

Assignment-4 Gen AI

Ques-Combine a retriever and generator to answer factual questions using RAG. Design a semantic search engine using a vector database. Use LlamaIndex to index unstructured data and enable question answering define step by step

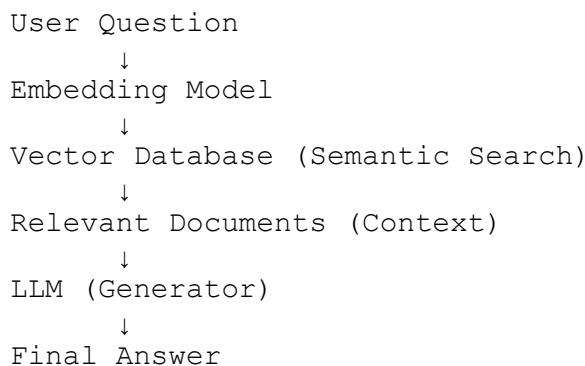
1. What is RAG (High-Level Idea)

RAG = Retriever + Generator

- **Retriever** → Finds relevant documents using **semantic similarity**
- **Generator (LLM)** → Uses retrieved context to generate accurate answers

This reduces hallucinations and improves factual accuracy.

2. Overall Architecture



3. Step-by-Step RAG Pipeline Using LlamaIndex

Step 1: Install Required Libraries

```
pip install llama-index openai chromadb
```

(You can replace **ChromaDB** with FAISS, Pinecone, Weaviate, etc.)

Step 2: Prepare Unstructured Data

Example data sources:

- PDFs
- Text files
- Word documents
- Web pages

```
from llama_index.core import SimpleDirectoryReader  
documents = SimpleDirectoryReader("data/").load_data()
```

- LlamaIndex automatically loads and parses unstructured data.
-

□ Step 3: Choose an Embedding Model

Embeddings convert text → vectors for semantic similarity.

```
from llama_index.embeddings.openai import OpenAIEmbedding  
embed_model = OpenAIEmbedding(model="text-embedding-3-small")
```

□ Step 4: Create a Vector Store (Semantic Search Engine)

Using **ChromaDB** as the vector database:

```
from llama_index.vector_stores.chroma import ChromaVectorStore  
from llama_index.core import StorageContext  
import chromadb  
  
chroma_client = chromadb.Client()  
collection = chroma_client.create_collection("rag_collection")  
  
vector_store = ChromaVectorStore(chroma_collection=collection)  
storage_context =  
StorageContext.from_defaults(vector_store=vector_store)
```

□ Step 5: Index the Documents with LlamaIndex

```
from llama_index.core import VectorStoreIndex  
  
index = VectorStoreIndex.from_documents(  
    documents,  
    storage_context=storage_context,  
    embed_model=embed_model  
)
```

- Now your data is stored as **semantic vectors**.
-

□ Step 6: Build the Retriever

The retriever fetches **top-k relevant chunks**.

```
retriever = index.as_retriever(similarity_top_k=3)
```

□ Step 7: Initialize the Generator (LLM)

```
from llama_index.llms.openai import OpenAI  
llm = OpenAI(model="gpt-4o-mini", temperature=0)
```

- Low temperature = more factual answers.
-

□ Step 8: Combine Retriever + Generator (RAG)

```
from llama_index.core.query_engine import RetrieverQueryEngine  
  
query_engine = RetrieverQueryEngine(  
    retriever=retriever,  
    llm=llm  
)
```

□ Step 9: Ask Questions (Question Answering)

```
response = query_engine.query(  
    "What are the key applications of supervised learning?"  
)  
  
print(response.response)
```

- The LLM answers **only using retrieved documents**.
-

4 □ How Semantic Search Works Here

| Step | Explanation |
|-------------------|--------------------------------|
| Text → Embeddings | Converts meaning into vectors |
| Similarity Search | Cosine similarity in vector DB |
| Top-k Retrieval | Finds most relevant chunks |
| Context Injection | Sent to LLM as prompt |
| Answer Generation | Factual response |

5 □ Why Use LlamaIndex for RAG?

- Handles unstructured data
 - Built-in retrievers & query engines
 - Easy vector DB integration
 - Modular & production-ready
-

6 □ Example Use Cases

- Enterprise knowledge base
 - Research paper Q&A
 - Legal document search
 - Chatbot over internal docs
 - Academic question answering
-

7□Final Summary

Step-by-step RAG with LlamaIndex:

1. Load unstructured data
2. Generate embeddings
3. Store vectors in vector DB
4. Retrieve relevant documents
5. Inject context into LLM
6. Generate accurate answers