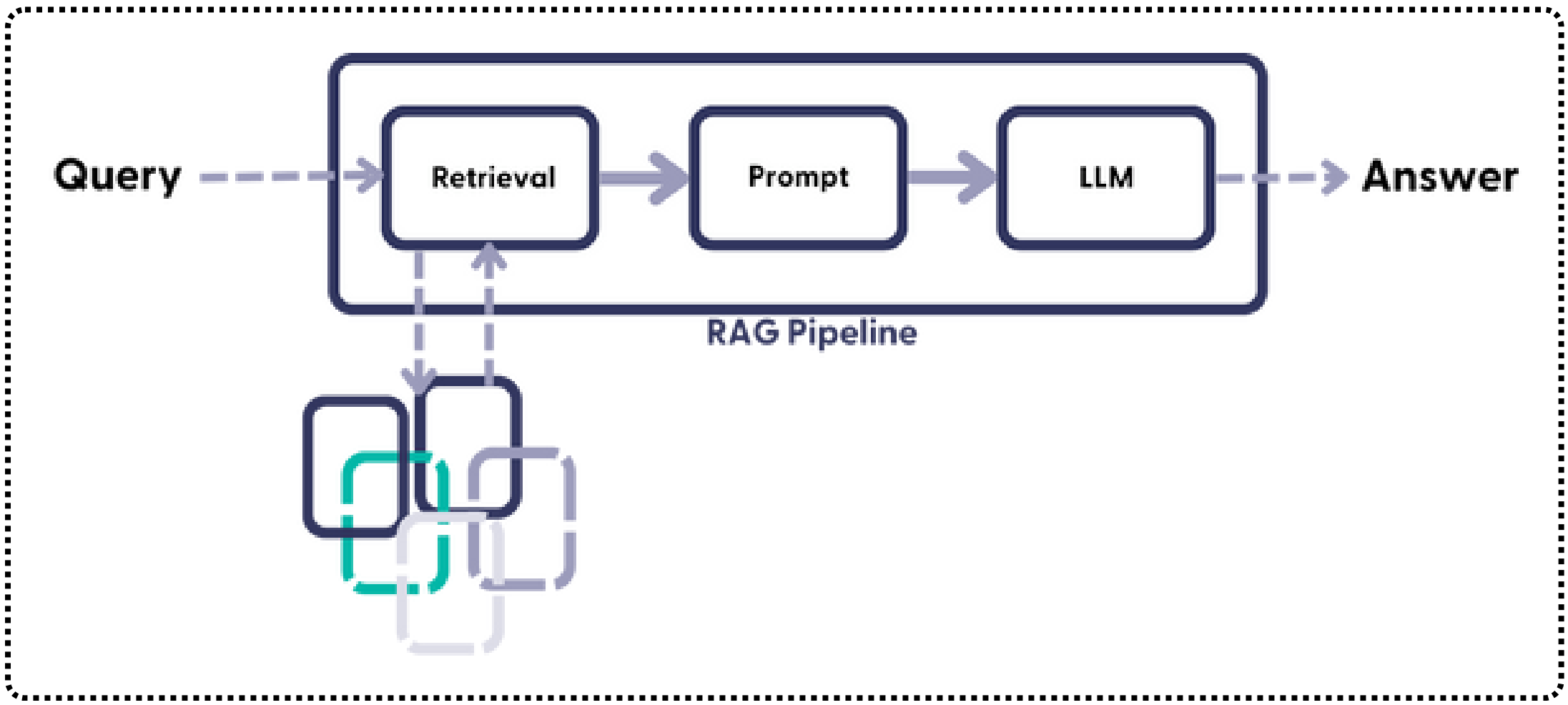
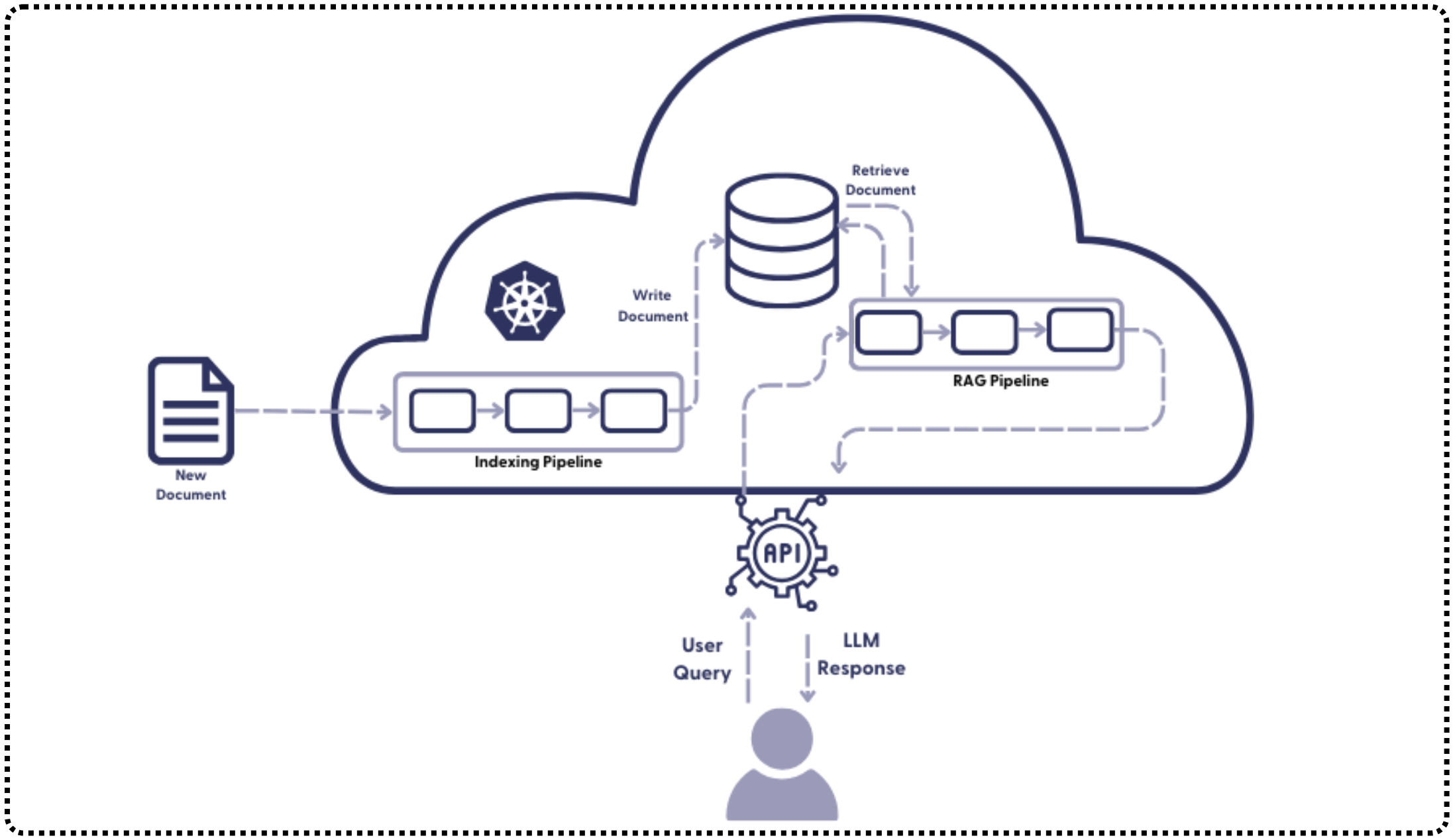
Analytics Vidhya

Mastering RAG

RAG with Haystack





Why Use RAG with Haystack?

- **Improved Answer Accuracy**: RAG enhances generative models with retrieved documents, reducing hallucination.

- **Domain-Specific Adaptation**: Can retrieve information from custom datasets, making responses more precise.

- **Scalability**: Haystack supports distributed processing and various document stores (Elasticsearch, FAISS, Weaviate, etc.).

- **Customizable Pipelines**: Flexible architecture allows for fine-tuning retrieval and generation models separately.

Architecture of RAG with Haystack

A RAG pipeline in Haystack consists of:

- **Retriever**: Fetches relevant documents from a knowledge base.
- **Reader** (or Generator): Processes retrieved documents to generate a response.
- **Pipeline**: Connects components and orchestrates query processing.

Common Retrievers

- **DenseRetriever** (e.g., DPR, SentenceTransformers)
- **SparseRetriever** (e.g., BM25, Elasticsearch)
- **HybridRetriever** (combining dense & sparse retrieval)

Generators for RAG

- Hugging Face's Transformers-based models (e.g., Flan-T5, GPT-2, Llama)
- Custom LLM APIs (e.g., OpenAI, Cohere, Anthropic Claude)

Analytics Vidhya

Setting Up RAG with Haystack

1. Install Haystack

```
pip install farm-haystack[all]
```

2. Load Pretrained Models

```
from haystack.nodes import DensePassageRetriever, FARMReader, Seq2SeqGenerator
from haystack.document_stores import FAISSDocumentStore

# Initialize FAISS document store
document_store = FAISSDocumentStore(embedding_dim=768)

# Use a Dense Retriever
retriever = DensePassageRetriever(document_store=document_store,
query_embedding_model="facebook/dpr-question_encoder-single-nq-
base",passage_embedding_model="facebook/dpr-ctx_encoder-single-nq-
base",use_gpu=True)

# Use a Hugging Face-based Generator
generator = Seq2SeqGenerator(model_name_or_path="facebook/bart-large-cnn")
```

Analytics Vidhya

Step 3. Index Documents

```
from haystack.utils import fetch_archive_from_http, clean_wiki_text,
convert_files_to_docs
from haystack.pipelines import DocumentSearchPipeline

# Load example data
fetch_archive_from_http(url="https://s3.eu-central-1.amazonaws.com/deepset.ai-
farm-models/6.117_TinyData.zip", output_dir="data")

# Convert text files to Haystack documents
docs = convert_files_to_docs(dir_path="data")
document_store.write_documents(docs)

# Update embeddings
document_store.update_embeddings(retriever)
```

4. Build the RAG Pipeline

```
from haystack.pipelines import GenerativeQAPipeline

# Create pipeline
rag_pipeline = GenerativeQAPipeline(generator=generator,
retriever=retriever)

# Query the system
question = "What is the capital of France?"
result = rag_pipeline.run(query=question, top_k_retriever=5,
top_k_generator=2)

# Display result
print(result["answers"][0].answer)
```

Analytics Vidhya

Optimizing RAG in Haystack

1. Using Hybrid Retrieval

```
from haystack.nodes import BM25Retriever, HybridRetriever

bm25_retriever = BM25Retriever(document_store=document_store)
hybrid_retriever = HybridRetriever(dense_retriever=retriever,
sparse_retriever=bm25_retriever)
```

2. Fine-tuning the Generator

- Train the seq2seq model on a domain-specific dataset for better results.

- Use OpenAI API for GPT-based models:

```
from haystack.nodes import OpenAIGenerator
generator = OpenAIGenerator(model="gpt-4")
```

3. Scaling with Elasticsearch

- Use ElasticsearchDocumentStore for large-scale applications.
- Enables efficient full-text search and retrieval.