Sarvam AI

Assignment: Building a RAG system and an Agent that can perform smart actions based on the user's query

RAG -

A **Retrieval-Augmented Generation (RAG) Agent** is a hybrid model that combines two powerful methods for answering user queries:

1. **Retrieval**: The agent retrieves relevant information from a knowledge base, document set, or vector database.
2. **Generation**: The agent uses a language model (Cohere) to generate a response based on the retrieved information.

A RAG agent aims to improve the quality of answers by combining factual information retrieval and generative capabilities, ensuring that the model's responses are accurate, relevant, and grounded in external knowledge.

How does a RAG work?

1. **Input Query**: The user submits a question.
2. **Retrieval Step**: The agent searches for relevant chunks of text from a document repository or vector database that could help answer the question. These chunks are retrieved based on similarity with the query.
3. **Generation Step**: Once relevant documents or text segments are retrieved, a language model generates a response based on the content of the retrieved information.
4. **Answer**: Then it returns the generated answer, which is more accurate because it is grounded in specific retrieved information rather than purely based on the model's internal knowledge.

Components of an RAG with Agent

1. **Retriever**:
    - Based on the user query, retrieve relevant documents or text chunks from a corpus or knowledge base.
    - Often uses vector databases like **Chroma** that store document embeddings to make retrieval efficient and precise.
2. **Language Model (LLM)**:
    - A pre-trained model (e.g., Cohere, GPT) that can generate natural language answers.
    - The retrieved documents are passed to the language model, which processes the information and generates a coherent answer.
3. **QA Chain**:

    - Combines the retrieval and generation steps.
    - The retrieved documents are used as input for the language model to generate responses in a contextually grounded way.
4. **Tool & Agent**:
    - A **tool** defines the functionality of the RAG chain.
    - An **agent** is a higher-level interface that can use multiple tools and determine which one to use for a given task.

Walkthrough of the Code

Here’s an explanation of how the provided code creates and utilizes an RAG and agent-

- **Loading Documents**: The first step is to load a document (in this case, a PDF) into memory. This document serves as the knowledge base from which relevant information will be retrieved.

```
loader = PyPDFLoader("/content/iesc111.pdf")
docs = loader.load()
```

- **Splitting Documents**: The document is split into smaller chunks of text because trying to retrieve large documents as a whole might lead to inefficient or inaccurate retrieval.

```
text_splitter = CharacterTextSplitter(chunk_size=800,
chunk_overlap=50)

docs_split = text_splitter.split_documents(docs)
```

- **Creating Embeddings**: Next, the code uses **Cohere Embeddings** to convert these document chunks into vector representations. This allows the agent to compare the user’s query with the document chunks efficiently.

```
embeddings = CohereEmbeddings(model="embed-english-v3.0")
```

- **Storing in a Vector Database**: The document embeddings are stored in **Chroma**, a vector database that enables retrieval based on the user's query.

```
vectordb = Chroma.from_documents(documents=docs_split,
embedding=embeddings, persist_directory=persist_directory)
```

- **Setting up the Retriever**: A retriever is set up to handle querying the vector database. It will pull relevant document chunks based on how similar they are to the input query.

```
retriever = vectordb.as_retriever()
```

- **LLM and RetrievalQA Chain**: A **Retrieval-Augmented Generation (RAG) chain** is created using the Cohere language model and the retriever. When the agent needs to answer a question, it will first retrieve relevant chunks of text and then use the language model to generate an answer.

```
llm = ChatCohere()
rag_chain = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",  # "stuff" means using retrieved
documents directly
    retriever=retriever
)
```

- **Creating a Tool**: A **tool** is defined for the agent. The tool encapsulates the functionality of the RAG chain, making it available for the agent to use.

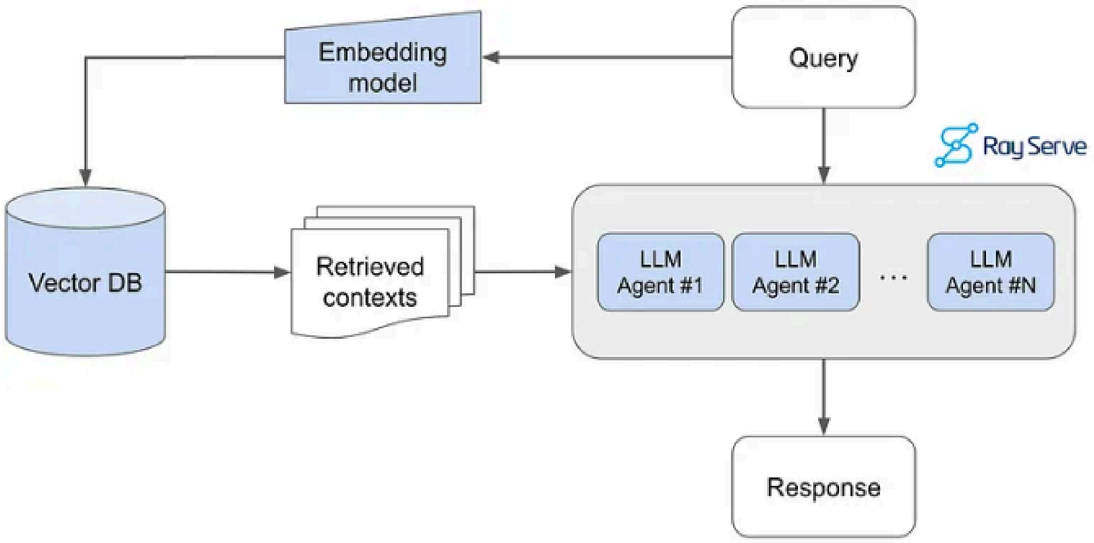
```
rag_tool = Tool(
    name="DocumentQA",
    func=rag_chain.run,
    description="Use this tool to answer questions by
retrieving and using context from documents."
)
```

- **Initializing the Agent**: The **agent** is initialized with the RAG tool. It is a **Zero-Shot React** agent, which means it decides which tool to use based on the description of the task.

```
agent = initialize_agent(
    tools=[rag_tool],
    agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
    llm=llm,
    verbose=True
)
```

- **Querying the Agent**: Finally, the code allows the user to input a query, which is processed by the agent. The agent retrieves relevant documents from the vector database, generates an answer using the language model, and returns it to the user.

```
def query_rag_chain(query):
    response = agent.run(query)
    print(response)
```


Embedding model
Query
Ray Serve
Vector DB
Retrieved contexts
LLM Agent #1
LLM Agent #2
...
LLM Agent #N
Response