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.

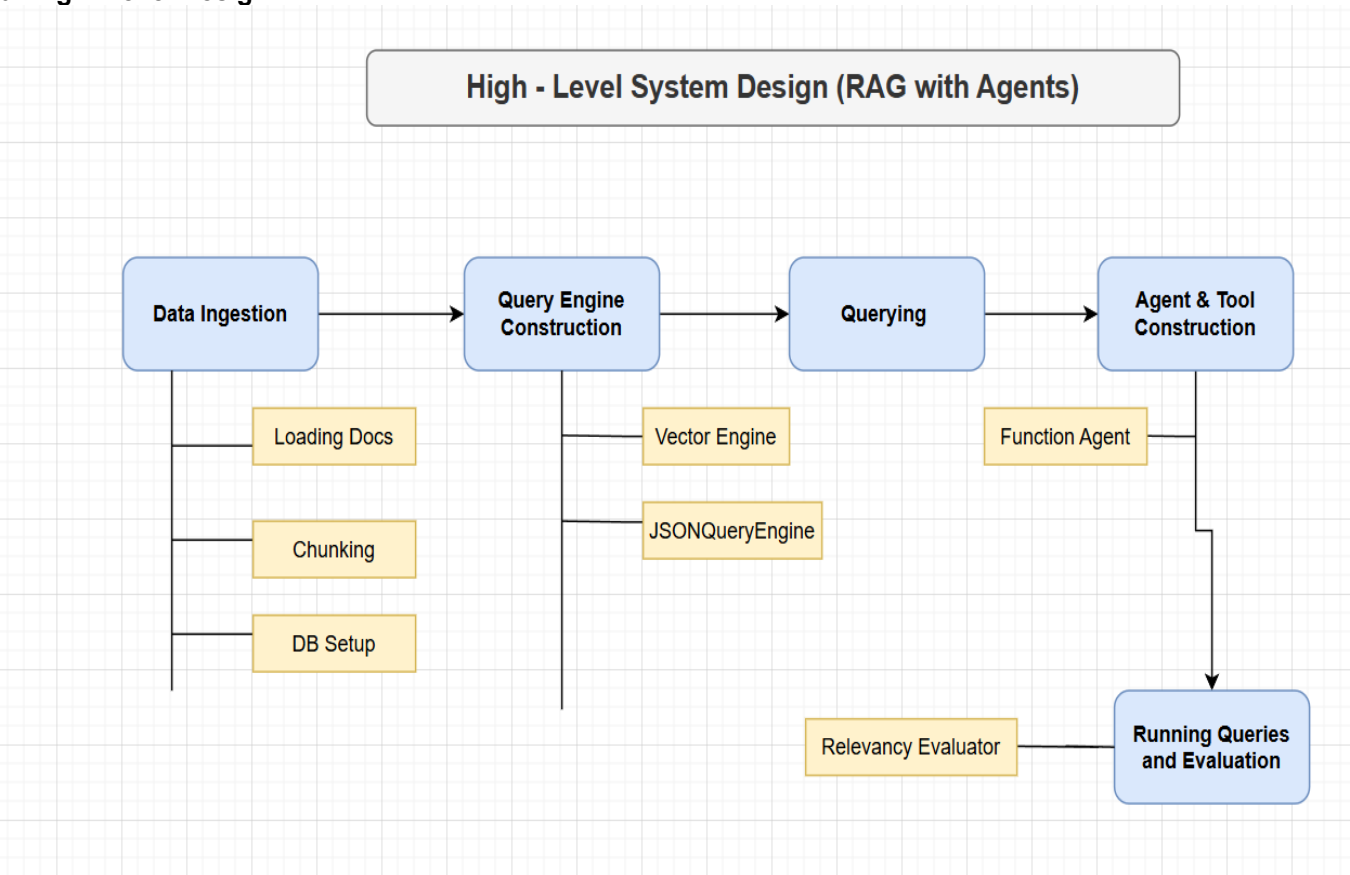
- **Chroma DB:** A Vector database to store the embeddings and to perform semantic search.

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

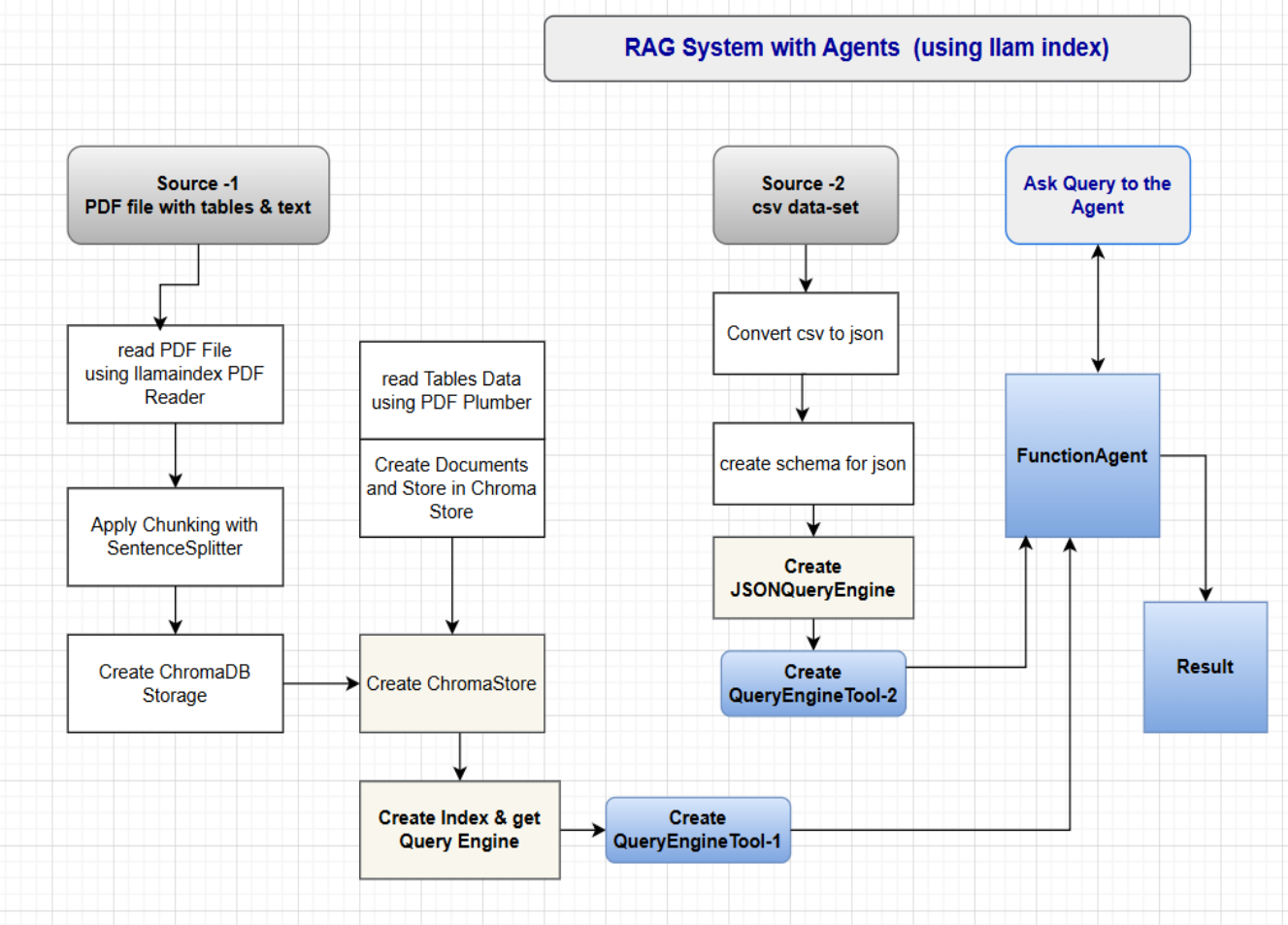
- **Hugging Face:** To Build and Deploy this application in the web.

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.

```
handler2 = agent.run("Status of level crossings on IR as on 01.04.2024?", ctx=ctx)
```

```
rr = await handler2
```

```
print(rr.response.blocks[0].text if rr.response.blocks else "No response received.")
```

```
As of April 1, 2024, the status of level crossings on Indian Railways is as follows:

- Total number of level crossings: 17,777
- Number of manned level crossings: 17,260 (97%)
- Number of unmanned level crossings: 513 (3%)
```

2.

```
handler3 = agent.run("provide details about train no 504", ctx=ctx)
trains = await handler3
```

```
trains.response.blocks[0].text if trains.response.blocks else "No response received."
```

```
'Train number 504, known as the PNBE-BTI FTR, operates from Patna Junction (PATNA JN.) to Bathinda Junction (BATHINDA JN) on Wednesdays.'
```

3.

```
handler5 = agent.run("Total Number of Gazetted Staff Trained During 2023-24", ctx=ctx)
res = await handler5
```

```
res.response.blocks[0].text if res.response.blocks else "No response received."
```

```
'During the year 2023-24, a total of 64,338 Gazetted Staff members were trained.'
```

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:

```
Evaluating Query : Ahmednagar - New Loni - Ashti belongs to which state?
**************************************************
Response Maharashtra
Evaluation Score: 1.0


Evaluating Query : What is New Bongaigaon- Kamakhya via Rangiya project length and anticipated cost?
**************************************************
Response The New Bongaigaon- Kamakhya via Rangiya project length is 176 km and the anticipated cost is 4,060 crore.
Evaluation Score: 1.0


Evaluating Query : Total Number of Gazetted Staff Trained During 2023-24?
**************************************************
Response The total number of Gazetted Staff trained during 2023-24 is 64,338.
Evaluation Score: 1.0
```

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.