Assignment - 1

Link to code - https://github.com/shivam8496/AccuNox-assesment

Q1- API Data Retrieval and Storage: You are tasked with fetching data from an external REST API, storing it in a local SQLite database, and displaying the retrieved data. The API provides a list of books in JSON format with attributes like title, author, and publication year.

Ans - I wrote the code for this problem assuming that , Multiple books can have same Author and also one Book cloud also have multiple authors, Hence making a **Many-to-Many** (m*n) cardinality relationship. And for this , managed 3 tables -

1-Books(id,name,publication_year) primaryKey(id)
2-Author(id,name) primaryKey(id)
3-book_author(book_id,author_id)
- foreignKeys(book_id,author_id)

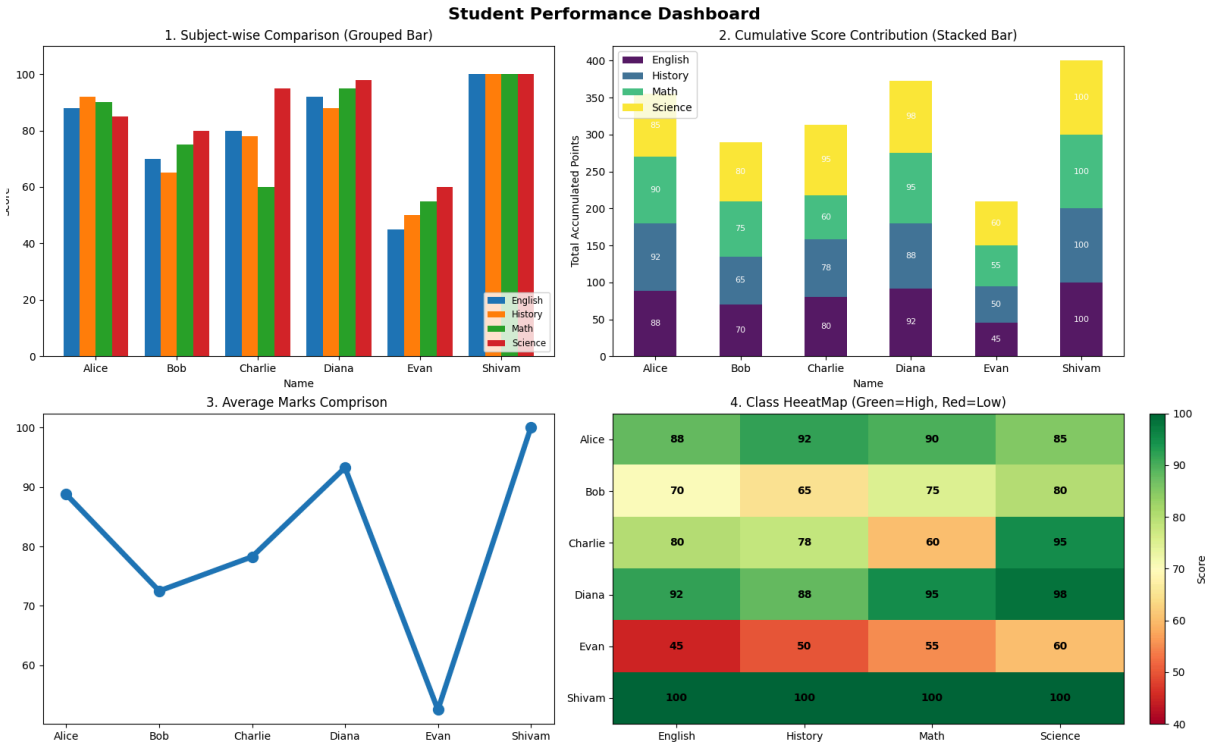
This code could also be done Using ORMs(Object relation Mapping) like **sqlAlchemy** , but since it is a simple implementation problem I used vanilla sql commands

Q2- Data Processing and Visualization: Given a dataset containing information about students' test scores, fetch the data from an API, calculate the average score, and create a bar chart to visualize the data.

Ans - Here I assumed the that the data of each of the student is coming subject wise , so that I can create various variations of plotting graphs, In Total I created 4 Graphs -

1-Grouped BarGraph
2-Stacked BarGraph
3-Simple Line Graph
4-HeatMap Graph

Graphs Eg -



Q3- CSV Data Import to a Database: Write a Python script that reads data from a CSV file containing user information (e.g., name, email) and inserts it into a SQLite database

Ans - Here also I Relational database , I assumed the CSV data from Will come the given from -

```
RowID,name,email,role,department,street,pincode
0,User_0,user_0@company.com,Dev,Sales,s1,18774
1,User_1,user_1@company.com,Dev,Sales,s2,767686
```

So I stored this data in three tables , In order to reduce the Data Redundancy ,

- **Employee**
- **Departments**
- **Address**

But , sometimes CSV could be larger in size containing larger datasets(100k-1M). Loading and saving all Data at once could Crash device, So I used CSV streaming technique to load data in chunks, Avoiding the system OverHead

Q4- Send a link to the most complex Python code you have written

- Link - https://github.com/shivam8496/RAG-Pipeline
- You could find the code explanation in the **README.md** file

Q5- Send a link to the most complex database code you have written

- Link - https://github.com/shivam8496/Milvus-Database
- Code Explanation in **README.md** file

Assignment - 2

Q1- Where would you rate yourself on (LLM, Deep Learning, AI, ML). A, B, C [A = can code independently; B = can code under supervision; C = have little or no understanding]

Ans - LLM - [A - Can Code independently]
Deep Learning - [A - Can Code independently]
AI - [A - Can Code independently]
ML- [A - Can Code independently]

Q2- What are the key architectural components to create a chatbot based on LLM? Please explain the approach on a high-level

Ans - 3

1-User Interface

- This is the part where we take the actual user input. There are many ways we could expose the input endpoint of a chatbot like, WebApplication, Mobile app, Chatting applications integration like WhatsApp and Slack extension . But for now I would take Web application as an example
- So the main purpose of this layer would be taking input from the user , Handling message Streaming , Calling APIs and displaying the output .

2-Backend

- This is the part where API calls are sent. This will be the Central controller , It manages and orchestrates all the interaction between UI , the LLM call and external Data Sources and other auxiliary services.
- Tasks it would do is :
    - Authentication and Authorizations
    - Handling Multiple users asynchronously
    - Users wise session and conversation state management
    - Prompting and context injection

- Calling external LLM APIs
- Logging , error Handling and Monitoring

There are multiple services that run in Backend

1- Prompt Engineering

- A LLMs output would be as good as a prompt would be .
- We can add predefined System prompts to LLMs so that they reply in specific tone , with defined constraints by following specific persona
- Also we can make dynamic prompts with the users input to enhance the LLMs output.
- We can add COT(Chain of thought) Technique to design the prompt.

2- Calling LLMs

- We could use LLM APIs like OpenAI APIs , Gemini APIs. Since these are the best LLM Models present in market

3- Retrieval-Augemented Generation (RAG)

- This is the most important architectural component for building a domain-specific chatbot
- Since a LLMs does not have information for our specific data, we need to specifically inject our own data with the prompt to keep LLMs to hallucinate and give required output
- In this approach:
  - User input is converted into vector embeddings (using tools like SentenceTransformers)
  - Relevant document or chunk are retrieved from the vector databases like Pinecone , Milvus etc
  - (Optional) Since vector Databases are not enough because they find similar words but fails to find the connected components .So we could use Knowledge Graph database like **Neo4j** to make LLM understand the relationship between components

4- Adding of MCP(Model Context Protocol) tools

- LLMs are not only limited to conversation alone
- We can Utilize the MCP services to make LLMs do the different tasks like :

- Database queries
- Scheduling or booking operations etc

5- Management of state and memory

- To maintain multi-turn conversations , memory management is essential
- We can add:
  - Short-term memory for recent conversation turns
  - Long-term memory: persistent user preference or history

6- Monitoring , logging and FeedBack Loop

- A AI Chatbot should always improve, in order to do that we need to continuously monitor our system , to find the bottlenecks and error (Logging is essential here) , So that we can improve it later

High-Level System Flow

The overall workflow of an LLM-based chatbot can be summarized as follows:

1. User submits a query via the UI
2. Backend receives and validates the request
3. Context and prompt are constructed
4. Relevant data is retrieved if required
5. LLM generates a response
6. Output is post-processed and returned to the user
7. Logs and feedback are stored for analysis

Q3- Please explain vector databases. If you were to choose a vector database for a hypothetical problem (you may define the problem) which one will you choose, and why?

Ans -

Vector Database- A vector database is a type of database which stores the data with the semantic meaning of it , These semantic meanings are called **Embeddings** . In Traditional databases we query the database and based on the exact match we get results. These databases fail whenever we want to search not the exact , but some similarity based pointers . Here comes the Vector Databases , we can query these databases by passing the **Vector embedding .** And define the similarity thresholds and based on that we could get our results . These Vector embedding are generally generated using Embedding generated Models like (e.g. text-embedding-3-large , Cohere's Embed v4) . There is a dedicated python framework called **SentenceTransformer** for embedding generations . We can simply give raw user input and the model will generate embeddings accordingly.

Problem Statement -

I am assigned to make a RAG System , for a law firm which has more than 500M internal legal Documents . To help lawyers with legal queries.

Things To keep in mind -

1-Since It is a law firm , all the data is confidential (cannot use any cloud based Database)

2-Hybrid search , Layers need to search by semantic query and strict filters as well

3-Scale and efficiency, Since Database is very large we need faster query responses.

Solution -

I will use **Milvus Database** for this scenario , and these are the reasons

1- Unlike **PineCone**(which is Saas) , Milvus is Open source and we can deploy it by ourselves , and literally create our own infrastructure (Since it is Kubernetes based) . Making our database completely private

2-Milvus is Specially designed to handle large datasets, Also there is separated logic for Storage and Computing . Meaning I can scale Searching nodes independently of the storing nodes . And for our 500M datasets , this distributed architecture could get the results in MilliSeconds

3-Not only semantic search we can also do the precise filtering of metadata like in Normal SQL databases

4- It could be expensive for small scenarios , but at such a big scale in our scenario instead I would save us money, because managed services like Pinecone (Which charges vector stored/hour) , could become so expensive at scale. And with milvus I could pay for only infrastructure that i provision