Analytics Vidhya

# Mastering RAG

## A Guide to FastRAG

```
from fastrag import RAGPipeline
from fastrag.vectorstores import FAISSVectorStore
from fastrag.embedders import HuggingFaceEmbedder
from fastrag.llms import OpenAILLM
```

```
# Set up embedding model
embedder = HuggingFaceEmbedder(model_name="all-MiniLM-L6-v2")

# Set up vector store
vectorstore = FAISSVectorStore(index_path="./my_index", embedder=embedder)

# Load documents and index
vectorstore.add_documents(["RAG improves factual accuracy", "LLMs often
hallucinate."])

# Set up LLM
llm = OpenAILLM(api_key="sk-...")

# Build RAG pipeline
pipeline = RAGPipeline(retriever=vectorstore, llm=llm)

# Run a query
answer = pipeline.query("How does RAG reduce hallucinations?")
print(answer)
```

# What is FastRAG?

FastRAG is an open-source toolkit for building retrieval-augmented generation (RAG) pipelines that are:

- Super fast
- Easy to set up
- Customizable
- Production-ready

It’s designed for developers who want grounded LLMs (less hallucination, more facts) without the overhead of complex frameworks like LangChain or Haystack.

Think: FastAPI + FAISS + HuggingFace + LLMs, stitched together in a clean, minimal way.

# Why FastRAG?

Let’s be real—most RAG frameworks are either bloated or too simplistic. FastRAG fills the sweet spot:

- **Lightweight**: Pure Python, no 10-layer abstraction nightmare.
- **Pluggable**: Choose your embedding model, vector store, retriever, LLM, etc.
- **Evaluable**: Comes with built-in tools to test retrieval + generation quality.
- **Optimized for performance**: Smart batching, async-ready, low-latency by design.

# Core Components

Here’s the basic architecture of FastRAG:

**Ingestion & Indexing**

- Chunk your documents (sliding windows, semantic breaks, whatever you want).
- Embed chunks using a model like:
    - sentence-transformers (local)
    - OpenAI embeddings (cloud)
- Store in a vector DB (default is FAISS, but others like Qdrant and Chroma are supported).

**Retrieval**

- Given a query, embed it and pull top-k similar chunks from your index.
- Hybrid search (dense + keyword) can be added via adapters.

## Generation

- Combine retrieved context with the query.
- Use an LLM (e.g., OpenAI, Mistral, LLaMA2, etc.) to generate the final answer.
- You can even rank / rerank chunks before sending to the LLM.

# Quick Example

Here’s a minimal pipeline using FAISS + HuggingFace + OpenAI:

```
from fastrag import RAGPipeline
from fastrag.vectorstores import FAISSVectorStore
from fastrag.embedders import HuggingFaceEmbedder
from fastrag.llms import OpenAILLM
```

```python
# Set up embedding model
embedder = HuggingFaceEmbedder(model_name="all-MiniLM-L6-v2")

# Set up vector store
vectorstore = FAISSVectorStore(index_path="./my_index", embedder=embedder)

# Load documents and index
vectorstore.add_documents(["RAG improves factual accuracy", "LLMs often
hallucinate."])

# Set up LLM
llm = OpenAILLM(api_key="sk-...")

# Build RAG pipeline
pipeline = RAGPipeline(retriever=vectorstore, llm=llm)

# Run a query
answer = pipeline.query("How does RAG reduce hallucinations?")
print(answer)
```

## Use Cases

FastRAG is a good fit for:

- Internal Knowledge Chatbots
- Document Q&A (PDFs, wikis, etc.)
- Developer Assistants (code search + explanation)\
- Customer Support Tools
- Fact-checking / Research Assistants

# Supported Stack

| Component | Options |
|---|---|
| **Embeddings** | Sentence Transformers, OpenAI, Cohere |
| **Vector Stores** | FAISS (default), Chroma, Qdrant, Weaviate |
| **LLMs** | OpenAI, Anthropic, Local (via HuggingFace Transformers) |
| **Frameworks** | FastAPI integration, async support |
| **Chunking** | Static, semantic, overlap, custom logic |
| **Evaluation** | Precision@k, recall, exact match, human-in-the-loop |

# Built-in Evaluation

One of FastRAG's underrated strengths is that it has evaluation tools baked in:

- Retrieval accuracy (e.g., hit@k)
- Generation quality (BLEU, ROUGE, or manual tagging)
- Compare runs (e.g., different retrievers or LLM prompts)