## Sujal Dhandre

## 1)Self-Assessment Across AI/ML Domains

### Machine Learning (ML): A

I would rate myself **A** in Machine Learning, as I can independently design, implement, train, and evaluate machine learning models. I have a clear understanding of core ML concepts such as data preprocessing, feature engineering, model selection, evaluation metrics, and optimization techniques. While I may use generative AI tools for productivity or reference, I am fully capable of writing ML code independently and understanding the underlying logic and trade-offs involved in model development.

### Deep Learning (DL): B

I would rate myself **B** in Deep Learning. I have worked on multiple deep learning projects and possess a solid conceptual understanding of neural networks, including architectures such as CNNs. However, for more complex architectural design choices, hyperparameter tuning, or advanced optimization strategies, I perform best with guidance or supervision. I am comfortable implementing deep learning models but continue to strengthen my expertise in designing and optimizing them independently.

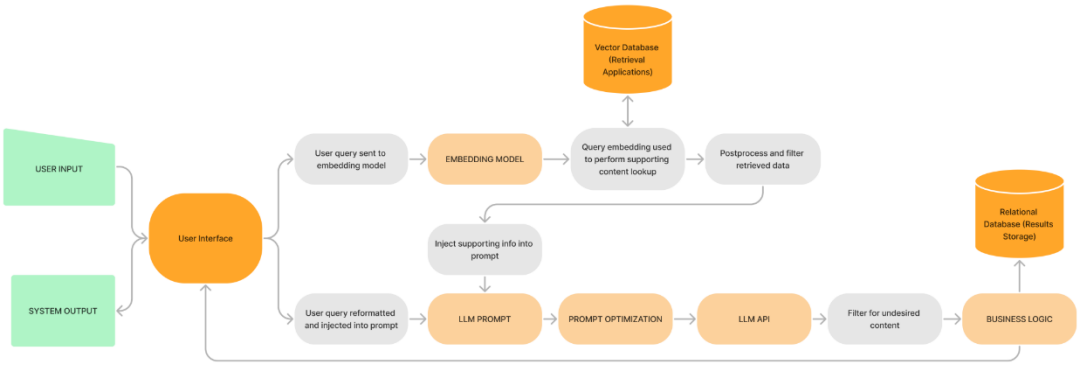
### Large Language Models (LLMs): C

I would rate myself **C** in Large Language Models. I have recently started learning about LLMs and related concepts such as Retrieval-Augmented Generation (RAG), vector databases, and frameworks like LangChain. While I have a foundational understanding of how these components work together at a high level, I am still in the early stages of gaining hands-on experience and cannot yet implement complete LLM-based systems independently. I am actively learning and building toward deeper practical expertise in this area.

## 2)Key Architectural Components to Create a Chatbot Based on LLM

While working on this topic, I researched extensively across multiple online resources to understand how modern LLM-based chatbots are architected. This included reading technical blogs and articles available on the internet, using tools like ChatGPT to clarify and validate concepts during my learning process, and referring to a detailed LinkedIn post that explained the end-to-end system design of LLM-based applications. The explanation below is based on my understanding and interpretation of these sources, combined with my own reasoning about how the different architectural components work together in real-world LLM chatbot systems.

Through this research, I realized that an LLM-based chatbot is fundamentally different from a simple API-based chatbot where user input is directly sent to a language model and the response is returned. In contrast, an LLM-based chatbot follows a layered architecture in which the language model is only one component within a larger system designed to handle context, knowledge, control, and reliability.



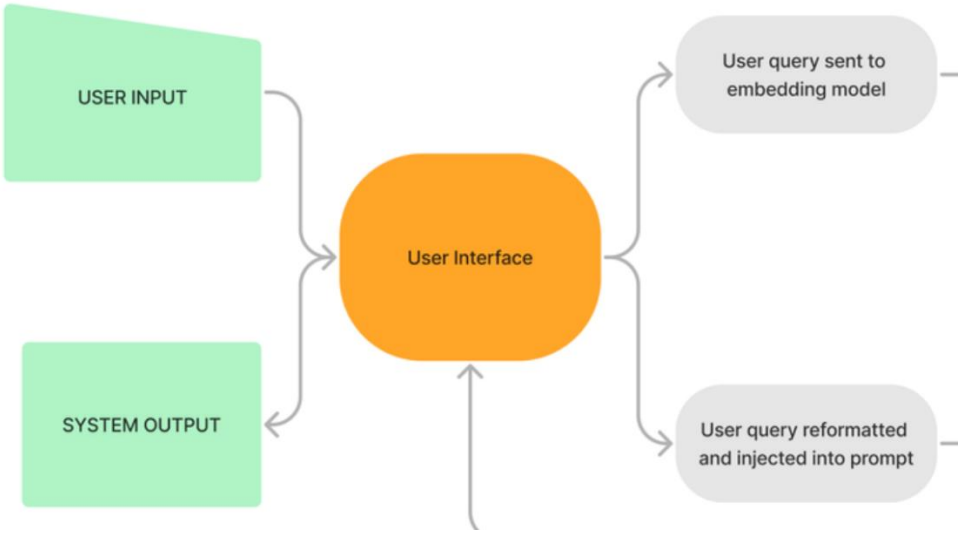

**Source(I have taken it from a LinkedIn post):**https://www.linkedin.com/pulse/architecting-llm-systems-chatbot-services-intelligent-tarun-thummala-2bvie/

### 1. User Input Layer

The architecture begins with the user input layer, where users interact with the chatbot through a web interface, mobile application, or API endpoint. At this stage, the system only captures the user's input. A key distinction is that the input is not immediately forwarded to the LLM, as doing so would result in unreliable and context-poor responses.

### 2. Pre-processing Layer

User input is often unstructured, incomplete, or grammatically incorrect. The pre-processing layer is responsible for cleaning, normalizing, and structuring this input. It helps handle incomplete queries, formatting inconsistencies, and ambiguity, ensuring that the input is suitable for further processing. This step improves overall response quality and reduces unnecessary noise before the input reaches the language model.

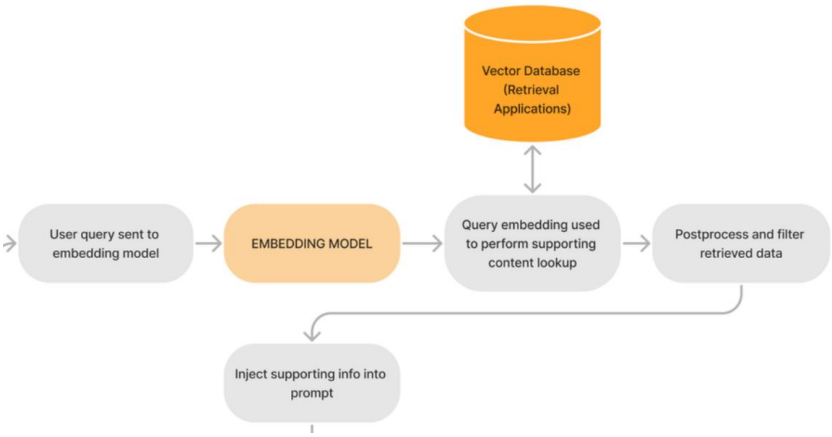



### 3. Context and Conversation Management

Large Language Models are stateless by design and do not retain memory across interactions. To support multi-turn conversations, a context and conversation management component is required. This layer manages session state and selectively includes relevant past interactions to maintain conversational continuity. At the same time, it ensures that only necessary context is included to avoid exceeding token limits.

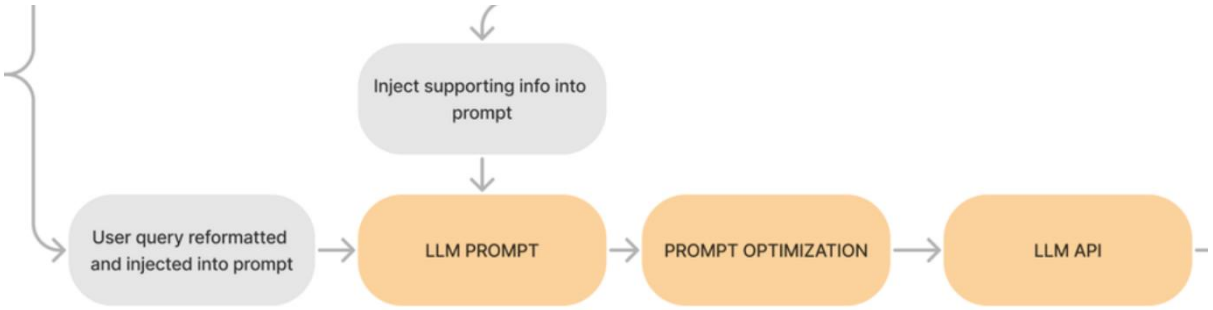
### 4. Knowledge Retrieval Layer (Retrieval-Augmented Generation)

For real-world applications, relying solely on the LLM's training data is insufficient. During my research, I found that most production-grade chatbots incorporate a knowledge retrieval layer to answer questions using external or domain-specific data such as documents, policies, or internal knowledge bases. This layer retrieves relevant information and supplies it to the LLM as contextual input. This overall approach is commonly referred to as Retrieval-Augmented Generation (RAG) and is critical for reducing hallucinations and improving factual accuracy.



### 5. Prompt Construction and Optimization

Once the user input and relevant contextual information are available, they are combined into a structured prompt. The prompt construction and optimization layer ensures that system instructions, retrieved knowledge, and the user query are organized in a clear and controlled manner. Effective prompt design helps guide the LLM's behavior, enforce constraints, maintain consistent tone, and improve the reliability of generated responses.

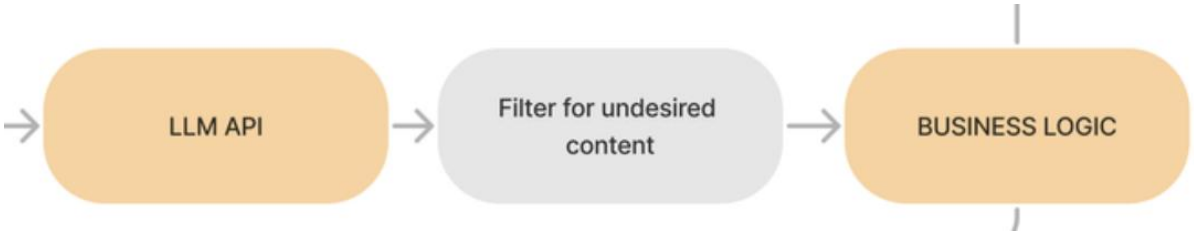



### 6. LLM Inference Engine

The structured prompt is then passed to the LLM inference engine, which generates the response. An important insight from my research is that the LLM does not independently search for information or verify facts. Instead, it relies entirely on the prompt and contextual data provided to it, and its role is limited to generating coherent and natural language output.

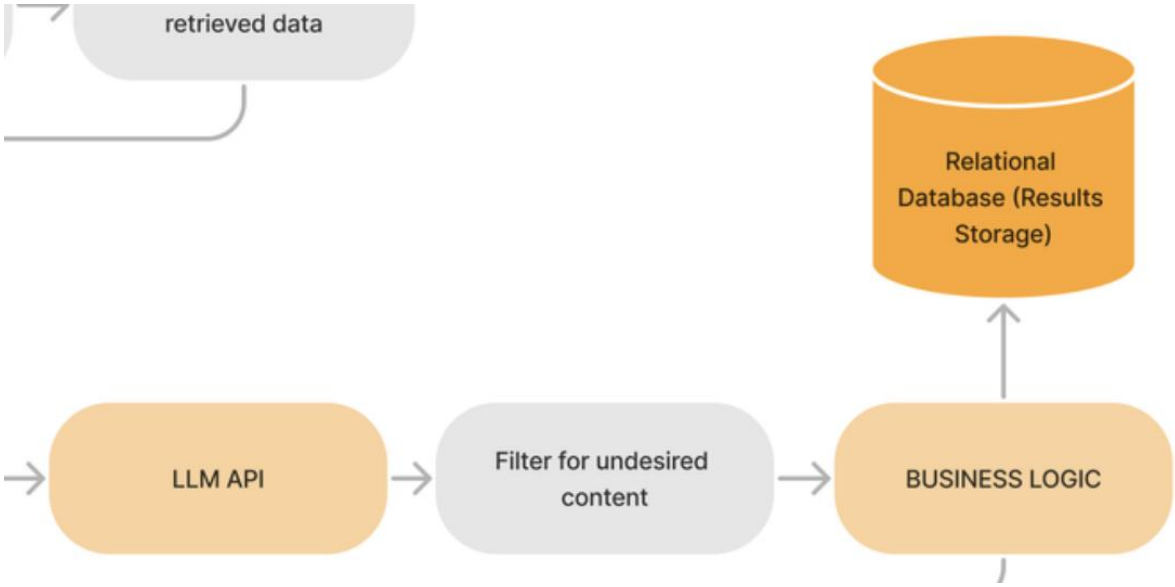
### 7. Post-processing and Content Moderation

The raw output generated by the LLM is not directly shown to the user. It first passes through a post-processing and content moderation layer, which filters sensitive or undesired content, enforces safety and compliance rules, and refines the response to match the desired tone or guidelines. This step is essential, especially for enterprise or public-facing applications.




### 8. Business Logic and Integration Layer

The business logic and integration layer applies domain-specific rules and connects the chatbot with external systems such as databases, APIs, or workflows. This layer enables the chatbot to perform actions, trigger workflows, store results, or integrate with existing systems based on the generated response.



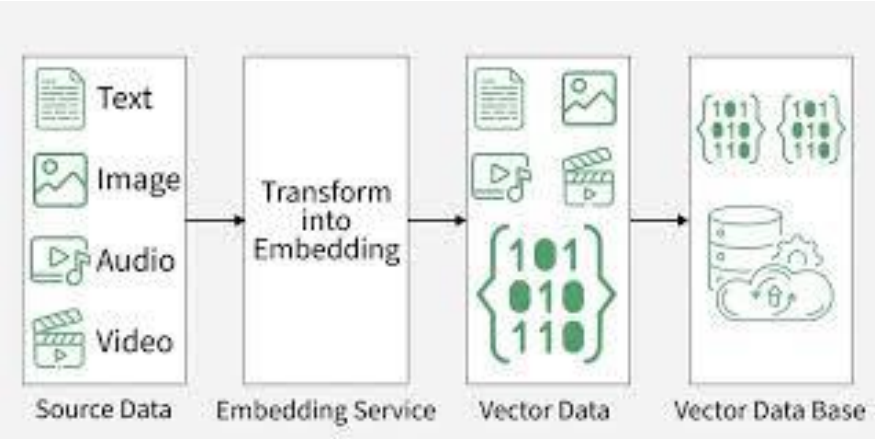

**9. Logging and Monitoring**

Finally, logging and monitoring are incorporated to track user interactions, system performance, and response quality. This data helps in debugging, evaluating system behaviour, and continuously improving the chatbot over time.

## 3)Understanding Vector Databases in LLM-Based Systems

While learning about LLM-based systems and Retrieval-Augmented Generation (RAG), I researched multiple resources across the internet to understand how vector databases work and why they are required. I explored different vector databases available today and tried to understand where each one fits based on scale, deployment, and use case.

From this research, I understood that vector databases are designed to store numerical representations (embeddings) of text data, which makes it possible to perform semantic search instead of simple keyword matching. This is especially important in LLM-based applications, where users may phrase the same intent in different ways.

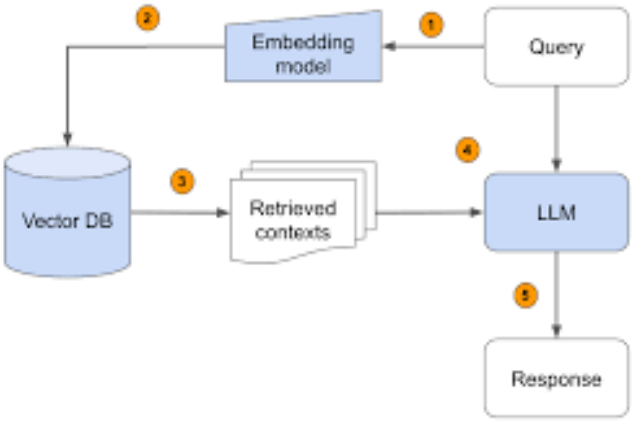



**Role of Vector Databases in RAG Systems**

In a RAG system, company documents are first broken into smaller chunks and then converted into embeddings using an embedding model. These embeddings capture the semantic meaning of the text and are stored in a vector database along with the original text.

When a user asks a question, the query is also converted into an embedding. The vector database compares this query embedding with the stored document embeddings and retrieves the most semantically relevant chunks. These retrieved chunks are then passed to the LLM as context, allowing it to generate accurate and grounded responses instead of relying only on its training data.

For example, queries like *"intern reimbursement policy"* and *"how much allowance do interns get"* may use different words but represent the same intent. Vector databases make it possible to handle such cases effectively.




**Research-Based Selection of a Vector Database**

After understanding the overall concept, I explored different vector databases such as FAISS, Pinecone, Weaviate, and Milvus. I compared them based on factors like deployment complexity, scalability, cloud dependency, and suitability for local or internal systems.

Based on this comparison, I found that **FAISS (Facebook AI Similarity Search)** is particularly useful for scenarios where fast semantic search is required and the system needs to run locally without relying on cloud-managed services.


**Hypothetical Problem Statement**

To apply this understanding practically, I considered a hypothetical scenario where I need to build an internal chatbot for a company. The chatbot's purpose is to answer employee questions using internal HR and policy documents such as PDFs and Word files. Since the data contains sensitive information, it must remain within the organization's infrastructure. The dataset size is moderate, and the primary requirement is fast and accurate semantic search.

**Why I Chose FAISS for This Problem?**

I chose FAISS for this hypothetical problem because:

- It is open-source and easy to integrate
- It supports very fast similarity search on high-dimensional vectors
- It works well for local and on-premise deployments
- It is widely used in research and real-world RAG implementations

In this setup, all document chunks would be converted into embeddings and indexed using FAISS. When a user submits a query, the query embedding would be compared against stored document embeddings to retrieve the most relevant content, which is then provided to the LLM as contextual input.

# Sujal Dhandre

9406017782 sujal.dhandre@gmail.com LinkedIn-Sujal Dhandre Github-Sdhandre Portfolio

## Education

**Medicaps University, Indore** **August 2022- July 2026**
*B.Tech in Computer Science & Engineering* *8/10 CGPA*

## Work Experience

**Feynn Labs** **June 2025 – August 2025**
*Data Science and Machine Learning Intern* *Remote*

- Documented and prototyped an **AI-based product idea**, defining 3+ core features and potential use cases.
- Performed market segmentation and exploratory data analysis (EDA) on 10,000+ **EV customer** records using Python, Pandas, and clustering techniques, generating 5 actionable customer personas to support AI product strategy.
- **Calculated a financial equation** for an AI product, projecting up to 20% revenue growth and implementing a working prototype.

## Projects

**AgriScan - Plant Leaf Disease Detection App** Source Code

- Designed and trained a **CNN-based image classification model** using TensorFlow and Keras on 15,000+ leaf images, applying data preprocessing and augmentation techniques.
- Conducted model evaluation and optimization, achieving 92% classification accuracy across **5+ plant disease** classes.
- Implemented an end-to-end inference pipeline for plant disease detection, enabling scalable and reliable prediction from unseen leaf images.

**CineMatch AI – Movie Recommendation Model** Source Code

- Built a content-based movie recommendation system using **TF-IDF vectorization and cosine similarity** on 20,000+ movies to generate personalized recommendations.
- Analyzed feature extraction and similarity scoring to rank relevant movies based on user-selected preferences.
- Developed an interactive **Streamlit application** with dynamic movie posters and trailers to enhance user engagement and usability.

**EduGauge – Real-Time Classroom Attention Monitoring System** Source Code

- Implemented a full-stack AI system using **FastAPI, MediaPipe Face Mesh, YOLOv8**, and a **RandomForest classifier** for real-time attention, drowsiness, and phone-usage detection.
- Executed a custom ML pipeline using 468 facial landmarks (1,404 features) and achieved **90% attention-state classification accuracy** on the test dataset.
- Enforced WebSocket-based low-latency streaming (<300ms round-trip) with real-time analytics visualization and temporal smoothing for robust multi-student tracking.
- Integrated **MongoDB** for session logging and **Google Gemini API** to generate automated educational insights and engagement summaries.

## Technical Skills

**Languages**: Python, C++, SQL, HTML/CSS
**Frameworks**: Flask, Bootstrap
**Developer Tools**: Git, GitHub
**Libraries**: Numpy, OpenCV, Pandas, Scikit-learn, TensorFlow, Keras, PyTorch
**Others**: Gen-AI, Data Science, Machine Learning, Cybersecurity, Video Editing, Branding & Marketing

## Research and Certifications

**Oracle Cloud Infrastructure – Certified Data Science and Generative AI Professional** **Verify Certificate**

- Cloud machine learning, model training, optimization, and deployment using OCI Data Science and OCI AI services.
- Experienced in training, optimizing, and deploying ML models with scalable cloud workflows.
- Certification valid through October 27, 2027.

**Data Science Bootcamp (Udemy)** **September 2024 – January 2025**

- Hands-on experience with data preprocessing, feature engineering, and ML model development.
- Built regression, classification, and clustering models with up to 80% improvement over baselines.

**Research Paper (IJEART): Integrating Blockchain in 5G Technologies** **February 2024 — Read Paper**

- Analyzed blockchain applications in 5G security, IoT, identity management, and network slicing
- Studied 5+ real-world use cases and security architectures.