Accuknox Internship Assignment

Problem Statement 1 -

Q1/A1 – My Understanding – So this task dives deep in API handling i.e. fetching data from an external REST API by request-response methodology and database usage i.e. storing the retrieved data that is in JSON format (i.e. dictionary form) in the SQLite database. For this piece of code I am actually retrieving data from

external REST API called “Open Library Search”.

Note- The coding part I have written in Italics to differentiate from other text

a. Importing the needed libraries

```
import requests
import sqlite3
```

b. Defining the base URL

```
base_url=https://openlibrary.org/search.json?q=
```

c. Defining the function to retrieve book data from the API

```
def retrieve_book_data(name):
    url=f"{base_url}{name}"
    response=requests.get(url)
    if response.status_code == 200:
        return response.json()
    else:
        print(f"Failed to retrieve data {response.status_code}")
```

d. Creating the database and the table inside it that will store the data

```
def createDatabase():
    connection = sqlite3.connect("Books_Data.db")
    cursor = connection.cursor()
    cursor.execute(""" CREATE TABLE IF NOT EXISTS Books_Info (
                  book_id integer PRIMARY KEY
AUTOINCREMENT,                    title text,
                  author text,
                  publication_year integer)"""
    connection.commit()
  return connection
```

<u>e. The function to store the retrieved info of book as per our query in the database but only the title, author and publication year</u>

```
def store_book(connection, book_info):
   cursor = connection.cursor()
 # Get the first book from the docs list
   first_book = book_info["docs"][0]
   title = first_book.get("title", "Unknown")
   authors = ", ".join(first_book.get("author_name",
["Unknown"]))    year = first_book.get("first_publish_year",
0)

   cursor.execute(""" INSERT INTO Books_Info (title,
author,              publication_year)
                 VALUES (?,?,?)""", (title, author,
            year))
   connection.commit()
```

<u>f. Creating the function to display the ‘Books_info’ table</u>

```
def display_Books_info(connection):
    cursor = connection.cursor()
    print("The books in the database are - ")
        for book in cursor.execute("SELECT * FROM
Books_Info"):          print(f" Id : {book[0]}, Title : {book[1]},
Author : {book[2]},          Publication_Year : {book[3]}")
```

g. Finally, the main function which calls all the functions for given query

```
def main():
   name="xyz"
   book_info=retrieve_book_data(name)
   connection = createDatabase()
   store_book(connection, book_info)
   display_Books_info(connection)
   connection.close()

if __name__ == "__main__":
   main()
```

Q2/A2 - Well we have to retrieve the data of students from external API (will be done with the help of ‘requests’ library) and calculate the average of their scores and plot or make the visual representation of the data using matplotlib library.

Note - The size of some of the text have been intentionally reduced to adjust the code Here is my code for the same

a. Importing the libraries-

```
  import requests
 from statistics import mean
 import matplotlib.pyplot as plt
```

b. Fetching the data from external API-

```
  def retrieve_students_data(url):
   response=requests.get(url)
   if (response.status_code==200):
     return response.json()
   else:
     print(f"Failed to retrieve data
     {response.status_code}")
```

c. Calculating the average of the scores of students-

```
  def calculating_average(students):
      all_scores=[score for score in students['score']]
      avg_score=mean(all_scores)
      return avg_score
```

d. Visualizing the data-

```
  def plot_scores(students):
   names=[name for name in students['name']]
   scores=[score for score in students['score']]
   plt.figure(figsize=(10,10))
   plt.bar(names, scores, color='skyblue')
   plt.title("Students Test Scores")
   plt.xlabel("Student Name", loc='center')
   plt.ylabel("Score", loc='center')
   plt.ylim(0, 100) # I am assuming scores are out of
100    plt.show()
```

e. Calling the main function

```
def main():
url="https://api.slingacademy.com/v1/sample-data/users"
      students=retrieve_students_data(url)
       print(f"The average score
is  {calculating_average(stude
nts):.2f}")          plot_scores(students)


if __name__ == "main":
  main()
```

Q3/A3 - Here is the code for that

Note - The size of some of the text have been intentionally reduced to adjust the code.

```
import csv

import sqlite3

def database_creation():
   connect = sqlite3.connect("database1.db")
   cursor = connect.cursor()
   cursor.execute(""" CREATE TABLE IF NOT EXISTS
employees_table (              employee_id integer PRIMARY KEY
AUTOINCREMENT,              name text,
            email text,
            designation text)""")
   connect.commit()
   return connect

def reading_csv_file(filepath):
   employees = []
   with open(filepath, newline='', encoding='utf-8') as
csvfile:      data = csv.DictReader(csvfile)
      for row in data:
         employees.append((row["name"], row["email"], row["designation"]))
   return employees

def insert_employees(connect, employees):
  cursor = connect.cursor()
  cursor.executemany(""" INSERT INTO employees_table (name, email,
designation)              VALUES (?,?,?)""", employees)
  connect.commit()
```

```
def display_employees(connect):
  cursor = connect.cursor()
  cursor.execute("SELECT * FROM employees_table")
  employees_data = cursor.fetchall()
  print("The employees in my organistaion are - ")
  for data in employees_data :
    print(f" Id : {data[0]}, Name : {data[1]}, Email : {data[2]}, Designation : {data[3]}")

connect=database_creation()
all_employees=reading_csv_file("xyzfilepath.csv")
insert_employees(connect, all_employees)
display_employees(connect)
connect.close()
```

4. The most complex python code that I have written is I think the project of 'Hierarchical Classification of Conversation data' along with my teammates as part of my curriculum in IIT Hyderabad/Talent Sprint.

Here is link for that -

https://github.com/Roshan-Kujur/AIET-Cohort-6-IITH-Learning-Phase/blob/2d16a58c6157e69112a8defcb0f19c81f50794d1/AIET-Project-ClassificationOfConversationaldata-Practice.ipynb

5. So I have created 'Library management system' to showcase my knowledge of databases and SQL but I have not got the chance to work on large database systems, but through this small project I want to demonstrate my basic skills in SQL, I'm actively improving my database skills and would be eager to work with more complex systems during the internship.

Link - https://github.com/Roshan-Kujur/Library-Management-Database

Problem Statement 2 –

Q1/A1 – B = Can code under supervision

Q2/A2 – Chatbots based on LLMs are of great value. Most of the businesses tend to build their own chatbot system that will help their customers. The brain of these chatbots are Large Language models whose foundations are Artificial Neural Networks and they are trained on very huge amount of data. Examples of LLMs are Open AI's GPT, Google's Gemini, Meta 's Llama model, etc. LLMs generate the next word based on the contextual meaning of the latest word.

The key architectural components to build chatbot based on LLM are –

1. User Interface Layer(Client Layer) – So User interface is the layer that is present between the users and the working system and it helps the users to interact with the brain of the chatbot i.e. LLM and the response (answers to our questions) are also displayed in the Interface only. It comes with lot of features like we take the example of ChatGPT then in it's interface we have chat box for typing our queries , we have mic that helps to record our queries and which then converts to textual content, they also manages sessions and helps to organize our chats, keeps records of our chats, etc.

2. Request Handling/ Backend Layer – Well this component is the entry point to the chatbot system. It has lot of work to do, this system identifies whether the input/request is coming from a valid user or not, it authorizes the user, whether the request is valid ensuring the appropriateness of the query/prompt , whether the query can be accepted at that point of time or not on the basis of rate limit and load, what conversation or session does the query belong to and very importantly normalizes the prompt by cleaning its textual content transforming it into clean, structured input that can be consumed by the next layer. And finally this layer passes a context-enriched object. This

layer ensures the system receives clean, structured, and safe input, preventing errors and security risks.

3. Context & Prompt Management Layer - It is very crucial layer, it is like the teacher/instructor to the LLM. It controls what information the LLM gets and how it is instructed. It executes the System prompt that defines the role and behaviour of chatbot, like for eg, giving it prompt as ‘You are a customer support assistant for an e-commerce platform’, it also stores the previous messages and selecting which past messages to get included in the each new prompt. This layer does Prompt Composition and Formatting and filters the irrelevant messages. The business value and guardrails are enforced in this layer to ensure the chatbot behaves predictably and aligns with product or organizational requirements.

d. LLM(The Brain) - Large Language Models or LLM are the AI models that are Artificial Neural Networks in base and are meant for Natural Language Understanding and Generation, they are the founding stones of Generative AI. They are built by training the ‘Transformers’ which have encoding-decoding layer architecture and works on self-attention mechanism on huge amount of textual data like almost everything on data and has very large number of parameters. LLM generate text based on the contextual meaning. Overall, The LLM is the core intelligence layer of a chatbot system. It is responsible for understanding natural language input, reasoning over the provided context, and generating coherent, human-like responses. LLM captures the Semantic meaning of different words. How the LLM system works is as

1. Firstly the input is tokenized that is given word,
sentence , paragraphs, etc are broken into smaller tokens that is subwords or words.
2. Then these tokens are converted to numerical representations that is into high-dimensional vectors which are called word embeddings and

they help the LLM to capture Semantic meaning i.e. similar words or words with same context will have similar vectors.

3. Then the core computation happens in the LLM based on Self Attention Mechanism and and finally the output is produced.

We can integrate LLM in our chatbot system in two ways-

i. By hosting API based LLM - We can host the models through there APIs which can be open source or closed source like Open AI's GPT, or Google Gemini or Anthropic, etc and can call the LLM via REST or SDK, it is fast way to integrate.

ii. By self hosting LLM - We can deploy the open-source models like (Mistral, Claude,etc) on our GPU or cloud for which we will have full control, and the data will stay internally and also can be cost-efficient in terms of long term.

There are number of frameworks that helps to build chatbot systems like Langchain, LlamaIndex, Microsoft Semantic Kernel, etc but I would prefer Langchain as it has large ecosystem that can help us to build end to end Generative AI applications and is widely used.

e. Retrieval and Knowledge Layer - This is optional layer but I consider it is very important to build Retrieval Augmented Generation(RAG) as it help to build such robust chatbot system that not only is limited to the data the LLM has been trained on but has the capability to retrieve necessary information from internet sources or internal data or any data source. It also helps to provide factual, real-time, up to date knowledge. It's components include document store like the database, APIs, PDFs where the data is stored, the embedding model which is responsible for generation of word embeddings, Vector Database like FAISS, Pinecone, Weaviate, Chroma,etc which are responsible for storing the vector representation of the data from various data sources and it provides the necessary context to the LLM whenever required and the retriever .

f.Tool / Function Calling Layer - This layer allows the chatbot to call APIs whenever required like when to call a tool with what parameters , it is also tasked with performing some calculations, querying databases and much more. LLM may incorporate the results it gets through the APIs in their response. Chatbot system can perform actions like sending email, schedule meetings with some person or book tickets,etc. The retrieval systems that can be used are ElasticSearch, Pinecone, Weaviate, etc.

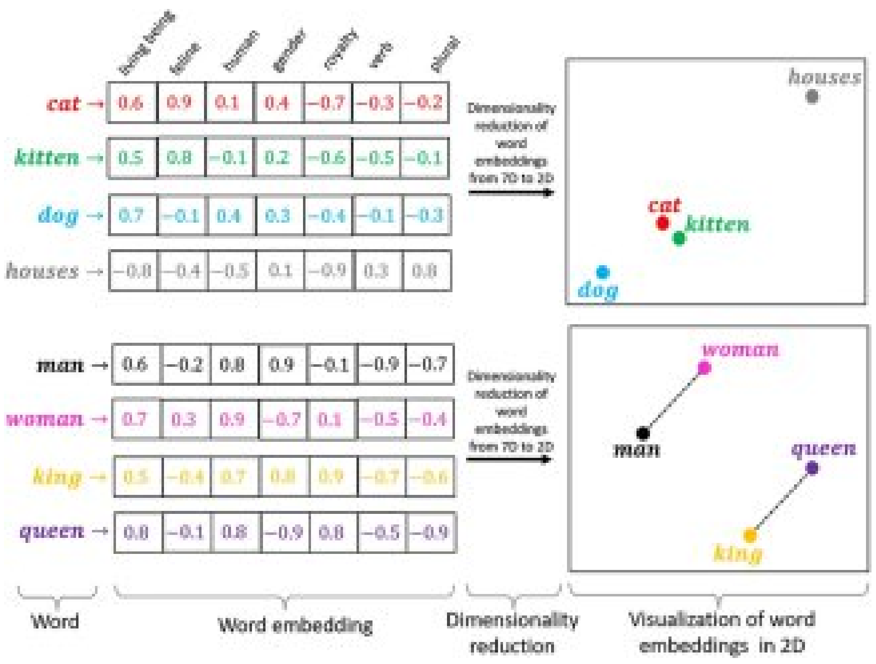
g. Response Postprocessing, Safety and Moderation Layer - Response Postprocessing layer refines LLM outputs by performing tasks like Formatting, Content Filtering, personalization, etc. While Safety and Moderation Layer, ensures safe, ethical, and reliable chatbot behavior. It filters out offensive, harmful, or biased content and enforces domain specific guardrails. Some moderation tools that can be used are OpenAI Moderation API, Perspective API,etc.

h. Backend & Orchestration Layer - This layer sits between the user interface and the LLM/tools. This layer acts like the “manager” of the chatbot system. It coordinates all parts of the system like it manages data flow, handles scaling and orchestrates calls to the LLM, external tools, and databases. The key functions of this layer includes Request Handling, Context & Session Management, LLM & Tool Orchestration, Error Handling & Scalability, Security & Logging.
Some of the tools used are -

1. API - FastAPI, Flask, Node.js
2. For session storage - Redis, PostgreSQL
3. For Scaling - Kubernetes, Docker
4. For Monitoring - ELK Stack, Grafana

i. <u>Monitoring & Feedback Loop Layer</u> - This layer is meant for continuous improvement of the chatbot. It tasks include track performance metrics (response accuracy, latency, user satisfaction), collect feedback to fine tune prompts, model parameters, or workflows,etc

<u>Q3/A3</u> – Vector Databases are the databases that store the data in the form of their vector representation i.e. mathematical vector embeddings. So, diving deep into more basics that actually why we need vector databases like for chatbot system based on LLM or some tasks involving Natural Language Generation then the one of the core logic involved is finding the contextual meanings of words by having the contextual embeddings. Text, pictures or videos in AI systems are converted to embeddings that are in numerical forms for computations. So each words or tokens are represented as vectors where each entries are numerical values that are computed on the basis of similarities or dissimilarities with different words. So whenever the context of one word is to be found out the algorithm goes for semantic search that is finding the similarities or dissimilarities with the other words based on similarity or dissimilarity in the vectors by the help of distance metrics. So overall the vector representations of the words given the words are high numbered needs to be stored in databases, but in traditional RDBMS they can be stored but whenever there is need of semantic search it would be very hectic for the algorithm to do so and would be very time consuming.
Vector databases come with this very feature that they store the word embeddings by indexing them, which reduces unnecessary comparisons at the time to semantic search and helps in saving time hence is very efficient.



cat
kitten
dog
houses
man
woman
king
queen
Dimensionality reduction of word embeddings from 7D to 2D
Word
Word embedding
Dimensionality reduction
Visualization of word embeddings in 2D

fig-How word embeddings look

Like for eg, applications like chatGPT(chatbot system equipped with RAG) have with them a vector database that actually contains the embeddings of lots of words which have been collected through external docs, websites, pdfs,etc. Now when user gives some prompt to the chatGPT then it is converted to embeddings and the similarity search happens between the prompt embeddings and the embeddings in vector databases and the relevant context is retrieved and sent along with the prompt(context enriched) to the LLM to reason over it.

Some of the vector databases are Pinecone, Weaviate Cloud, Zilliz Cloud (Milvus Cloud), Qdrant, Chroma, FAISS,etc.

Suppose I want to build a chatbot system for an e-commerce platform

Then its work should be

1. Handling user queries on real time based on like huge amount of product descriptions, manuals, reviews and past chat logs,etc.
2. Must filter product by type/category, region, language,etc 3. Can scale easily as new products are added always.

I will prefer Pinecone as -

1. It is cloud-based so it can handle huge number of vectors without the need of any complex infrastructure.

2. It will provide us high-performance ANN search valuing the time of the customers for real time conversation as it will use HNSW indexing which will ensures fast semantic retrieval.
3. It provides Hybrid Search Support(Vector+keyword) so filtering by metadata like product type, region, etc will be easy for accurate responses.
4. It can handle scaling, replication, etc making it reliable for customer facing applications and with the help of it we can easily integrate Python or REST APIs with the chatbot backend that help with tasks like ordering the product, switching an alarm that can inform us the availability of the product,etc.