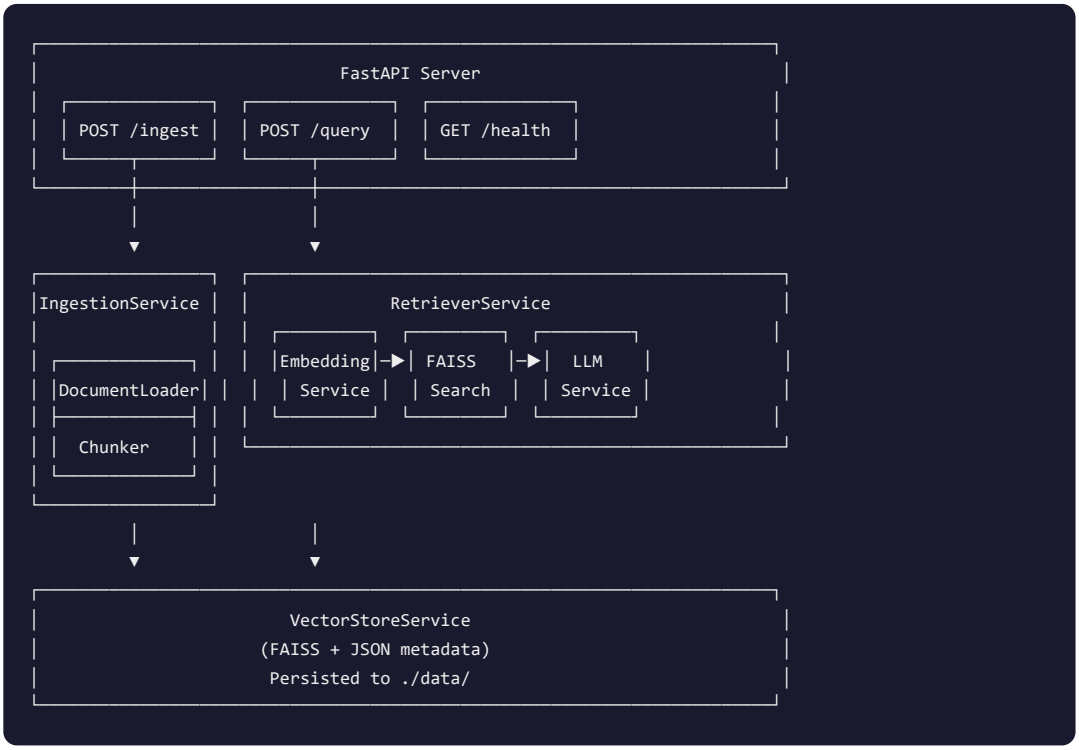
This Enterprise RAG System is a **backend-only** implementation designed for production use. Key features:

Feature	Description
No Hallucination	Answers only from ingested documents
Mandatory Citations	Every answer includes source references
Persistent Storage	FAISS index and metadata survive restarts
Embedding Cache	SHA-256 based caching for efficiency
Multi-Format Support	PDF, DOCX, and Markdown documents

Technology Stack

2. System Architecture

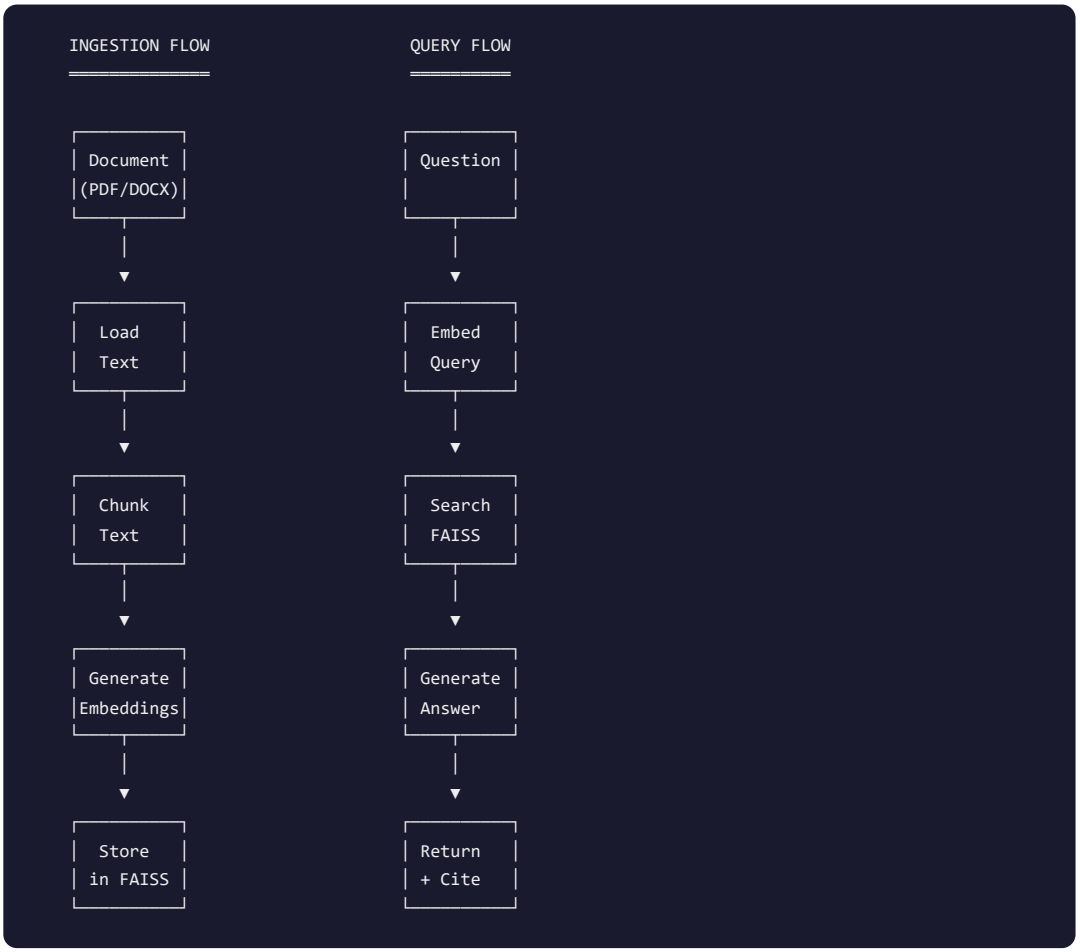
High-Level Architecture Diagram



Component Overview

Component	Responsibility	Key Files
FastAPI Server	HTTP endpoint handling, request validation	`main.py`, `api/routes/`
Ingestion Service	Document loading, chunking, processing	`ingestion/pipeline.py`
Embedding Service	Vector generation with OpenAI	`core/embeddings.py`
Vector Store	FAISS index management, similarity search	`core/vector_store.py`
LLM Service	Context-constrained answer generation	`core/llm.py`
Retriever Service	RAG pipeline orchestration	`core/retriever.py`

Data Flow Overview



3. Core Components

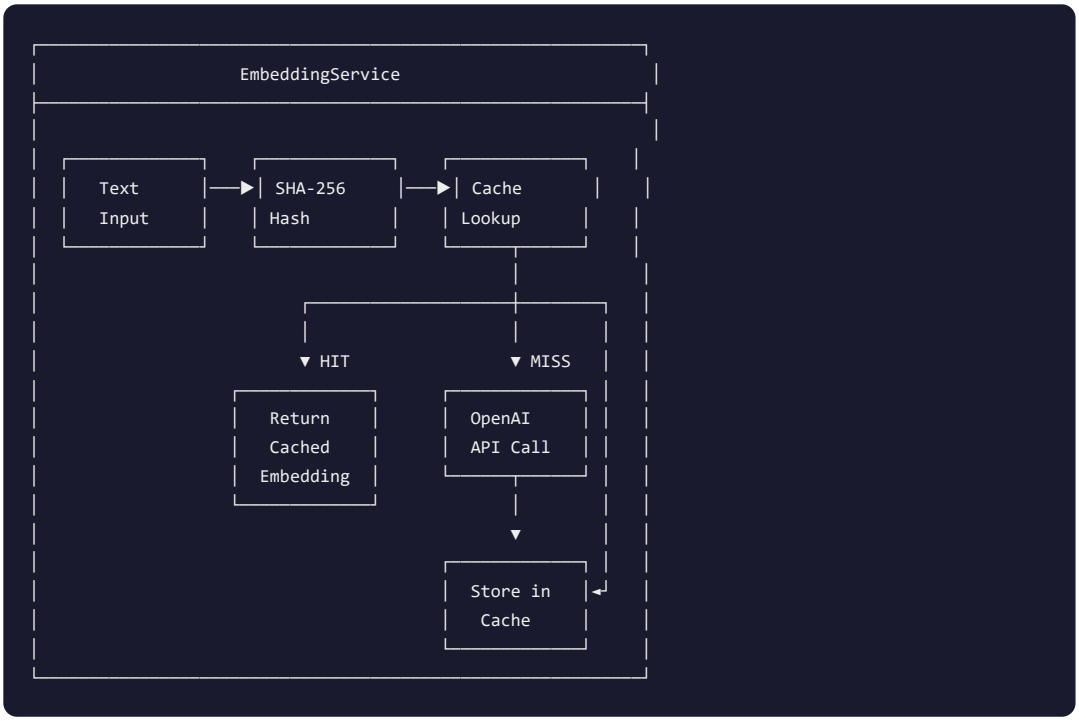
3.1 Embedding Service (`core/embeddings.py`)

The Embedding Service converts text into numerical vectors (embeddings) that capture semantic meaning. These vectors enable similarity search.

Key Features

Feature	Implementation
Model	OpenAI `text-embedding-3-small` (1536 dimensions)
Caching	SHA-256 hash-based persistent cache
Batching	Process up to 100 texts per API call
Retry Logic	Exponential backoff for rate limits

Architecture Diagram



Cache Benefits

1. **Idempotent Indexing**: Same text always produces same embedding
2. **Cost Reduction**: Avoid redundant API calls
3. **Faster Rebuilds**: Cached embeddings persist across restarts

Key Methods

Method	Purpose
`get_embedding(text)`	Single text embedding (cached)
`get_embeddings_batch(texts)`	Batch embedding with cache
`get_query_embedding(query)`	Query embedding (uncached)
`clear_cache()`	Clear all cached embeddings

3.2 LLM Service (`core/llm.py`)

The LLM Service generates answers strictly from provided context. It is the core component ensuring **no hallucination**.

Anti-Hallucination System Prompt

The LLM receives this strict system prompt:

```
ABSOLUTE RULES - VIOLATION IS FORBIDDEN:
1. You may ONLY use information explicitly stated in the provided CONTEXT.
2. You must NEVER use your training data, prior knowledge, or make assumptions.
3. If the CONTEXT does not contain enough information to answer, you MUST respond:
   "I don't know based on the provided documents."
4. If the CONTEXT is empty or completely irrelevant, you MUST respond:
   "Answer not found in documents."
5. Always cite which source(s) you used with [Source N] notation.
```

Configuration

Parameter	Default	Description
`LLM_MODEL`	gpt-4-turbo-preview	OpenAI model
`LLM_TEMPERATURE`	0.0	Deterministic output
`LLM_MAX_TOKENS`	1024	Maximum response length
`LLM_TIMEOUT`	60s	Request timeout

Confidence Scoring

The service calculates a confidence score based on:

Condition	Score
No answer found	0.0
1 relevant context	0.5-0.7
2-3 relevant contexts	0.7-0.85
4+ relevant contexts	0.85-0.95

3.3 Vector Store Service (`core/vector_store.py`)

The Vector Store manages document embeddings using FAISS (Facebook AI Similarity Search).

FAISS Index Configuration

```
Index Type: IndexFlatIP (Inner Product)
Dimension:  1536
Similarity: Cosine (vectors are normalized)
```

Storage Architecture

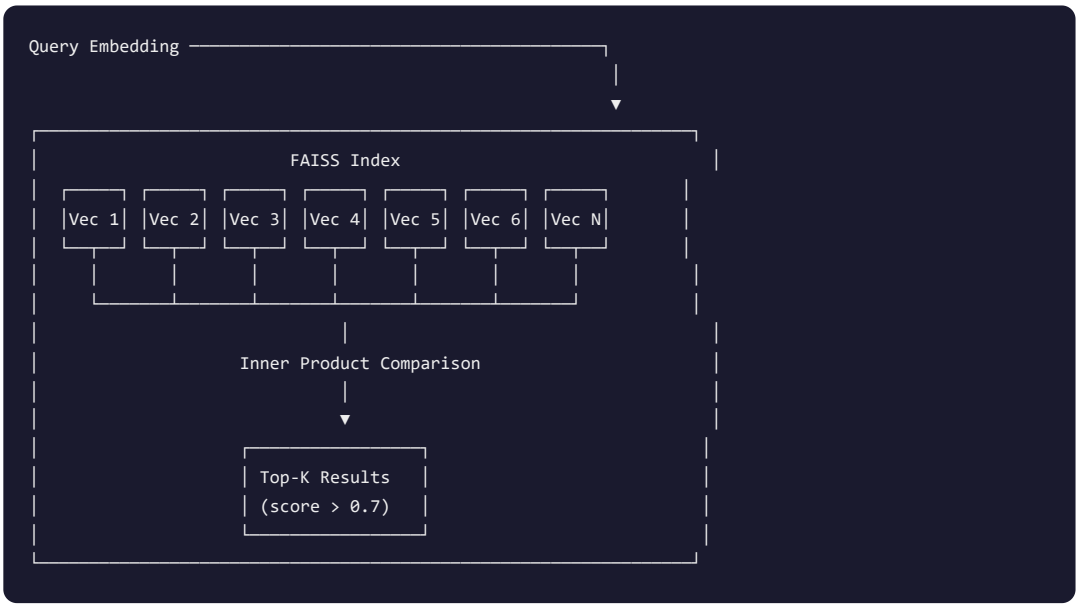
```
./data/faiss_index/
├── faiss.index      # Binary FAISS index file
├── chunks.json      # Chunk text and metadata
└── documents.json   # Document-level metadata

./data/documents/
└── embedding_cache/ # Cached embeddings (NPY files)
```

Key Operations

Operation	Description
`add_chunks()`	Add document chunks with embeddings
`similarity_search()`	Find top-k similar chunks
`delete_document()`	Remove document (rebuilds index)
`get_stats()`	Get index statistics

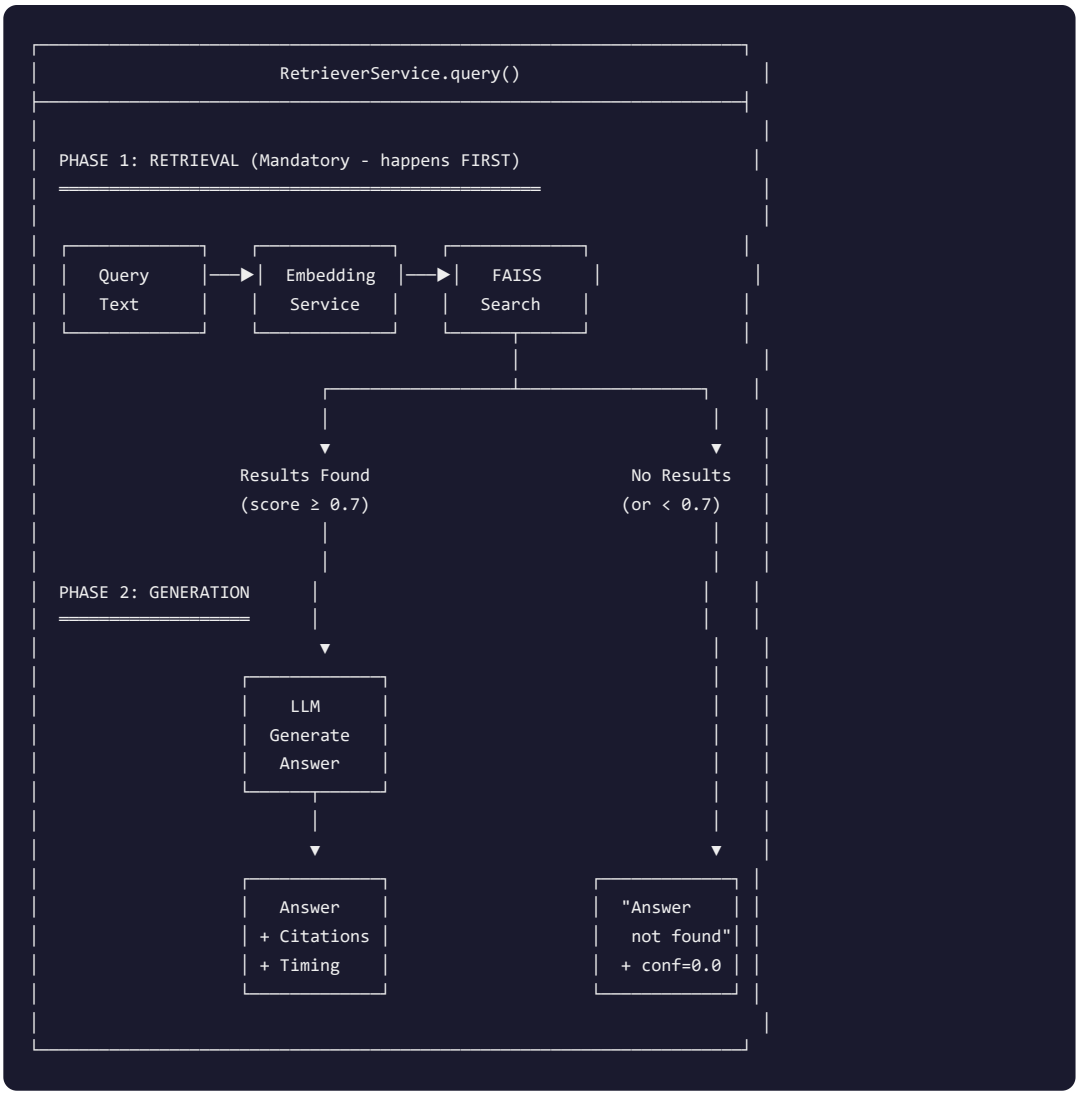
Similarity Search Flow



3.4 Retriever Service (`core/retriever.py`)

The Retriever orchestrates the complete RAG pipeline, connecting all components.

Pipeline Flow



Critical Design: Retrieve First

The retriever **ALWAYS** performs retrieval before generation. This ensures:

1. The LLM only sees relevant context
2. If no context is found, no LLM call is made
3. Resources are not wasted on unanswerable questions

4. Ingestion Pipeline

Overview

The ingestion pipeline processes documents through three stages:

Stage 1: Document Loading (`ingestion/loader.py`)

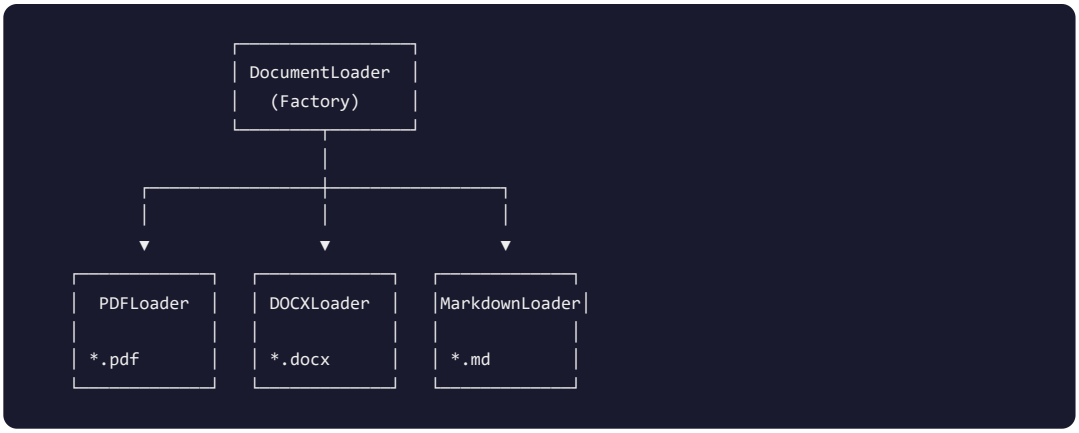
Supported Formats

Format	Loader	Features
PDF	`PDFLoader`	Page-by-page extraction with page numbers
DOCX	`DOCXLoader`	Section detection via heading styles
Markdown	`MarkdownLoader`	Header-based sections, HTML conversion

Document Object

```python
@dataclass
class Document:
    text: str                    # Extracted text content
    metadata: dict[str, Any]     # Source, page, section info
```

Loader Selection



Stage 2: Semantic Chunking (`ingestion/chunker.py`)

Chunking Strategy

The `SemanticChunker` splits documents intelligently:

1. **Split into sentences** using regex patterns
2. **Group sentences** to reach target chunk size
3. **Add overlap** from previous chunk for context continuity

Configuration

Parameter	Default	Description
`CHUNK_SIZE`	512 tokens	Target chunk size
`CHUNK_OVERLAP`	50 tokens	Overlap between chunks

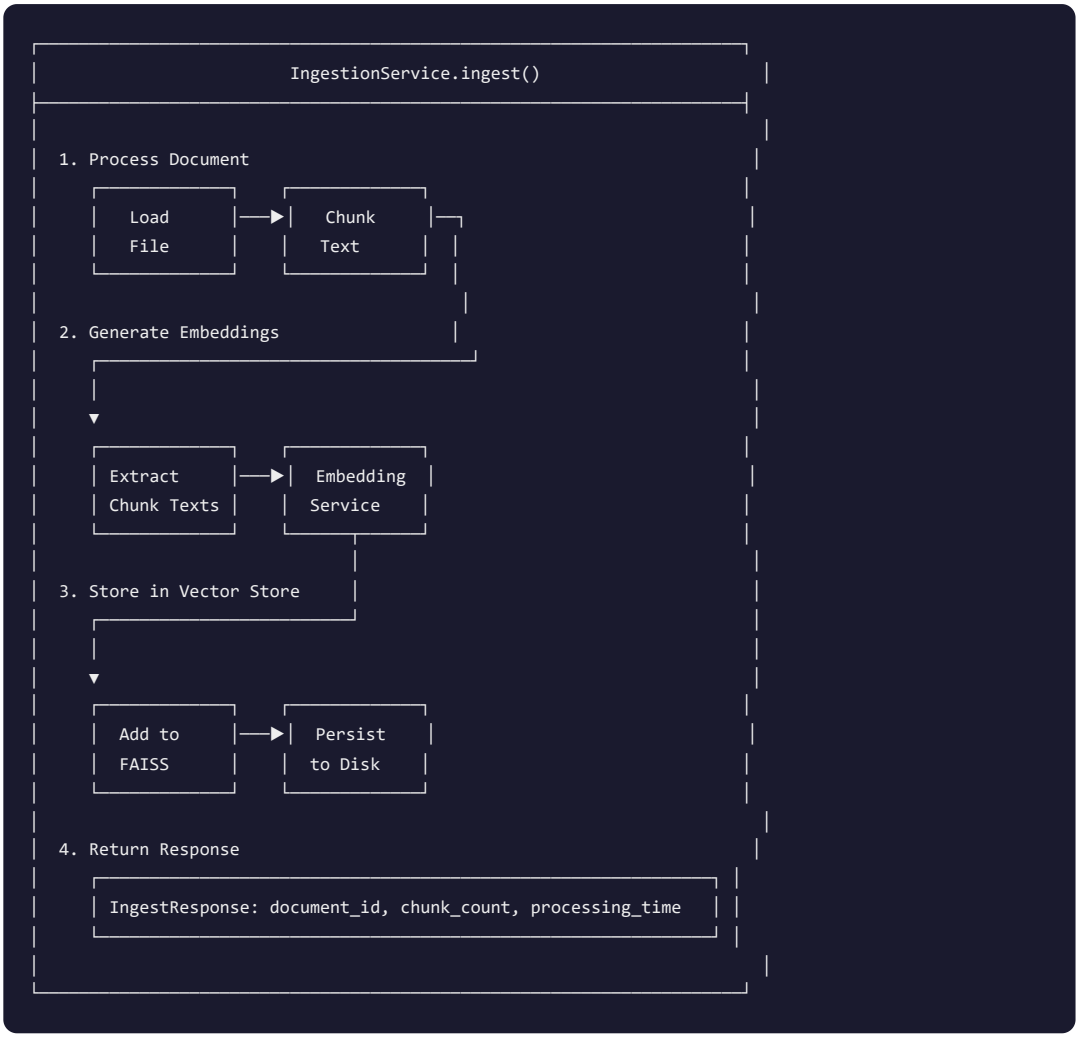
Chunking Visualization

Chunk Object

```python
@dataclass
class Chunk:
    chunk_id: str              # Deterministic ID (SHA-256)
    text: str                  # Chunk content
    metadata: dict[str, Any]   # Source, page, section, index
```

Stage 3: Ingestion Service (`ingestion/pipeline.py`)

The `IngestionService` coordinates the complete flow:


IngestionService.ingest()
1. Process Document
Load File
Chunk Text
2. Generate Embeddings
Extract Chunk Texts
Embedding Service
3. Store in Vector Store
Add to FAISS
Persist to Disk
4. Return Response
IngestResponse: document_id, chunk_count, processing_time

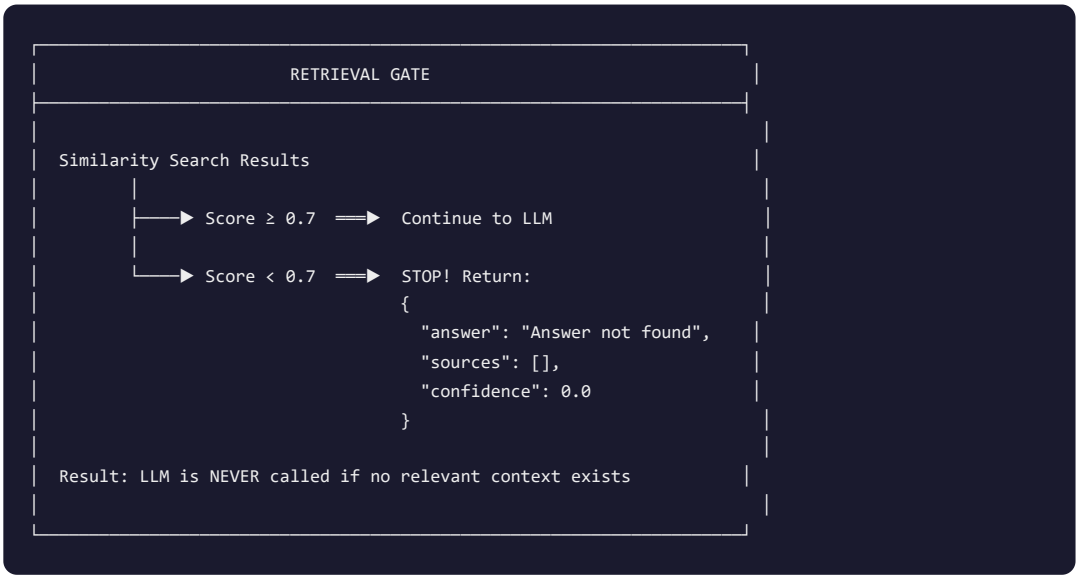
5. Query Pipeline

Complete Query Flow

6. Hallucination Prevention

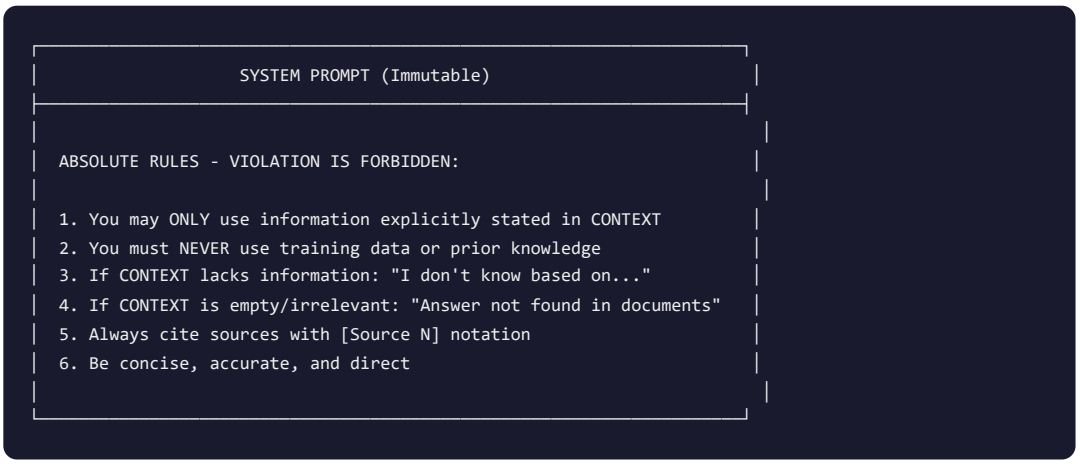
This system implements **four layers** of hallucination prevention:

Layer 1: Retrieval Gate

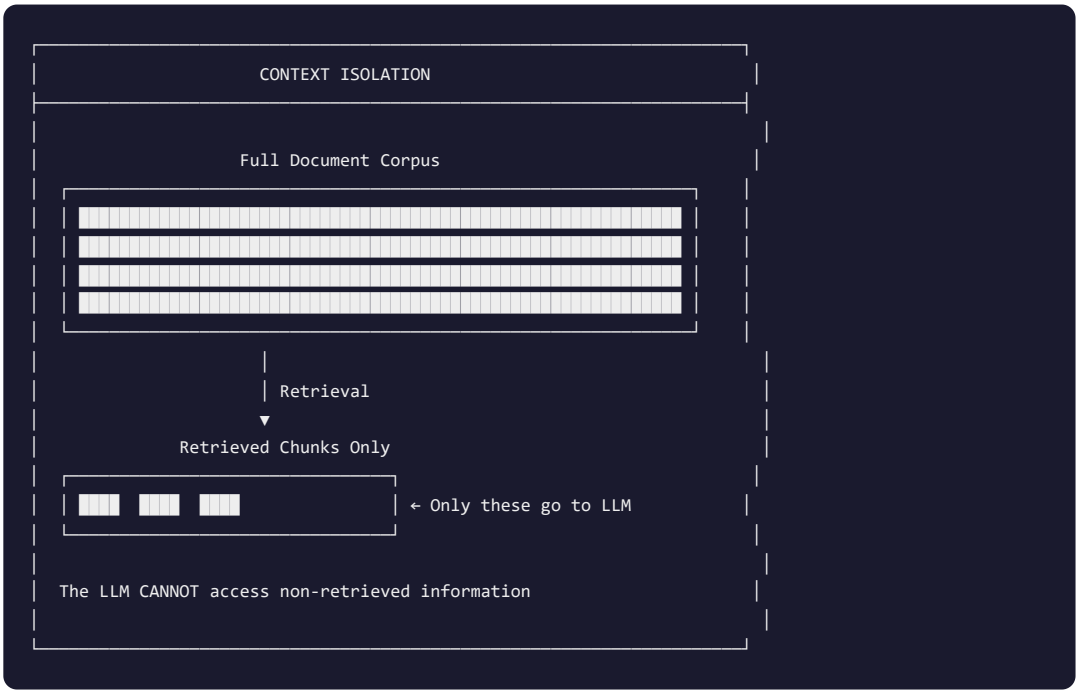


Layer 2: System Prompt Constraint

The LLM receives strict instructions:



Layer 3: Context Isolation



Layer 4: Mandatory Citations

7. API Reference

Base URL

```
http://localhost:8000
```

Endpoints Overview

Method	Endpoint	Description
`GET`	`/health`	Health check with component status
`POST`	`/ingest`	Ingest documents
`POST`	`/query`	Ask questions

GET /health

Check system health and component status.

Request

```
curl http://localhost:8000/health
```

Response (200 OK)

```
{
  "status": "healthy",
  "version": "1.0.0",
  "timestamp": "2024-12-25T10:30:00Z",
  "components": {
    "embeddings": "healthy",
    "vector_store": "healthy",
    "llm": "healthy"
  }
}
```

POST /ingest

Ingest documents into the RAG system.

Request Body

Field	Type	Required	Description
`document_type`	string	Yes	`"pdf"`, `"docx"`, or `"markdown"`
`file_path`	string	No*	Path to document file
`content`	string	No*	Raw markdown content
`metadata`	object	No	Custom metadata

*Either `file_path` or `content` is required.

Example: Ingest PDF

```
curl -X POST http://localhost:8000/ingest \
  -H "Content-Type: application/json" \
  -d '{
    "document_type": "pdf",
    "file_path": "C:/docs/policy.pdf",
    "metadata": {"department": "HR"}
  }'
```

Example: Ingest Markdown Content

```
curl -X POST http://localhost:8000/ingest \
  -H "Content-Type: application/json" \
  -d '{
    "document_type": "markdown",
    "content": "# Policy\n\nEmployees get 20 vacation days per year.",
    "metadata": {"source": "hr_policy"}
  }'
```

Response (200 OK)

```
{
  "success": true,
  "message": "Successfully ingested with 5 chunks",
  "documents": [
    {
      "document_id": "doc_a1b2c3d4e5f6",
      "source": "policy.pdf",
      "document_type": "pdf",
      "chunk_count": 5,
      "ingested_at": "2024-12-25T10:30:00Z",
      "metadata": {"department": "HR"}
    }
  ],
  "total_chunks": 5,
  "processing_time_ms": 1234.56
}
```

POST /query

Ask questions about ingested documents.

Request Body

Field	Type	Required	Default	Description
`question`	string	Yes	-	Question to ask
`top_k`	integer	No	5	Max documents to retrieve
`similarity_threshold`	float	No	0.7	Min relevance score
`include_context`	boolean	No	false	Include chunk text
`metadata_filter`	object	No	null	Filter by metadata

Example Request

```
curl -X POST http://localhost:8000/query \
  -H "Content-Type: application/json" \
  -d '{
    "question": "How many vacation days do employees get?",
    "top_k": 3,
    "similarity_threshold": 0.8
  }'
```

Response (Answer Found)

```
{
  "success": true,
  "answer": "Employees get 20 vacation days per year. [Source 1]",
  "sources": [
    {
      "document_id": "doc_a1b2c3d4e5f6",
      "source": "hr_policy.pdf",
      "page_number": 5,
      "section": "Vacation Policy",
      "relevance_score": 0.89,
      "chunk_text": "Employees get 20 vacation days per year."
    }
  ],
  "confidence": 0.85,
  "query_time_ms": 1456.78,
  "retrieval_time_ms": 45.23,
  "generation_time_ms": 1411.55
}
```

Response (No Answer)

```
{
  "success": true,
  "answer": "Answer not found in documents.",
  "sources": [],
  "confidence": 0.0,
  "query_time_ms": 45.23,
  "retrieval_time_ms": 45.23,
  "generation_time_ms": 0.0
}
```

8. Installation & Setup

Prerequisites

Requirement	Version
Python	3.10+
pip	Latest
OpenAI API Key	Required

Step-by-Step Installation

Step 1: Clone/Navigate to Project

```
cd RAG
```

Step 2: Create Virtual Environment

```
# Create virtual environment
python -m venv venv

# Activate (Windows)
venv\Scripts\activate

# Activate (Linux/Mac)
source venv/bin/activate
```

Step 3: Install Dependencies

```
pip install -r requirements.txt
```

Step 4: Configure Environment

```
# Copy example config
cp .env.example .env

# Edit .env file
notepad .env  # Windows
# OR
nano .env     # Linux/Mac
```

Add your OpenAI API key:

```
OPENAI_API_KEY=sk-your-api-key-here
```

Step 5: Run the Application

```
uvicorn app.main:app --reload --port 8000
```

Step 6: Verify Installation

```
curl http://localhost:8000/health
```

Expected output:

```
{
  "status": "healthy",
  "components": {
    "embeddings": "healthy",
    "vector_store": "healthy",
    "llm": "healthy"
  }
}
```

9. Configuration Reference

All Environment Variables

Variable	Required	Default	Description
Application			
`APP_NAME`	No	Enterprise RAG System	Application name
`APP_ENV`	No	development	Environment (development/staging/production)
`DEBUG`	No	false	Enable debug mode
`LOG_LEVEL`	No	INFO	Logging level (DEBUG/INFO/WARNING/ERROR)
`LOG_FORMAT`	No	json	Log format (json/console)
API			
`API_HOST`	No	0.0.0.0	API server host
`API_PORT`	No	8000	API server port
`CORS_ORIGINS`	No	*	Allowed CORS origins
OpenAI			
`OPENAI_API_KEY`	**Yes**	-	OpenAI API key
`OPENAI_ORG_ID`	No	-	OpenAI organization ID
Embedding			
`EMBEDDING_MODEL`	No	text-embedding-3-small	OpenAI embedding model
`EMBEDDING_DIMENSION`	No	1536	Embedding vector dimension
`EMBEDDING_BATCH_SIZE`	No	100	Batch size for embeddings

LLM			
`LLM_MODEL`	No	gpt-4-turbo-preview	OpenAI LLM model
`LLM_TEMPERATURE`	No	0.0	Temperature (0=deterministic)
`LLM_MAX_TOKENS`	No	1024	Maximum response tokens
`LLM_TIMEOUT`	No	60	Request timeout (seconds)
Retrieval			
`SIMILARITY_THRESHOLD`	No	0.7	Minimum similarity score
`TOP_K`	No	5	Maximum documents to retrieve
Chunking			
`CHUNK_SIZE`	No	512	Chunk size (tokens)
`CHUNK_OVERLAP`	No	50	Overlap between chunks
Storage			
`FAISS_INDEX_PATH`	No	./data/faiss_index	FAISS index directory
`DOCUMENT_STORE_PATH`	No	./data/documents	Document metadata directory

10. Appendix

A. Project Structure

```
RAG/
├── app/
│   ├── __init__.py
│   ├── config.py              # Pydantic Settings configuration
│   ├── main.py                # FastAPI application entry point
│   │
│   ├── api/
│   │   ├── __init__.py
│   │   ├── dependencies.py    # Dependency injection (singletons)
│   │   └── routes/
│   │       ├── __init__.py
│   │       ├── health.py      # GET /health endpoint
│   │       ├── ingest.py      # POST /ingest endpoint
│   │       └── query.py       # POST /query endpoint
│   │
│   ├── core/
│   │   ├── __init__.py
│   │   ├── embeddings.py      # OpenAI embeddings + cache
│   │   ├── llm.py             # LLM service with anti-hallucination
│   │   ├── retriever.py       # RAG orchestration
│   │   └── vector_store.py    # FAISS vector store
│   │
│   ├── ingestion/
│   │   ├── __init__.py
│   │   ├── loader.py          # Document loaders (PDF/DOCX/MD)
│   │   ├── chunker.py         # Semantic text chunking
│   │   └── pipeline.py        # Ingestion orchestration
│   │
│   └── schemas/
│       ├── __init__.py
│       ├── common.py          # HealthResponse, ErrorResponse
│       ├── documents.py       # IngestRequest/Response
│       └── query.py           # QueryRequest/Response
│
├── data/
│   ├── documents/             # Document metadata
│   │   └── embedding_cache/   # Cached embeddings
│   └── faiss_index/           # FAISS index persistence
│       ├── faiss.index
│       ├── chunks.json
│       └── documents.json
│
├── examples/
│   ├── sample_policy.md       # Sample document
│   └── test_rag.py            # Test script
│
├── .env.example               # Example environment config
├── requirements.txt           # Python dependencies
└── README.md                  # Project readme
```

B. Data Persistence

All data is persisted to the `./data/` directory:

File	Purpose
`faiss_index/faiss.index`	Binary FAISS index
`faiss_index/chunks.json`	Chunk text and metadata
`faiss_index/documents.json`	Document-level metadata
`documents/embedding_cache/`	Cached embedding vectors

To reset all data: Delete the `./data/` directory.

C. Dependencies

Core Framework

- `fastapi==0.109.0` - Web framework
- `uvicorn[standard]==0.27.0` - ASGI server
- `pydantic==2.5.3` - Data validation
- `pydantic-settings==2.1.0` - Settings management

AI/ML

- `openai==1.12.0` - OpenAI API client
- `faiss-cpu==1.7.4` - Vector similarity search
- `numpy>=1.24.0,<2.0.0` - Numerical computing

Document Processing

- `pypdf==4.0.1` - PDF text extraction
- `python-docx==1.1.0` - DOCX processing
- `markdown==3.5.2` - Markdown parsing
- `beautifulsoup4==4.12.3` - HTML parsing

Utilities

- `python-dotenv==1.0.0` - Environment loading
- `tenacity==8.2.3` - Retry logic
- `httpx==0.26.0` - HTTP client
- `structlog==24.1.0` - Structured logging

D. Troubleshooting

Issue	Solution
`OPENAI_API_KEY not set`	Add key to `.env` file
`Connection refused`	Check if server is running on port 8000
`No answer found`	Lower `similarity_threshold` or ingest more documents
`Rate limit exceeded`	Built-in retry handles this; wait and retry
`FAISS index corrupted`	Delete `./data/faiss_index/` and re-ingest

End of Documentation

This documentation was generated for the Enterprise RAG System v1.0.0