**RAG System with Agents: (Railways Annual Report + Train Details)**

**1. Objective:**

The Objective of this Project is to build a RAG Application using “llamaindex” framework along with OpenAI. We are using two data sources one is a PDF File (Railway Annual report 23-24) and another one is a csv dataset (train details), both these data sources are collected from internet from public sites. At the end of the project, we will be having a rag application with agents which can answers queries related to Railways annual report 23-24 data and general train details.

**2. Sources/Datasets used for the Project:**

PDF File:

<https://indianrailways.gov.in/railwayboard/uploads/directorate/stat_econ/2025/Indian%20Railways%20Annual%20Report%20%20Accounts%202023-24%20-English.pdf>

CSV Dataset:

<https://github.com/ayazroomy/Semantic_RAG_Project_Indian_Railways/blob/main/data_set/train_info.csv>

**3. Tools/Frameworks used:**

**- llamaindex:**

I have used llama index it easy and quicker way to load different data sources with minimal set of code and provides different types of query engines and database option to configure and setup easily. Also it supports creating of Agents and workflows.

**- OpenAI:** LLM for making API calls to large language Model.

- **PDF Plumber:** To extract in-depth details from pdf’s tables.

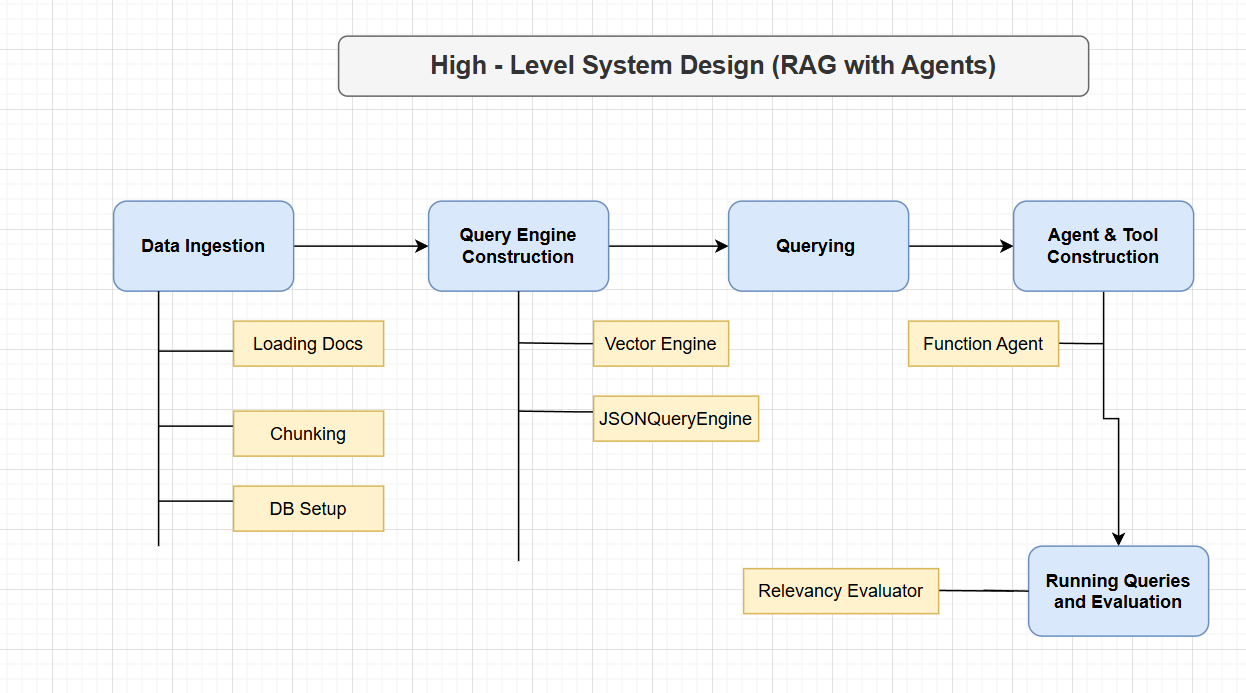
- **Chroma DB:** A Vector database to store the embeddings and to perform semantic search.

- **Fast API:** To Create a Python API Server to host the RAG functionalities.

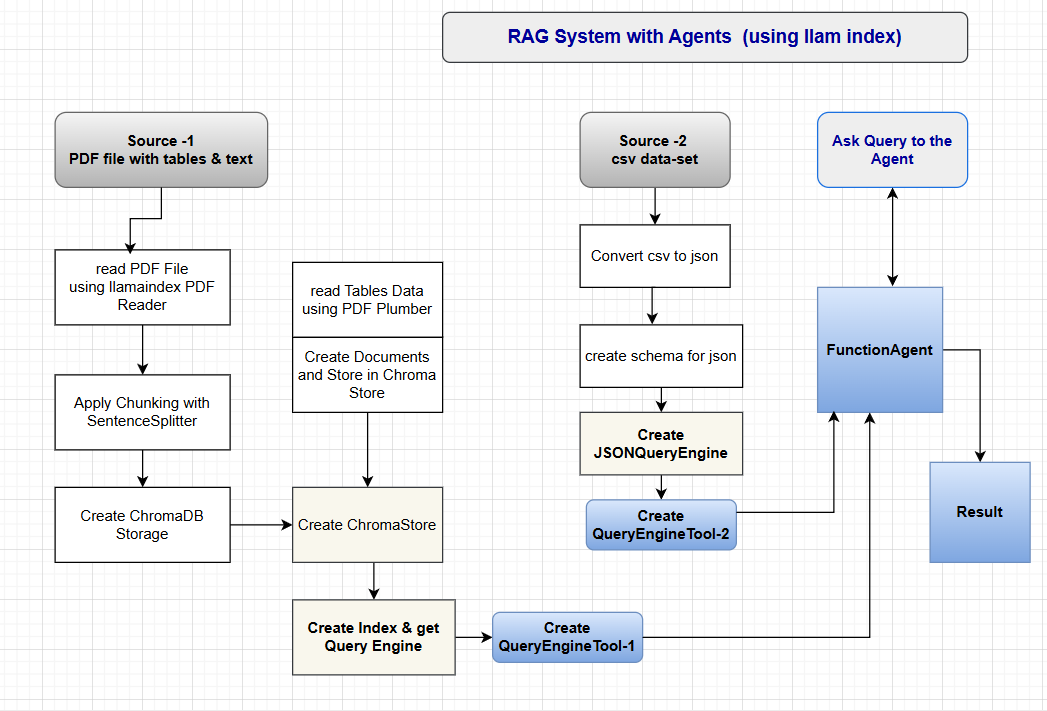
- **Hugging Face:** To Build and Deploy this application in the web.

**4. System Design:**

**a. High -Level Design:**



**b. Application Design / Flow:**



* Step 1: Perform Data Ingestion from PDF Source
* Step 2: Read PDF File using llama index PDF Reader
* Step3: Apply Chunking using Sentence Splitter and Store the chunks in chroma db by converting them to embedding by default OpenAI Embeddings.
* Step 4: Create a Storage Context from ChromDB for further use
* Step 5: Extract Data from tables in the PDF’s using PDFPlumber libraries and store the details in the Array of Documents
* Step 6: Store these Document in the Chroma’DB Storage Context.
* Step 7: Create a Vector Store for indexing using llama’s Vector Store with the storage context
* Step 8: Get a Query Engine from this Index to perform Querying.
* Step 9: Perform Data Ingestion for Source-2 a csv data-set
* Step 10: Convert CSV to a JSON file using json library
* Step 11: Create a JSON Schema for the generated JSON file which we can pass to the JSON Query Engine.
* Step 12: Create a JSONQueryEngine with the json dataset and json schema
* Step 13: Create two QuerEnginesTools using JSONQuery and index query engine.
* Step 14: Create a FunctionAgent and pass these two query engine tools to it.
* Step 15: Perform different querying to the FunctionAgent and check the result.

**5. Implementation:**

Providing step by step implementation details can be too much for this doc , I will try to provide higher -level implementation details how I have built this application below:

**## 1. Data Ingestion & Preprocessing**

**### PDF Data Loading**

- Used `PDFReader` to load the annual report PDF:

  ```python

  reader = PDFReader()

  docs = reader.load\_data("data/Indian\_Railways\_Annual\_Report \_23\_24.pdf")

  ```

**### Text Chunking**

- Split PDF content into chunks for vectorization:

  ```python

  splitter = SentenceSplitter(chunk\_size=500, chunk\_overlap=50)

  nodes = splitter.get\_nodes\_from\_documents(docs)

  ```

**### Vector Database Setup**

- Used ChromaDB for persistent vector storage:

  ```python

  chroma\_client = chromadb.PersistentClient(path="./chroma\_store")

  chroma\_collection = chroma\_client.get\_or\_create\_collection("my\_documents")

  vector\_store = ChromaVectorStore(chroma\_collection=chroma\_collection)

  storage\_context = StorageContext.from\_defaults(vector\_store=vector\_store)

  index = VectorStoreIndex(nodes, storage\_context=storage\_context)

  ```

**### Table Extraction from PDF**

- Used `pdfplumber` to extract tables and convert them to LlamaIndex Document objects:

  ```python

  documents = extract\_tables\_to\_documents("data/Indian\_Railways\_Annual\_Report \_23\_24.pdf")

  nodes\_table = splitter.get\_nodes\_from\_documents(documents)

  index.insert\_nodes(nodes\_table, storage\_context=storage\_context)

**## 2. Query Engine Construction**

**### PDF Query Engine (Unstructured Data)**

- Created a query engine for the PDF content:

  ```python

  query\_engine = index.as\_query\_engine()

  ```

**### Train Info Query Engine (Structured Data)**

- Loaded train info from CSV and saved as JSON:

  ```python

  data\_frames = pd.read\_csv("data\_set/train\_info.csv")

  data\_frames.to\_json("data\_set/train\_info.json", orient="records")

  ```

- Defined a JSON schema for train info and used `JSONQueryEngine`:

  ```python

  train\_info\_schema = { ... }

  nl\_query\_engine = JSONQueryEngine(json\_value=train\_info\_json\_obj, json\_schema=train\_info\_schema)

  ```

**## 3. Querying and Evaluation**

**### Querying the Engines**

- Example PDF query:

  ```python

  response = query\_engine.query("At What Year Indian Railways have conducted full scale disaster management exercise?")

  ```

- Example train info query:

  ```python

  nl\_response = nl\_query\_engine.query("Give the details for the train no 107?")

  ```

**## 4. Agent and Tool Construction**

**### QueryEngineTool**

- Created tools for both query engines:

  ```python

  query\_engine\_tool\_1 = QueryEngineTool.from\_defaults(query\_engine=nl\_query\_engine, ...)

  query\_engine\_tool\_2 = QueryEngineTool.from\_defaults(query\_engine=query\_engine, ...)

  ```

**### Agent Setup**

- Used `FunctionAgent` to combine tools and LLM:

  ```python

  agent = FunctionAgent(tools=[query\_engine\_tool\_1,query\_engine\_tool\_2], llm=OpenAI(model="gpt-4o"))

  ctx = Context(agent)

  ```

**### Running Queries via Agent**

- Example of running a query and streaming results:

  ```python

  handler = agent.run("Give the details for the train no 108?", ctx=ctx)

  async for ev in handler.stream\_events():

      # ... handle events ...

  response = await handler

  ```

**## 5. Evaluation**

- Used `RelevancyEvaluator` to evaluate responses from the annual report query engine:

  ```python

  from llama\_index.core.evaluation import RelevancyEvaluator

  eval\_result = evaluator.evaluate\_response(query=query, response=response)

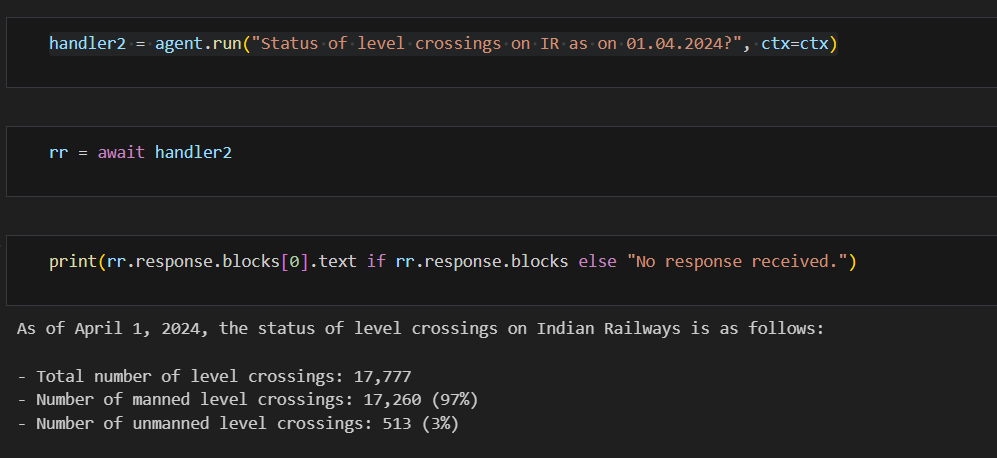
  ```

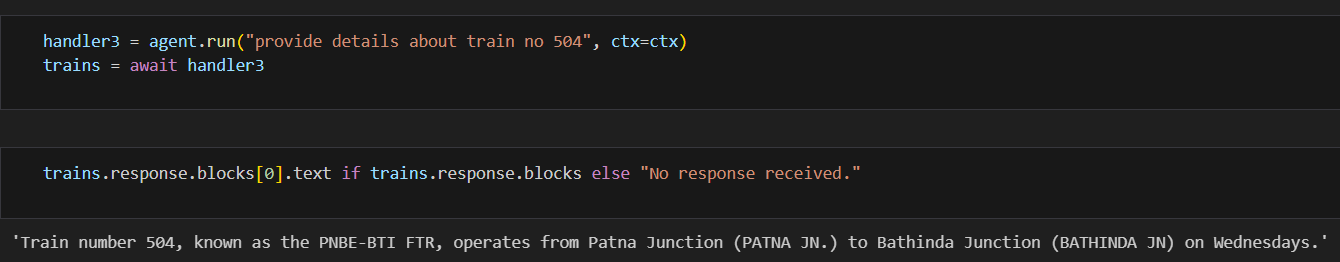
**6. Query and Evaluation:**

**Running Some Queries:**

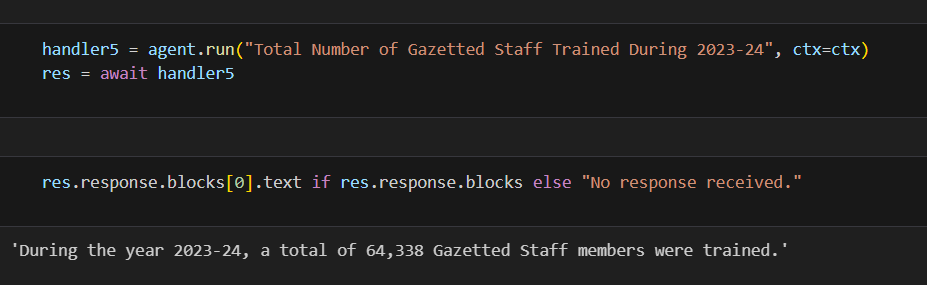
**-------------------------------**

**1.**



**2.** 

**3.**



**Evaluation:**

**------------------**

Perform Different type of Querying and do a “RelevancyEvaluation” to the Vector QueryEngine.

# define evaluator

evaluator = RelevancyEvaluator()

your\_eval\_dataset = ["Ahmednagar - New Loni - Ashti belongs to which state?",

                     "What is New Bongaigaon- Kamakhya via Rangiya project length and anticipated cost?",

                     "Total Number of Gazetted Staff Trained During 2023-24?"]

# query index

for query in your\_eval\_dataset:

    print(f"Evaluating Query : {query}")

    print('\*'\*50)

    response = query\_engine\_tool\_2.query\_engine.query(query)

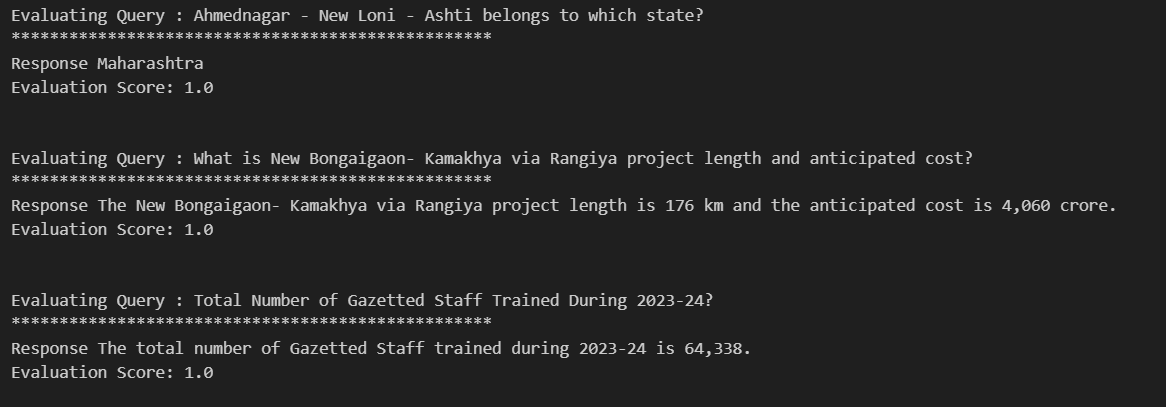
    eval\_result = evaluator.evaluate\_response(query=query, response=response)

    print('Response', response)

    print('Evaluation Score:',str(eval\_result.score))

    print('\n')

**Result:**



**7. Lesson Learned and Challenges:**

* Identify the right type of queryEngine from llamIndex to do the job , llamaindex many queryengines.
* Choose the Right Agent to setup the Agents , llamaindex provides FunctionAgent, reactive Agent and others.
* FunctionAgent always works with context , so having one context across the application is must to provide better result.
* Llamaindex multiple Routing can also be used to setup the different query engines tool in case if the FunctionAgent does not work well.
* Perform Evaluation to make sure the result are getting good.

**8. Deployment to Cloud:**

* This application has been served using FastAPI server and deployed into the Huggingface cloud environment.
* Demo: <https://ayazroomy-my-rag-space-railway.hf.space/webapp/index.html>

**9. Git Hub Repo :**

Please visit this GitHub Repo:

<https://github.com/ayazroomy/Semantic_RAG_Project_Indian_Railways/tree/main>

For more details on the Project , please check the project readme.md file to get more details.