Problem statement- assignment 1:

GitHub Link: https://github.com/Resh844/Reshma_Hegde_Assignment

1. API Data Retrieval and Storage:
2. Data Processing and Visualization:
3. CSV Data Import to a Database:
    - Github link for first three Tasks https://github.com/Resh844/Reshma_Hegde_Assignment

4. Most complex python code:
    - Link: https://github.com/Resh844/AccuKnox_Task_4-Most_Complex_python_code

5. Most complex database code :
    - Link: https://github.com/Resh844/AccuKnox_Task_5_most_complex_database_code

Problem statement- assignment 2:

1. Where would you rate yourself on (LLM, Deep Learning, Al, ML). A, B, C [A - can code independently; B can code under supervision; C- have little or no understanding]

1. **Large Language Model (LLM):** Choice B: I have built agents that helps me to find jobs and to tweak resume. I have practical understanding of

LLMs and their applications (chatbots, RAG). I am working on training and fine-tuning models from scratch.

2. **Deep Learning (DL):** Choice B: I have worked with deep learning models and I understand the fundamentals. I am exploring on understanding application architecture that uses deep learning.

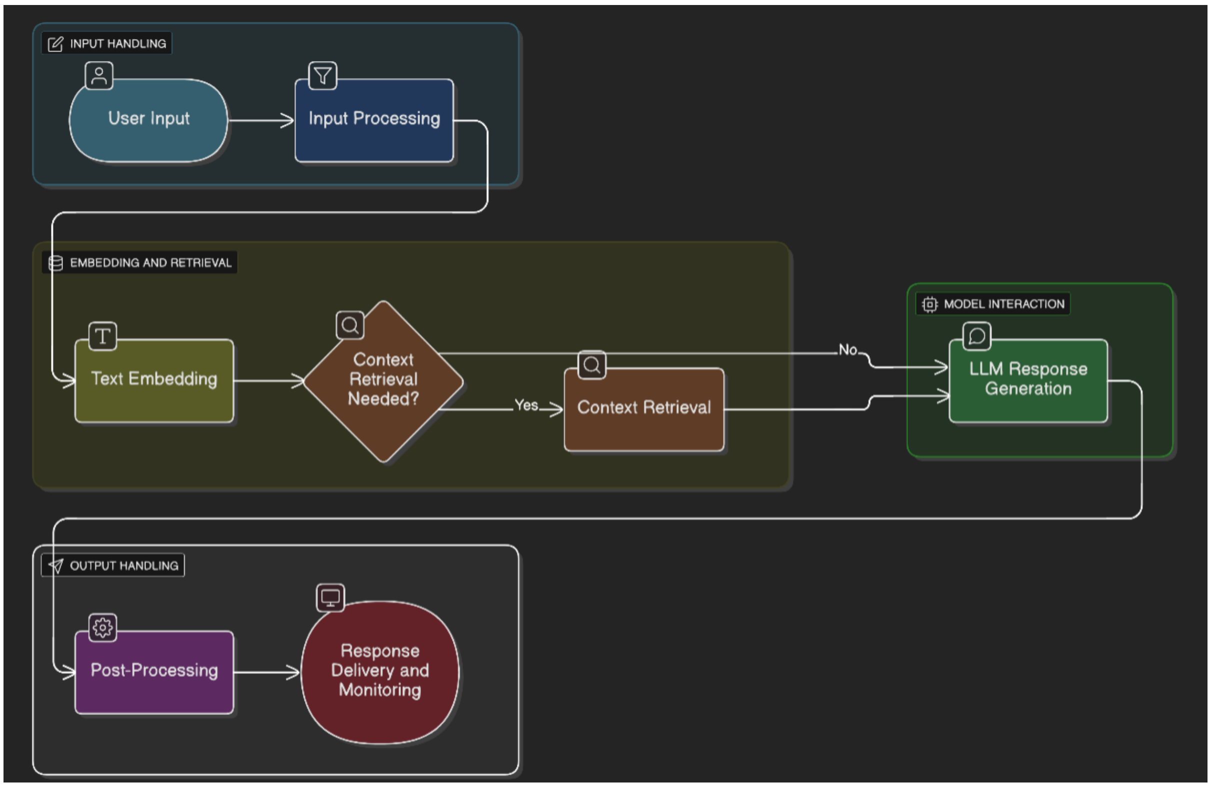
3. **Artificial Intelligence (AI):** Choice A: I have built agents and bots. I understand what AI can do and how it takes decisions. With the help of documentation, I can solve real world problems.

4. **Machine Learning (ML):** Choice A: I have practical experience with ML models like Logistic Regression and basic pipelines. I have also worked on ML projects. This includes preprocessing, training, and evaluation. I always like to have better accuracy, for which I refer to documents and training videos.

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

Answer:

Before implementing chatbot, it’s best to write down the goals of chatbot. Key things to note are, what are the use cases? how the prompt and response should look like? Are there any blockers? It is also important to write down metrics to measure the accuracy of the chatbot and how and when do you mark it as completed.



Step 1: User Input

The user enters a query through a chat interface such as a web or mobile application. The input can be in natural language and does not have to follow a fixed format. Chatbot can also have predefined queries based on the usecase.

Step 2: Input Processing

The system preprocesses the input by cleaning the text, handling minor spelling or formatting issues, and validating the request. This step ensures that the input is suitable for further processing and does not contain harmful or invalid content.

Step 3: Text Embedding

The processed input is converted into a numerical representation called an embedding. These embeddings capture the semantic meaning of the text, allowing the system to understand the intent of the query rather than just matching keywords.

Step 4: Context Retrieval

If the chatbot needs external knowledge, the embedding is used to search a vector database for relevant documents or past conversations. When the model gets more information, it becomes better at predicting the output, leading to better accuracy and better answers.

Step 5: LLM Response Generation

The user query and any retrieved context are passed to the Large Language Model. The LLM generates a response dynamically based on learned patterns from training data. The response is **not hard-coded** and can adapt to different types of questions. Even the prompt and their respective responses can be added but this makes our chatbot to answer to only limited questions.

Step 6: Post-Processing

The generated response is refined by formatting the text, removing unwanted content, and ensuring the answer is appropriate and safe for the user.

Step 7: Response Delivery and Monitoring

The final response is sent back to the user. Logs and feedback can be collected to monitor chatbot performance and improve future responses.

This step by step architecture allows chatbot to answer diverse user queries dynamically without relying on hardcoded user intent and responses while maintaining the accuracy and relevance. Overall, this architecture helps the chatbot to generate relevant responses to the user's intended questions.

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?

Answer:

A vector database is a specific type of storage for extensive searching and storing vector embeddings, which are digital decimals representing data like text. These vectors reflect the meaning of words and sentences semantically rather than just by giving the exact keywords.

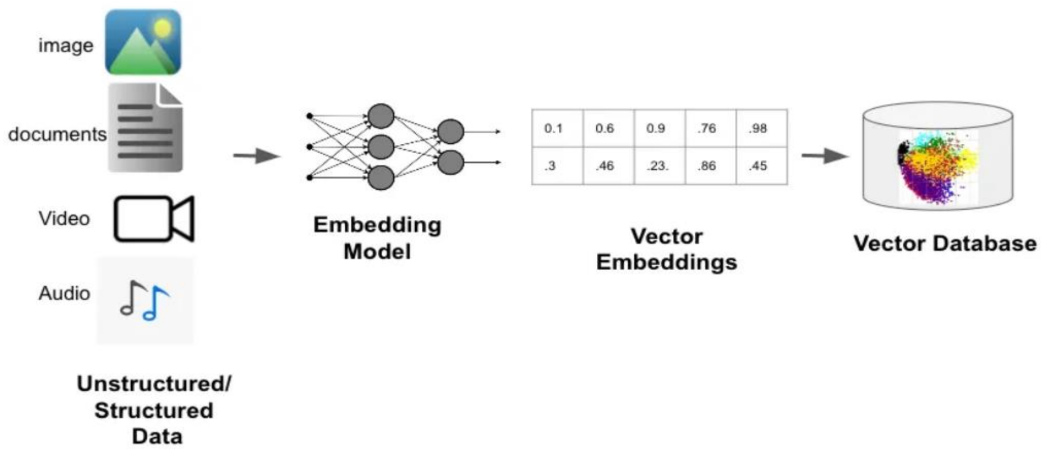
Reasons for the insufficiency of traditional databases:

In keyword-based traditional search systems, the term "Apple" would be considered a synonym. However, the word "Apple" in everyday language has different meanings based on the situation:

"An apple a day keeps the doctor away" === it is the fruit

"Apple brings out a new iPhone" === it is the company

A standard database will have a hard time because it uses exact matches.



The solution provided by vector databases:

The word depicting in a vector database each word is converted into embeddings that reflect the meaning.

Even though the word "Apple" stays the same, its embedding varies according to the context. This means the system retrieves results that are semantically coupled with the user's intent.

"This enables systems to interpret user intent even when the same word has multiple meanings."

To illustrate:

Nutrition-related queries will yield content pertaining to fruits

Smartphones related queries will get content related to the brand

Theoretical Use Case (Based on My Project):

Let’s consider my recent project an AI-driven Cold Email Generator which is able to assist students in making individualized cold emails and cover letters depending on their CVs and the job descriptions. In the case of this system, the users are free to express their skills, experiences, or target roles in any manner they like, but the system should still comprehend the basic purpose behind the request and produce corresponding outreach content accordingly.

The scenario would be quite hard for the traditional keyword-based matching approach since resumes and job descriptions are likely to use different vocabulary for the same roles and skills. For instance, a resume might refer to "software development" while a job description might use "application engineering." However, although the terms used differ considerably, the meanings are closely tied.

The system thus converts resumes and job descriptions into vector embeddings that contain their semantic meaning to tackle this issue. These embeddings are kept in a vector database. Upon the user uploading the resume and job description, the system gets the sections with the highest relevance not through exact keyword matches but via semantic similarity. This, in turn, allows the language model to generate more personalized and context-aware cold emails and cover letters.

Selection of a Vector Database (ChromaDB):

ChromaDB is the recommended vector database option for this project.

ChromaDB is a lightweight, open-source vector database that offers easy integration with Python-based frameworks such as LangChain. It is ideal for use cases such as document search and contextual augmentation in LLM systems. Given that the Cold Email Generator handles resumes and job descriptions of moderate size, its use of ChromaDB for similarity search is both efficient and straightforward, without adding unnecessary complexity.

In this setup:

- Resume and job description texts are converted into embeddings
- These embeddings are stored in ChromaDB.
- The user inputs are then compared to the stored embeddings through similarity search.
- The context that is retrieved is then sent to the LLM for the generation of personalized emails.

By employing a vector database in this way, the system is able to grasp the semantic connections between resumes and job descriptions, thus producing student-specific and more relevant outreach content.