Problem statement- assignment 2

1. **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]**

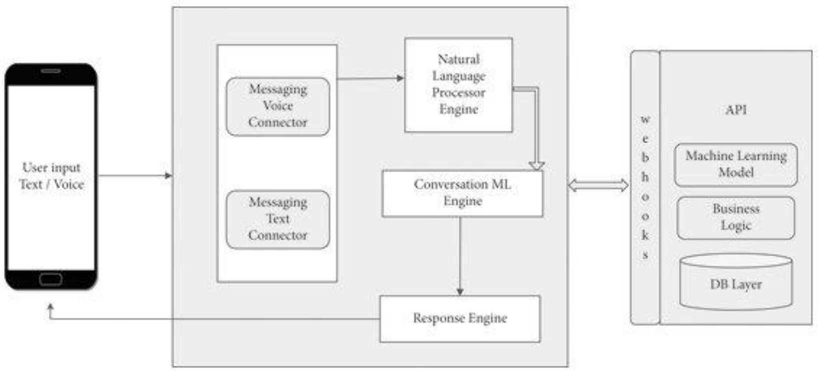
Ans1)

Area	**Rating**	**Explanation**
Artificial Intelligence (AI)	**B**	I understand core AI concepts and applications and can work under guidance.
Machine Learning (ML)	**B**	Familiar with basic ML workflows, data preparation, model usage, and evaluation, but still building hands-on depth.
Deep Learning	**C**	Conceptual understanding of neural networks; limited hands-on implementation so far.
Large Language Models (LLM)	**C**	Early-stage understanding; learning concepts like prompts, RAG, and agents, but not yet hands-on at production level.

Overall, my strengths are in **Python, data handling**, and understanding how data flows through systems. I pick up new tools and concepts quickly by experimenting and reading documentation, and I'm comfortable learning through hands-on practice. I may not have deep experience in AI or LLMs yet, but I put consistent effort into improving and asking the right questions. With proper mentorship and real-world exposure, I am confident I can grow into deeper AI and LLM work and contribute effectively to the team.

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

Ans2)



When creating a chatbot using a **Large Language Model**, I think of it as a pipeline rather than a single model doing everything. At a ***high level***, the system is made up of multiple components that work together to understand the user's input, decide what information is needed, and generate a meaningful response.

The process starts with a user interface, which could be a ***web chat, mobile app, or API endpoint.*** This is simply where the user types or speaks their query. The goal of this layer is only to collect input and pass it forward.

Next comes input processing. Here, the user's message is cleaned and formatted so the model can understand it properly. This may include adding prompt instructions, handling different input formats, or maintaining basic conversation rules. This step ensures consistency before the query reaches the model.

In many practical systems, there is an optional but very important step called context retrieval, often implemented using **Retrieval-Augmented Generation (RAG)**. In this step, the user's query is converted into an embedding and matched against a vector database to find relevant documents or knowledge. These retrieved results are then passed to the language model as additional context. This approach is useful because it allows the chatbot to work with domain-specific or updated information instead of relying only on what the model was trained on.

The **LLM** inference layer is the core of the chatbot. This is where the language model takes the user query along with any retrieved context and generates a response. The model could be accessed through a hosted API or deployed internally, depending on system requirements.

After the model generates a response, post-processing is applied. This includes formatting the output, validating the response, and applying safety or policy checks. This step helps ensure that the final answer is appropriate and usable.

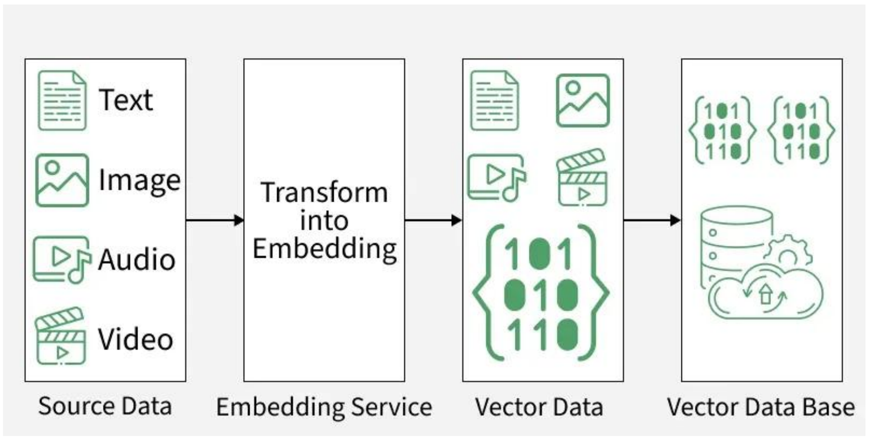
Finally, the response delivery layer sends the processed output back to the user through the same interface they used to ask the question.

Overall, this modular architecture makes the chatbot easier to scale and improve over time. Individual components like the LLM, retrieval system, or user interface can be updated or replaced without redesigning the entire system.

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

Ans3)

A vector database is a specialized database designed to store and search high-dimensional vector representations *(embeddings)* efficiently.



A **vector database** is used when we want to search data based on meaning rather than exact words. Unlike traditional databases that work well with structured data and exact matches, vector databases store numerical representations (*embeddings*) of data and allow similarity-based search.

In simple terms, when *text, documents, or other data* are converted into embeddings, each item becomes a vector that captures its semantic meaning. A **vector database** stores these vectors and helps find the closest matches when a new query comes in. This is especially useful when different words can have similar meaning, which is difficult to handle using normal SQL queries.

Vector databases are commonly used in scenarios like semantic search, recommendation systems, and chatbots that need to retrieve relevant context before generating a response. They play a key role in **Retrieval-Augmented Generation (RAG)**, where relevant documents are fetched and passed to a language model to improve accuracy and reduce hallucinations.

Hypothetical problem

For example, suppose we are building a chatbot for a company that needs to answer questions using internal documents such as security policies, logs, and technical documentation. These documents may be large, frequently updated, and written in different formats. In this case, keyword-based search would not be sufficient, because users may ask questions in many different ways.

Here, a vector database would allow us to convert both user queries and documents into embeddings and retrieve the most semantically relevant content, even if the wording is different.

Vector database choice and reasoning

For this type of problem, I would choose a vector database that is easy to integrate, performant, and flexible. During early development, I would prefer a managed or easy-to-deploy vector database so that the focus stays on building the application logic rather than managing infrastructure.

An open-source option like **Qdrant** is appealing because it is lightweight, supports metadata filtering, and works well with modern LLM workflows. It allows storing vectors along with additional information, which is useful when filtering results based on document type, date, or source.

Overall, the choice of vector database depends on the stage of the project, but for a learning or early production setup, simplicity, performance, and ease of integration are more important than advanced optimizations.