Problem Statement 2

My Self-Rating on LLM, Deep Learning, AI, and ML

When I look at my skills honestly, I would put myself somewhere in the middle for most of these topics. I'm not a complete beginner, but I'm also not at a level where I can independently build advanced systems without guidance. So my ratings are mostly "B", and here is how I justify them:

LLM — B

I can work with LLMs in a practical sense (like using APIs, writing prompts, calling the model inside an application).
I understand concepts like embeddings and retrieval, but I haven't trained or fine-tuned an LLM by myself. So I would say I am comfortable experimenting, but I still need supervision for deeper architectural decisions.

Deep Learning — B

I have built basic neural networks before, mostly for image or text classification tasks.
I can modify model layers, tune hyperparameters, and run training loops, but usually by referring to existing examples or guidance.
I wouldn't say I can design complex architectures from scratch, so B feels right.

AI (general understanding) — B

My understanding of AI is broad — I know different subfields (NLP, CV, reinforcement learning) and I've tried small projects in a few of them.
However, I still rely on tutorials or references when it comes to implementing larger AI systems.
So again, somewhere between beginner and independent.

Machine Learning — B

Out of all four areas, ML is the one I'm most comfortable with.
I can independently work with datasets, clean data, train models like logistic regression, decision trees, random forest, etc.
I can also evaluate results and tune models without much help.
Still, I won't call myself "A" because I haven't done very large or highly optimized projects yet.

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

1. The LLM (Large Language Model)

This is the main intelligence of the chatbot or in simple terms "brain"..
It interprets what the user is saying and generates a meaningful reply.
Most developers don't train their own LLM because it's extremely expensive, so they usually use:

- API models like GPT, Claude
- open-source models like Llama, Mistral
- or smaller quantized models running locally

So the LLM is more like the "core intelligence" but not the entire system.

2. Retrieval Component (only if external knowledge is needed)

If a chatbot must answer questions from private files (PDFs, policies, product manuals), then the LLM alone can't do it.
For that, we use **Retrieval-Augmented Generation (RAG)**.

The retrieval pipeline usually includes:

- breaking documents into chunks
- creating embeddings for each chunk
- storing these embeddings in a **vector database**
- converting the user's question into an embedding
- retrieving the most similar chunks
- sending those chunks along with the question to the LLM

This keeps the chatbot factual and reduces hallucinations.

This retrieval part is exactly what the yellow gear in your diagram represents (managing context + external info).

3. Memory / Context Management Layer

LLMs cannot remember long chats by themselves.
So the system needs a memory layer that decides:

- how many recent messages to keep
- whether to summarise old messages
- what to store as "long-term" memory

Short-term memory is kept inside the prompt.
Long-term memory can again be stored using a vector database.

This is what makes the chatbot feel consistent in longer conversations.

4. System Prompts / Instruction Templates

Before sending the user's query to the LLM, the system adds instruction templates such as:

- the assistant's behaviour
- safety rules
- formatting guidelines
- previous messages or summaries

This part is shown in your diagram as the **Template box** interacting with the user input.

5. Orchestration Layer (the glue)

Tools like **LangChain**, **LlamaIndex**, **Haystack**, or even a custom pipeline sit here.

This layer decides things like:

- when to call the LLM
- when to retrieve extra knowledge
- how to mix user input + memory + system rules
- how to format the final response

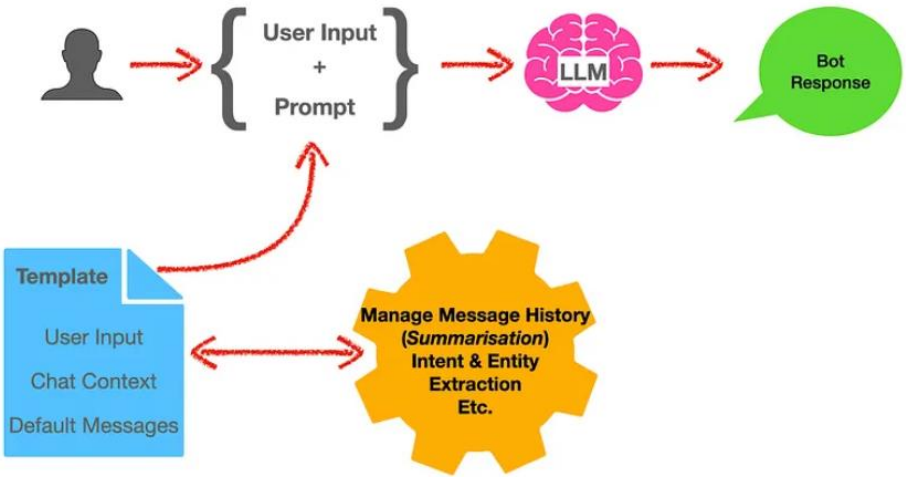
Basically, it manages the entire flow.

6. Application Layer (UI Integration)

Finally the chatbot needs to be exposed somewhere:

- website
- mobile app
- WhatsApp bot
- Slack integration
- or simple API endpoint

This part only handles communication with the user.


User Input
+
Prompt
LLM
Bot
Response
Template
User Input
Chat Context
Default Messages
Manage Message History
(Summarisation)
Intent & Entity
Extraction
Etc.

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?

A vector database is basically a database that stores information in the form of vectors.
Vectors are long lists of numbers that come from embedding models.
The idea is that when two pieces of text have similar meaning, their vectors will also be close to each other. So instead of matching exact words, the database can match **meaning**, which normal SQL databases cannot do.

This is useful when a user asks a natural-language question and the system has to find the most relevant answer from documents, PDFs, or notes.
Because the search is based on similarity between vectors, it can pick up related concepts even if the words don't match exactly.

Hypothetical Problem I am considering

Suppose I want to build a customer support chatbot for Bajaj, where customers can quickly get answers about their bike issues without calling the service center every time.

Customers usually ask things like:

- "My bike is not starting in the morning, what should I check first?"
- "How do I claim warranty for my Bajaj model?"
- "Which service includes brake pad replacement?"
- "What should I do if the engine overheats during long rides?"
- "How can I book a service appointment or find the nearest service center?"

The information for these questions is available in:

- customer support FAQs
- warranty and service policy documents
- troubleshooting guides
- service manuals
- dealership service checklists

But the problem is that these documents are written in a very formal and technical way, and users ask questions in simple, everyday language.
So a plain keyword search isn't enough because the user's query may not match the exact words used in the manual.

By converting the documents into **vectors** and storing them inside a vector database, the system can retrieve the content that *matches the meaning* of the query, even if the wording is different.

For example,
User says: **“Bike overheats after 20 minutes, why?”**
Manual says: **“Engine temperature may rise due to low coolant level or restricted airflow.”**
A vector database can connect these two automatically because the meanings are related.

Which Vector Database I would choose ?

Ans. I would choose **ChromaDB**.

Reasons:

1. **Very easy to set up**
   I don’t have to manage servers or difficult configurations. It works directly in Python.
2. **Free and open source**
   As a student, I prefer tools that don’t require cloud accounts or billing.
3. **Good enough for small to medium projects**
   My hypothetical Bajaj chatbot wouldn’t need millions of documents, so Chroma is more than enough.
4. **Integrates well with LLM frameworks**
   Especially tools like LangChain, which makes building the retrieval pipeline easier.
5. **Lightweight**
   It runs on a normal laptop without any special hardware.