Documentation

API Data Retrieval and Storage

In this task i worked with a simple REST api which gives book data in json formate. I created a small flask api in the file **book_api.py** which returns a list of books having title, author and year details. After that i used another python file **api_to_sqlite.py** to call this api using the requests library.

The data which comes from the api is first converted into json and then stored into a local sqlite database named **books.db**. If the database or the table is not already present then it gets created automatically while running the code. After storing the book records into database, i again fetched the data from the same table and printed it on terminal just to check whether the data is saved properly or not.

Data Processing and Visualization

In this part i worked with student test score data. I created a small api in the file **prob1_datavisual1.py** which returns student name and their total score from the database. This api sends the data in json format.

Using another python file **prob1_datavisual2.py**, i fetched this data from the api using requests. After fetching the data, i stored the student names and scores in two seperate lists. Then i calculated the average score of all students using simple python logic.

To visualise the scores, i used matplotlib and created a bar chart showing each student and their total score. I also added a horizontal line to show the average score so that it becomes easy to compare individual scores with the average.

CSV Data Import to Database

For this task i created a csv file named **userrs_listt.csv** which contains user information like name and email. After that i wrote a python script **prob1_3.py** which reads the csv file line by line using the csv module.

The data read from the csv file is then inserted into a local sqlite database named **userrss.db**. If the database or the table is not already present then it gets created automatically while running the script. After execution of the code all the records from the csv file gets stored into the database successfully.

Most Complex Python Code

The most complex python code I have written is related to my previous project which involved api handling and data processing. I have shared the github link of that project below:

https://github.com/bhavisha016/Food_Chat_Bot

Most Complex Database Code

For database related work I have mostly used sqlite and mysql in my projects. The database handling code used in this assignment and previous projects includes table creation insertion and fetching data. I have shared my github profile link where database related code can be found.

https://github.com/bhavisha016

Self Rating on LLM, Deep Learning, AI, ML

If i have to rate myself honestly, then i would say the following.

For **Machine Learning**, i would rate myself as **A**. I am comfortable with core ml concepts like supervised and unsupervised learning, regression, classification and basic model building. I can code ml models on my own and understand how the algorithms work internally at a basic level.

For **Deep Learning**, i would rate myself as **B**. I understand concepts like neural networks, cnn, backpropagation and loss functions, but for complex architectures i sometimes need guidance or reference material. I can code under supervision and slowly improve with practice.

For **AI**, i would rate myself as **B**. I understand what ai systems try to achieve and how different components like ml, dl and reasoning come together, but i am still learning how to design complete ai systems independently.

For **LLM**, i would rate myself as **B**. I understand how llms work at a high level, how they are trained and how they are used in applications like chatbots, but for advanced fine tuning or architecture level changes i still need guidance.

Key Architectural Components of an LLM Based Chatbot

To create a chatbot using a large language model, there are few main components which work together to give proper responses to the user. I am explaining this in high level based on my understanding and learning.

The first component is the **user interface**. This is the part where the user actually interacts with the chatbot. It can be a website, mobile app or even a simple chat window. The user types a question or message and waits for the response from the system.

The second component is the **backend layer**. This layer takes the user input and prepares it to be sent to the language model. It also handles small things like storing conversation history, checking input formate and sometimes removing unwanted input. This layer is important because it connects the user side with the intelligence part of the system.

The next main component is the **LLM itself**. This is a pretrained large language model like GPT or similar models. The model processes the input text and generates a response based on the data and patterns it has learned during training. The LLM does not directly know everything, but it predicts the best possible answer based on probability and context.

In many practical chatbot systems, there is also a **retrieval component**. This part is used when the chatbot needs information from external sources like documents, pdf files or databases. The retrieval system finds relevent information and sends it to the LLM so that the response becomes more accurate and less hallucinated.

After the LLM generates the response, a **post processing layer** is used. This layer cleans the output, removes unnecessary text and formats the response properly before showing it to the user.

Overall, the approach is to combine user input, backend logic, LLM reasoning and sometimes external knowledge to generate meaningful and helpful chatbot responses.

Vector Databases Explanation and Selection

A vector database is a special type of database used to store data in the form of vectors or embeddings. These vectors are numerical representation of text, images or other types of data. Vector databases are mainly used to find similarity between data instead of exact matching.

In simple words, vector databases help in finding content which is similar in meaning even if the exact words are not same. This is very useful in applications like chatbots, semantic search and recommendation systems.

For a hypothetical problem, suppose i want to build a chatbot which answers questions from a large collection of pdf documents like class notes or user manuals. In this case, all the

documents are first converted into embeddings using an embedding model. These embeddings are then stored inside a vector database.

When a user asks a question, the question is also converted into an embedding and compared with stored embeddings to find the most similar documents. The relevent content is then passed to the LLM to generate the final answer.

For this type of problem, i would choose **FAISS** as the vector database. The main reason is that FAISS is open source, fast and easy to use with python. It is suitable for learning, experimentation and small to medium scale projects. It also performs similarity search efficiently which is required for chatbot use cases.

If the system needs to scale for very large production level systems, then other vector databases can also be considered, but for a learning and academic use case, FAISS is a good choice.

BHAVISHA TEKCHANDANI

Lucknow, Uttar Pradesh, India
■ bhavisha016@gmail.com | ■ 9026573337
GitHub: https://github.com/bhavisha016 | LinkedIn: https://www.linkedin.com/in/bhavisha-tekchandani-b55418278/

PROFILE

AI & Full-Stack Engineering aspirant with hands-on experience building machine learning systems, backend APIs, and data-driven applications. Strong foundation in Python, applied machine learning, computer vision, NLP, and backend development using FastAPI. Experienced in converting ideas into working systems with real users, clean code, and production-oriented thinking.

EDUCATION

Bachelor of Technology (B.Tech) – Computer Science & Engineering
Amity University, Lucknow
CGPA: 9.4 | Batch: 2026

TECHNICAL SKILLS

Programming: Python, Java, C, C++
Backend & Full-Stack: FastAPI (REST APIs), backend–frontend integration, basic API design, input validation & error handling, Dialogflow, Streamlit, Basic React, HTML, CSS
Machine Learning: Supervised & Unsupervised Learning, feature engineering, model training & evaluation, performance metrics
Deep Learning & Computer Vision: Neural Networks, CNNs, image classification, image recognition, preprocessing, augmentation, OpenCV (basic)
NLP: Text preprocessing, TF-IDF, Word2Vec (applied), emotion detection, NLP pipelines
Statistics & Data: Descriptive statistics, EDA, distributions, outlier analysis
Tools: Pandas, NumPy, SQL, MySQL, Git, GitHub, Docker (basic), Matplotlib, Seaborn, Scikit-learn, TensorFlow

PROJECTS

Food Ordering & Tracking Chatbot
GitHub: github.com/bhavisha016/Food_Chat_Bot | Live: lambent-cassata-701efb.netlify.app
• Built an intelligent chatbot using FastAPI and Dialogflow for food ordering and order tracking.
• Integrated MySQL database and deployed frontend on Netlify.

Potato Leaf Disease Classification
GitHub: github.com/bhavisha016/Potato_leaf_disease_classification
• Developed a CNN-based computer vision model for crop disease classification using real datasets.

INTERNSHIPS

Data Science Intern – Renu Sharma Healthcare & Educational Foundation
April 2025 – July 2025
• Worked on data analysis, machine learning concepts, and healthcare datasets.

Intern – Infosys Springboard
August 2025 – October 2025
• Built an Emotion Detection & Music Recommendation System using ML & NLP techniques.

ACHIEVEMENTS & ACTIVITIES

• 2nd Place – Code-Swap Competition
• Research Intern – Multispectral Crop Disease Detection
• IEEE Volunteer | Peer Mentor