MediMate

"हमसे पूछो, ***Google*** से नहीं"

ARYAMAN GUPTA, VAIBHAV SHARMA, KRISH SINGH

TABLES OF CONTENTS

TABLE OF CONTENTS

INDEX

(II)
ABSTRACT

The machine learning-powered Explainable AI Chatbot, seamlessly integrated within a Node-based web application, offers users personalized medical diagnoses and corresponding treatment plans based on their individual medical histories. Conversations with the chatbot are saved, enabling users to reference previous interactions.

This innovative system leverages the Explainable AI feature by sourcing information from diverse text resources to generate comprehensive responses. By considering prior conversations and user history, the Large Language Model (LLM) crafts accurate and tailored responses, utilizing the amalgamated knowledge from various texts.

The primary objective of this application is to deliver precise diagnoses and personalized treatment strategies aligned with each user's unique medical conditions and history.

(III)

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 General Introduction

In contemporary healthcare, numerous challenges persist, impeding individuals' access to timely and personalized medical assistance. Accessibility to immediate healthcare advice remains a substantial issue, often due to geographical constraints, overwhelming patient volumes in clinics or hospitals, and a shortage of healthcare professionals. This lack of immediate access can significantly impact individuals' health outcomes, especially in urgent situations or for those with limited access to healthcare facilities.

Moreover, the conventional healthcare system often struggles to deliver personalized care tailored to individual needs. Standard medical practices may not always account for the intricacies of each person's medical history, leading to generalized treatments that might not be optimal or effective for everyone.

In this context, an innovative application leveraging AI-driven technology emerges as a promising solution. Such an application, capable of analyzing individual medical records and considering past interactions, can provide tailored and accurate diagnoses and treatment plans. By bridging the gap in immediate access to medical guidance, it serves as a virtual responder, offering preliminary assessments crucial in urgent situations or when immediate medical attention isn't readily available. This addresses the critical need for timely healthcare advice.

Furthermore, the application's ability to deliver personalized medical assistance is transformative. By incorporating an individual's unique medical history, it ensures that recommendations are finely tuned to specific health conditions and histories. This level of personalization aligns with the evolving paradigm of patient-centered care, where tailored approaches lead to better health outcomes.

In essence, this innovative application represents a vital step forward in overcoming the challenges of accessibility and personalization in healthcare. By offering timely and personalized medical assistance, it stands as a beacon of hope in empowering individuals with reliable guidance and support for their health concerns, irrespective of geographical or resource constraints.

1.2 Problem Statement

The proposed application aims to address these pressing medical assistance issues by leveraging AI-driven technology to offer immediate and personalized medical assistance.Hence, the problem statement is to develop a machine learning-powered Explainable AI Chatbot, seamlessly integrated within a Node-based web application, offering users personalized medical diagnoses and corresponding treatment plans based on their individual medical histories.

1.3 Significance of the Problem

The challenges within the current healthcare landscape underscore a pressing need for solutions that transcend conventional limitations. Timely and personalized medical assistance stands as a linchpin in reshaping healthcare accessibility and outcomes. The significance of addressing these issues reverberates through numerous facets of healthcare delivery. Immediate access to accurate medical guidance is not just a matter of convenience; it's a critical factor in mitigating health crises, especially in remote areas or during urgent situations. Personalized care, tailored to individual medical histories

and needs, holds the potential to revolutionize treatment efficacy and patient outcomes. By embracing technology to bridge these gaps, healthcare systems can progress towards a more equitable and inclusive model, where geographical boundaries and resource limitations do not impede access to quality care. The ability to provide precise diagnoses and personalized treatment plans is not just about enhancing patient satisfaction; it's about fostering a healthcare environment that maximizes the potential for improved health outcomes, reducing healthcare disparities, and ensuring that every individual receives the care they deserve, when they need it most.

MediMate aims to provide a relevant diagnosis tailored to patients' medical history and provide a treatment plan as well. A basic diagnosis provided so easily would help to provide medical help to a larger number of people, free of cost.

The reference used by our Large Language Model comes from a medical book **Harison's Principles of Internal Medicine.** The entire book is broken into chunks and vectorized using some Vector Embeddings model. These vector embeddings are stored in a Vector Database, which help to provide an efficient searching mechanism for these vectors.

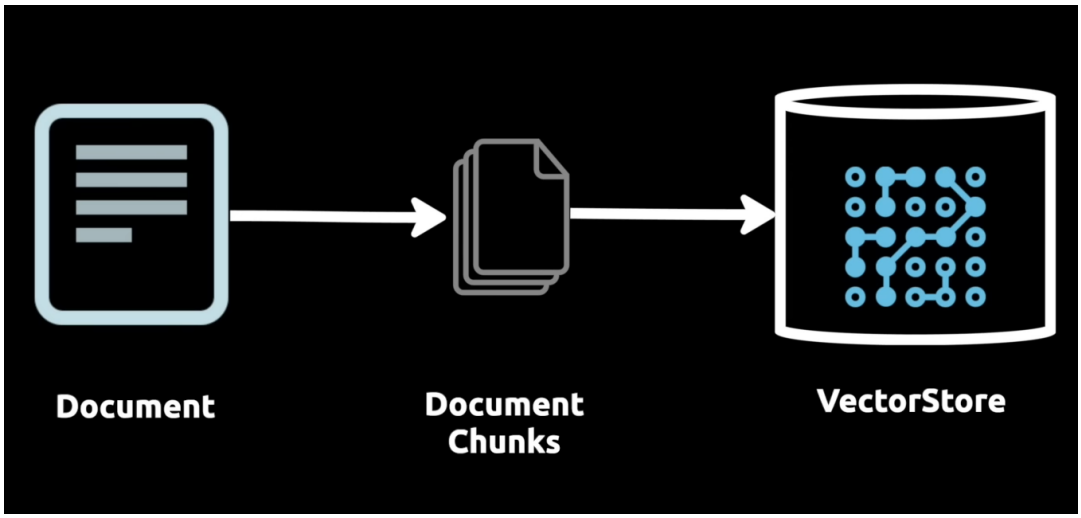

Figure 1.1 CREATION OF VECTOR DATABASE

Once a prompt is given by the user, it is converted into vectors and then the database is searched for most similar vectors using Similarity Search which uses cosine similarity rule.

The initial prompt and the 2 (can be varied) best results are sent to the Large Language Model as a system prompt. The LLM generates a response which is then shown to the user.

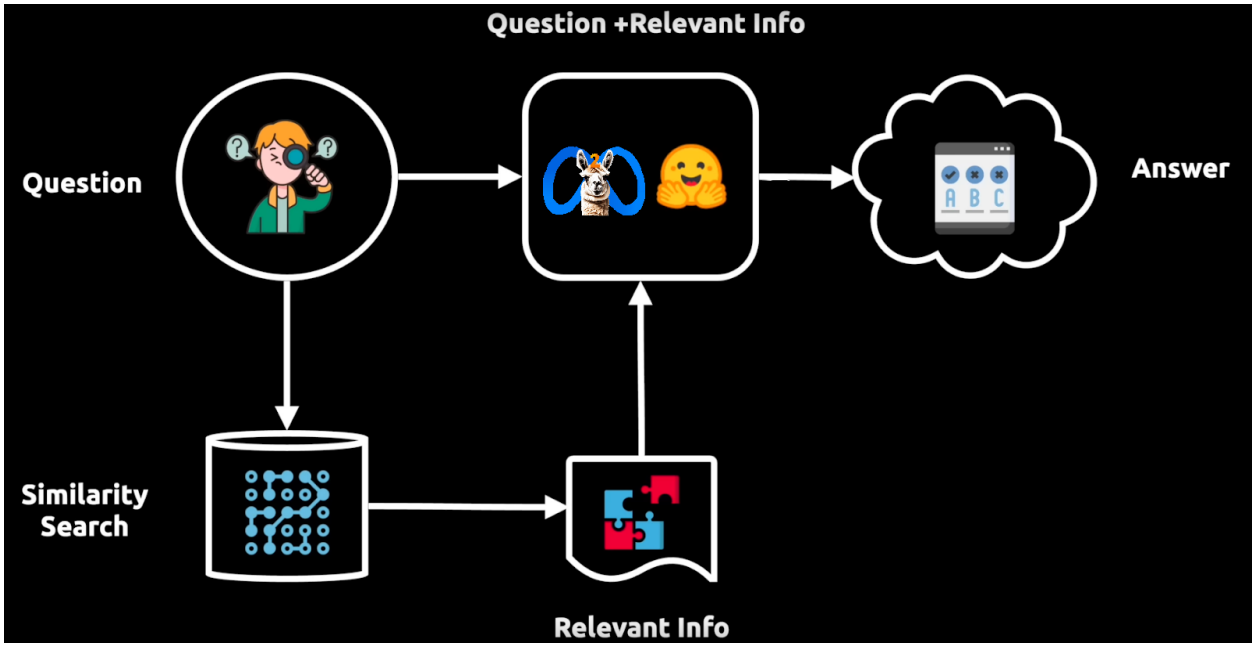

Figure 1.2 ARCHITECTURE OF THE LLM

1.4 COMPARISON OF EXISTING APPROACHES TO THE PROPOSED SYSTEM

1 . **Mayo Clinic Symptom Checker** - A symptom checker with a wonderful database, Mayo Clinic has made a selection system where users can select symptoms from a plethora of given options. They can select related factors to it as well, and as a result it provides multiple causes of those symptoms.

The issue is that there is no consideration of users medical history or that of any tests results that user may have, that could give a more appropriate diagnosis. Moreover there is no option of showing a treatment plan.

2. **Zini AI** - Zini is a AI driven virtual physician She can discuss the whole scenario in detail, figure out what's wrong and guide you in a timely manner. A full fledged company, Zini has gotten recognition from various renowned doctors and institutes including AIIMS.
The issue is that there is no explainability of the answer they provide you. The results are as such, and it is up to the user to trust it or not.

Our app surpasses existing solutions by leveraging individual medical histories for precise diagnoses, unlike Mayo Clinic's Symptom Checker. It excels over Zini AI by offering transparent insights through Explainable AI, building user trust. Furthermore, your app doesn't stop at diagnosis; it provides personalized treatment plans, a unique feature absent in both existing solutions. This comprehensive approach ensures accuracy, transparency, and actionable guidance, setting your app apart as the superior choice for personalized medical assistance.

CHAPTER 2
LITERATURE SURVEY

2.1 SUMMARY OF RESEARCH PAPER STUDIED

- The paper - **“Artificial Intelligence to Improve Antibiotic Prescribing: A Systematic Review” -** A review paper which looked into various Artificial Intelligence techniques used in Antibiotic prescribing. Studies reviewed in this paper used supervised machine learning (ML) models as a subfield of AI, such as logistic regression, random forest, gradient boosting decision trees, support vector machines and K-nearest neighbors. Each study showed a positive contribution of ML in improving antibiotic prescribing, either by reducing antibiotic prescriptions or predicting inappropriate prescriptions.
- The paper - **“AI Doctor: An Intelligent Approach for Medical Diagnosis”** This paper touches on the difficulty in the diagnosis of a particular medical problem, which in turn will lead to incorrect medication. As the relationship between symptoms and diseases are not always simple, there are many cases in which the symptoms and diseases overlap, in such situations diagnosis of a particular disease becomes difficult. Hence, the solution they proposed was to use a External Knowledge Base to extract results instead training AI model directly with that data.

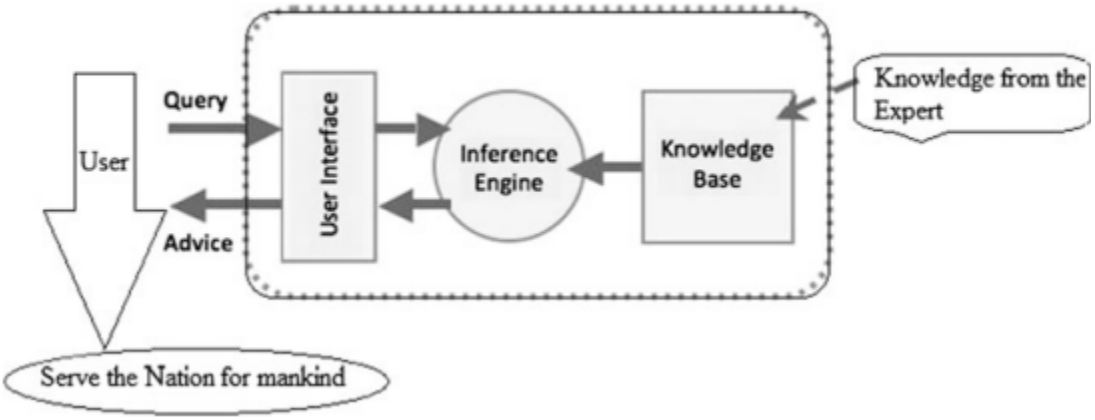

Figure 2.1 External Knowledge Base Approach

- The paper - **"DDXPlus: A New Dataset For Automatic Medical Diagnosis" by Arsène Fansi Tchango 2205.09148.pdf (arxiv.org)** The Automatic Symptom Detection are designed to interact with patients, collect evidence about their symptoms and relevant antecedents, and possibly make predictions about the underlying diseases. Despite recent progress in this area, an important piece of doctors' interactions with patients is missing in the design of these systems, namely the differential diagnosis. Hence, this paper provides a differential diagnosis dataset along with it's analysis.

  We tried to use the dataset, but the best possible thing we could come up with was a Yes/No question answering system which would give you a diagnosis at the end. This system felt very unattractive and inaccurate, hence was dropped later.

2.2 INTEGRATED SUMMARY OF LITERATURE STUDIED

These papers collectively advocate for AI's role in medical diagnosis and antibiotic prescribing. They highlight the challenges of accurate diagnosis due to overlapping symptoms and the necessity for more sophisticated AI approaches beyond simple models.

While exploring various AI techniques, the consensus leans towards incorporating External Knowledge Bases for richer data insights and enabling to expand dataset, if needed, in the future, enabling more accurate predictions. However, the struggle to harness complex medical datasets indicates the need for improved AI strategies, emphasizing Explainable AI—systems that offer transparent, understandable results—to ensure trust and reliability in medical decision-making.

CHAPTER 3
METHODOLOGY AND SOLUTIONS APPROACH

3.1 REQUIREMENT ANALYSIS

Our requirement for the implementation and execution of the project involved using the following -

1. TheBloke/Llama-2-70B-Chat-GGML Large Language Model
2. hkunlp/instructor-xl instruction-finetuned text embedding model
3. A powerful CPU based cloud machine to run the model (running on AWS)
4. An API Gateway, to route API to local machine
5. NextJs library to create frontend
6. ClerkAPI to authenticate a user
7. Local machine to run frontend and call API

Functional Requirements -

1. Authenticate a user whenever he/she logs in
2. Enable users to see their previous chats
3. Provide a valid diagnosis, based on the symptoms provided
4. Provide a treatment plan based on the diagnosis

5. Create a follow up response keeping Conversational History in buffer

3.2 PROPOSED SYSTEM

The machine learning explainable AI chatbot incorporated within a Node based web app proposed is designed to provide users with diagnosis tailored to their medical history and then give a corresponding treatment plan as well. Your chats are saved and can be referred to later as well.

Additionally, the chatbot is compatible with chat history so that it takes your previous chats/prompts into consideration while generating a new response. The Large Language Models explainability comes from the resources we give it to generate the answer, meaning that the resources are searched for from various texts and then given as answers to the LLM. The LLM then uses answers to formulate a perfect english paragraph keeping your past chats into consideration.

The app's main goal is to provide a valid diagnosis with a personalized treatment plan tailored to each user's medical conditions/history.

CHAPTER 4
MODELING DETAILS

4.1 EXPERIMENTAL SETUP

Our Machine Learning model is mainly divided into 2 subparts -

1. Creating a Vector Embeddings and storing it in a Vector Database
2. Generate responses using Large Language Model using initial prompt and the Vector Database

The dataset used by our Large Language Model comes from a medical book "**Harison's Principles of Internal Medicine".** The entire book is broken into chunks and vectorized using some Vector Embeddings model. These vector embeddings are stored in a Vector Database, which help to provide an efficient searching mechanism for these vectors.

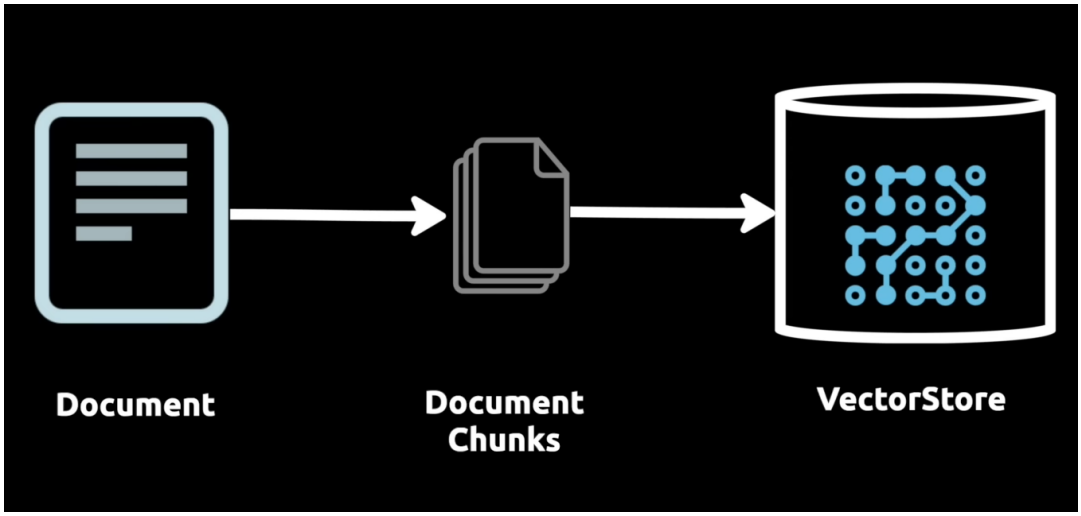

Figure 1.1 CREATION OF VECTOR DATABASE

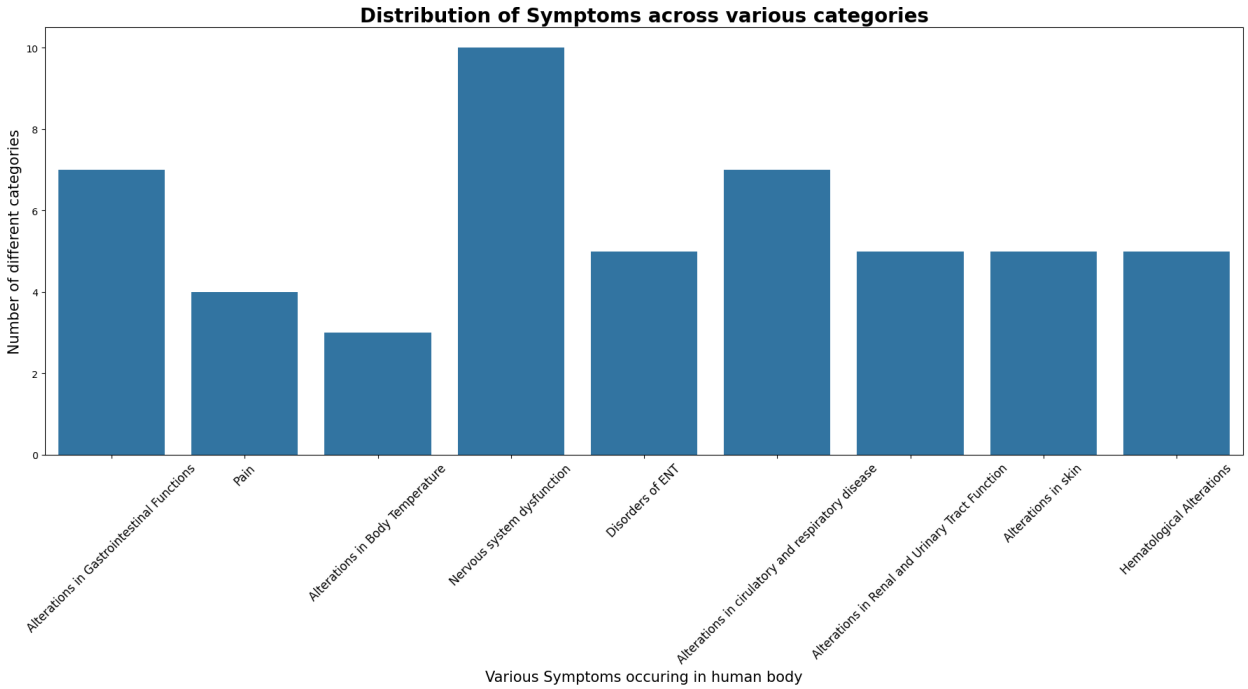

Figure 4.1 DATASET ANALYSIS - 1

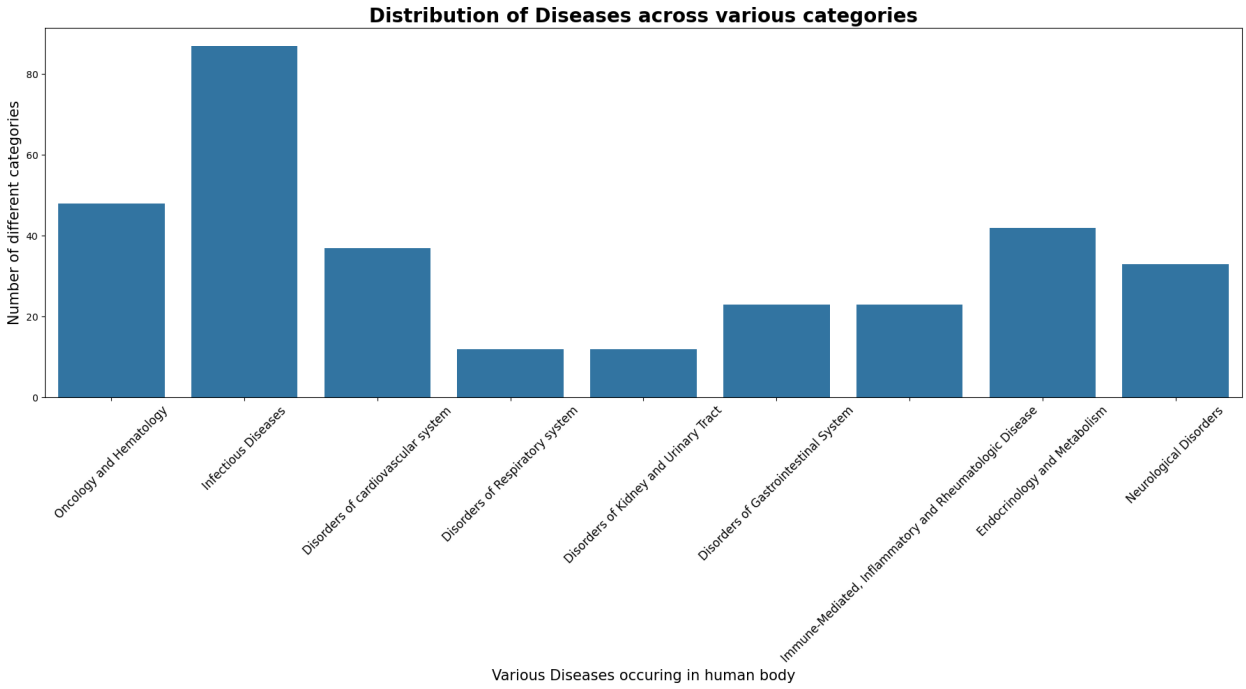

Figure 4.2 DATASET ANALYSIS - 2

Once a prompt is given by the user, it is converted into vectors and then the database is searched for most similar vectors using Similarity Search which uses cosine similarity rule.

The initial prompt and the 2 (can be varied) best results are sent to the Large Language Model as a system prompt. The LLM generates a response which is then shown to the user.

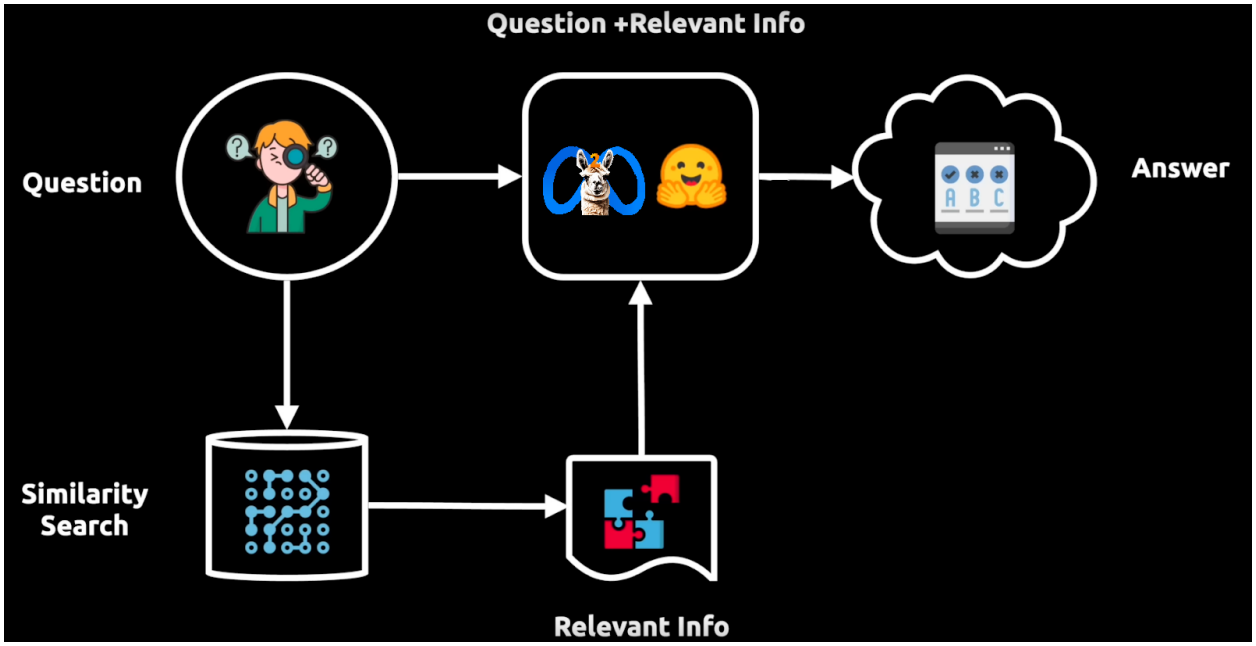

Figure 1.2 ARCHITECTURE OF THE LLM

4.1.1 VECTOR EMBEDDINGS

Each sentence in natural language such as English has a different meaning based on the context it is being used. Hence, the semantic meaning of each sentence is different. This difference in semantic meaning of different sentences cannot be understood directly by machine, hence there is a need for Language Representation.

Language representations are used in natural language processing (NLP) to convert words or tokens into numerical vectors that can be processed by machine learning models. These representations help capture the meaning and context of words in a way that computers can understand. There are two main types of language representations: context-free and contextual.

1. Context-Free Language Representation:
    - In context-free language representations, such as GloVe (Global Vectors for Word Representation) and Word2Vec, each token (word or word piece) in the vocabulary is assigned a fixed, constant vector representation.
    - These representations are pre-trained on large text corpora and do not depend on the context in which the word appears in a specific sentence or

document. In other words, the same word will have the same vector representation regardless of the sentence it's used in.
    - This can be thought of as a "one-size-fits-all" approach, where a word's meaning and usage are represented the same way in all contexts.
2. Contextual Language Representation:
    - In contextual language representations, such as ELMo (Embeddings from Language Models) and OpenAI GPT (Generative Pre-trained Transformer), the vector representation of a token depends on the context in which it is used within a sentence or document.
    - These representations are generated using deep learning models, like transformers, which take into account the surrounding words and their relationships to determine the token's vector representation.
    - This allows contextual language representations to capture nuances and variations in the meaning of words based on their context. For example, the word "bank" could have different meanings in the context of "river bank" and "financial institution."

Clearly, in our use case using Contextual Language Representation is a must. For example,

"The patient's discharge improved after the treatment." (Here, "discharge" refers to the patient leaving the hospital after recovery.)

"The patient experienced a foul-smelling discharge." (In this context, "discharge" refers to an abnormal bodily secretion, often indicating an infection or medical issue.)

To create vector embeddings there are multiple open source text embedding models present on HuggingFace and other websites.

The examples are given below -

1. **Instructor-Large embeddings**
   Ranking on MTEB Leaderboard - 15

Model Size - 1.34GB

An instruction-finetuned text embedding model that can generate text embeddings tailored to any task (e.g., classification, retrieval, clustering, text evaluation, etc.) and domains (e.g., science, finance, etc.) ***by simply providing the task instruction, without any finetuning.***

```
from InstructorEmbedding import INSTRUCTOR

model = INSTRUCTOR('hkunlp/instructor-large')
sentence = "I have fever"
instruction = "Represent the Medical title:"
embeddings = model.encode([[instruction,sentence]])
print(embeddings)
```

Output -

```
load INSTRUCTOR_Transformer
max_seq_length  512
[[-2.57632770e-02 -2.26812977e-02 -9.65026207e-03  3.62211913e-02
   1.89272147e-02  4.17050980e-02 -1.15692550e-02 -5.63997589e-03
  -4.33029681e-02  4.56826435e-03  4.01602760e-02  2.16395338e-03
   5.60584590e-02  5.93853258e-02 -5.33237420e-02  1.95816555e-03
  -3.80801298e-02  1.28692286e-02 -4.70876023e-02  3.38280238e-02
   7.04244003e-02  1.74206942e-02  2.39804573e-02  6.82936981e-02
   3.12318206e-02  3.37150507e-02 -1.68771408e-02  1.82671435e-02
   4.52423282e-02 -1.91237740e-02 -9.23950784e-03 -2.12482605e-02
  -4.59990799e-02 -6.06917515e-02 -1.67212281e-02  4.96358611e-02
...               ...         ...         ...
...               ...         ...         ...
...               ...         ...         ...
     5.10672666e-02 -3.62283997e-02  4.24865186e-02  4.25614864e-02
   1.94066912e-02  5.58609068e-02 -3.04381037e-03 -4.45343405e-02
   3.22658429e-03 -3.86688299e-02  3.70877683e-02 -4.21995148e-02
  -1.73537079e-02 -4.85996865e-02 -2.29737163e-02 -1.48055525e-02
   1.77637208e-02 -2.18810025e-03  1.45752290e-02 -6.13045925e-03
   4.28388678e-02 -5.15000299e-02 -8.76099989e-02 -5.09872325e-02
  -1.74751636e-02  3.38365212e-02  3.66140865e-02 -1.78974066e-02
```

```
-1.86085317e-03 -3.27622220e-02 -2.69401278e-02  3.91918756e-02
-4.54978943e-02 -3.46259363e-02  3.34719196e-02  4.22909856e-02
-6.90983087e-02  3.55564468e-02  1.02446359e-02 -4.64519039e-02
 4.82780337e-02  5.11324685e-03  3.59296128e-02  3.78740951e-02
 1.62743833e-02 -8.52099434e-03  6.13585673e-03 -3.54682617e-02
-5.43214940e-02 -2.52745934e-02  1.14600547e-02  5.72433732e-02]]
```

```
from InstructorEmbedding import INSTRUCTOR
from sklearn.metrics.pairwise import cosine_similarity

model = INSTRUCTOR('hkunlp/instructor-large')

sentence_a = "I have malaria. Because of which I am having headache, vomitting,
high temperature, fever and shivering."
sentence_b = "Crocin should be taken in cases of high fever"
instruction = "Represent the Medical title:"

embeddings_a = model.encode([[instruction,sentence_a]])
embeddings_b = model.encode([[instruction,sentence_b]])
similarities = cosine_similarity(embeddings_a,embeddings_b)

print(similarities)
```

Output -

```
load INSTRUCTOR_Transformer
max_seq_length  512
[[0.84754217]]
```

2. **Instructor-XL embeddings**

   Ranking on MTEB Leaderboard - 14

   Model Size - 4.96GB

   An upgraded version of Instructor-large embeddings, Instructor-XL has a more appropriate context recognition from its embeddings.

3. **Cohere embeddings**

   Ranking on MTEB Leaderboard - 13

   Model Size - None (because it is an API)

   Cohere API, even though paid, is a wonderful and a fast Semantic Searching vector embedding model, which is widely used in the industry.

```
import cohere
import numpy as np

cohere_key = key
co = cohere.Client(cohere_key)

#Encode your query with input type 'search_query'
query = "What is Pytorch"
query_emb = co.embed([query], input_type="search_query",
model="embed-english-v3.0").embeddings
query_emb = np.asarray(query_emb)

print(query_emb)
```

```
Output-
[[-0.00658035 -0.00161648 -0.0014534  ... -0.0508728  -0.0489502
  -0.01771545]]
```

```
import cohere
import numpy as np

cohere_key = key
co = cohere.Client(cohere_key)

docs = ["The capital of France is Paris",
       "PyTorch is a machine learning framework based on the Torch
library.",
       "The average cat lifespan is between 13-17 years"]


#Encode your documents with input type 'search_document'
```

```
doc_emb = co.embed(docs, input_type="search_document",
model="embed-english-v3.0").embeddings
doc_emb = np.asarray(doc_emb)


#Encode your query with input type 'search_query'
query = "What is Pytorch"
query_emb = co.embed([query], input_type="search_query",
model="embed-english-v3.0").embeddings
query_emb = np.asarray(query_emb)
query_emb.shape

#Compute the dot product between query embedding and document embedding
scores = np.dot(query_emb, doc_emb.T)[0]

#Find the highest scores
max_idx = np.argsort(-scores)

print(f"Query: {query}")
for idx in max_idx:
 print(f"Score: {scores[idx]:.2f}")
 print(docs[idx])
 print("--------")
```

Output -

```
Query: What is Pytorch
Score: 0.73
PyTorch is a machine learning framework based on the Torch library.
--------
Score: 0.12
The capital of France is Paris
--------
Score: 0.06
The average cat lifespan is between 13-17 years
--------
```

The time taken by the instructor XL embeddings to create a vector database of my 15000 page pdf was roughly 1 hr 24 mins where as by Cohere it was 1 hr 59 Mins.
Clearly Instructor XL is faster, and the accuracy tradeoff is not that much, hence we would be using Instructor-XL in our case.

4.1.2 VECTOR DATABASE

A **traditional database**, or more formally, a relational database, organizes data into tables, rows, and columns, and uses Structured Query Language (SQL) for managing and manipulating the stored data. They are the backbone of many enterprise systems, with popular examples including MySQL, Oracle, and PostgreSQL.

A **vector database**, on the other hand, stores data in a mathematical construct known as a vector space. Each data point is represented as a vector, and vector databases can perform high-speed computations involving these vectors, making them well-suited for tasks involving similarity searches and machine learning. Faiss and Annoy are examples of libraries used to build vector databases.

	TRADITIONAL DB	**VECTOR DB**
STORAGE COST	Increases linearly	Depends on vector dimensions
QUERY PERFORMANCE	Can degrade significantly	Designed to handle large-scale data
MAINTENANCE OVERHEAD	Can increase significantly	Distributed nature can mitigate some overhead

Table 4.1 Traditional DB vs Vector DB

For our use case, **Facebook AI Similarity Search (FAISS)** database served the best purpose. Facebook have built nearest-neighbor search implementations for billion-scale data sets that are some 8.5x faster than the previously reported state-of-the-art, along with the fastest k-selection algorithm on the GPU known in the literature.

FAISS does searching of vectors using this formula -

$$j = argmin_i \|x - x_i\|$$

where $\|\cdot\|$ is the Euclidean distance (L^2).

It can return not just the nearest neighbor, but also the 2nd nearest, 3rd, …, k-th nearest neighbor. Hence, in our use case we get 1st and 2nd nearest neighbors as answers.

4.1.3 LARGE LANGUAGE MODEL

Finally the 3rd component of our model comes in the form of a Large Language Model. This model is

1. the one the users would be conversing with,
2. the answers extracted from the database would be fed
3. maintenance of conversational history would also be done by it

We decided to go with Llama-2 Large Language Model, again state-of-the-art conversational LLM, which is being used in the industry for development purposes. As the original Llama-2 variants are very heavy, and not meant to run on simple machines we are using quantized Llama-2 models available on HuggingFace.

There are 2 types of quantization (LLM compression) algorithms -

1. GPTQ - Compression algorithm mainly focused to run on GPU
2. GGUF (previously known as GGML) - Compression algorithm mainly focused to run on CPU

Hence, due to unavailability of good GPU machines, we decided to go with GGUF compressed models.

Again we tried 3 models -

1. Llama-2-7B-chat-GPTQ : As already stated earlier, due to the limitations in GPU capacity on local machine and no free GPU cloud machines available online, GPTQ models had to be dropped
2. Llama-2-13B-chat-GGML: A great model, approximately 8GB model which requires 10-12GB of max RAM usage. The problem we faced when using this model was that it often hallucinated. *LLM hallucinations are the events in which ML models, particularly large language models (LLMs), produce outputs that are coherent and grammatically correct but factually incorrect or nonsensical.*
3. Llama-2-70B-chat-GGML: Best in class model, approximately 41GB model requiring close to 43GB of RAM. The main advantage is that the quality loss from the original model is very low and this model is great at maintaining Conversation History in a local buffer. Moreover, it has a 4096 max token limit and hence can be fine tuned easily according to our needs.

One question which comes to mind is that what does 7B, 13B, 70B mean? Why the quality is increasing as this number increases?

70B or "70 billion parameters" refers to the network's neural connections that are adjustable during the training phase. Parameters in a neural network are the variables that the model tunes during training to learn patterns from data. These parameters include weights and biases across the numerous layers of the network.

Having 70 billion parameters indicates an extensive and complex network architecture capable of capturing and understanding intricate patterns within vast datasets, making it highly capable in processing and generating accurate language-based responses. The sheer number of parameters allows the model to learn and adapt to a wide range of linguistic nuances and context, enhancing its performance in generating coherent and contextually relevant text.

Hence for our use case, the 70B Llama2 model gave the best performance.

4.1.4 STITCHING EVERYTHING TOGETHER

Connecting a Vector Database with the Large Language Model along with maintaining a conversational buffer is not that easy. This is where Langchain comes in.

What is Langchain?

An open source software that allows AI developers to combine LLM models like GPT-4, with external sources of computation and data like excels, pdfs etc.

4.2 DESIGN DIAGRAM

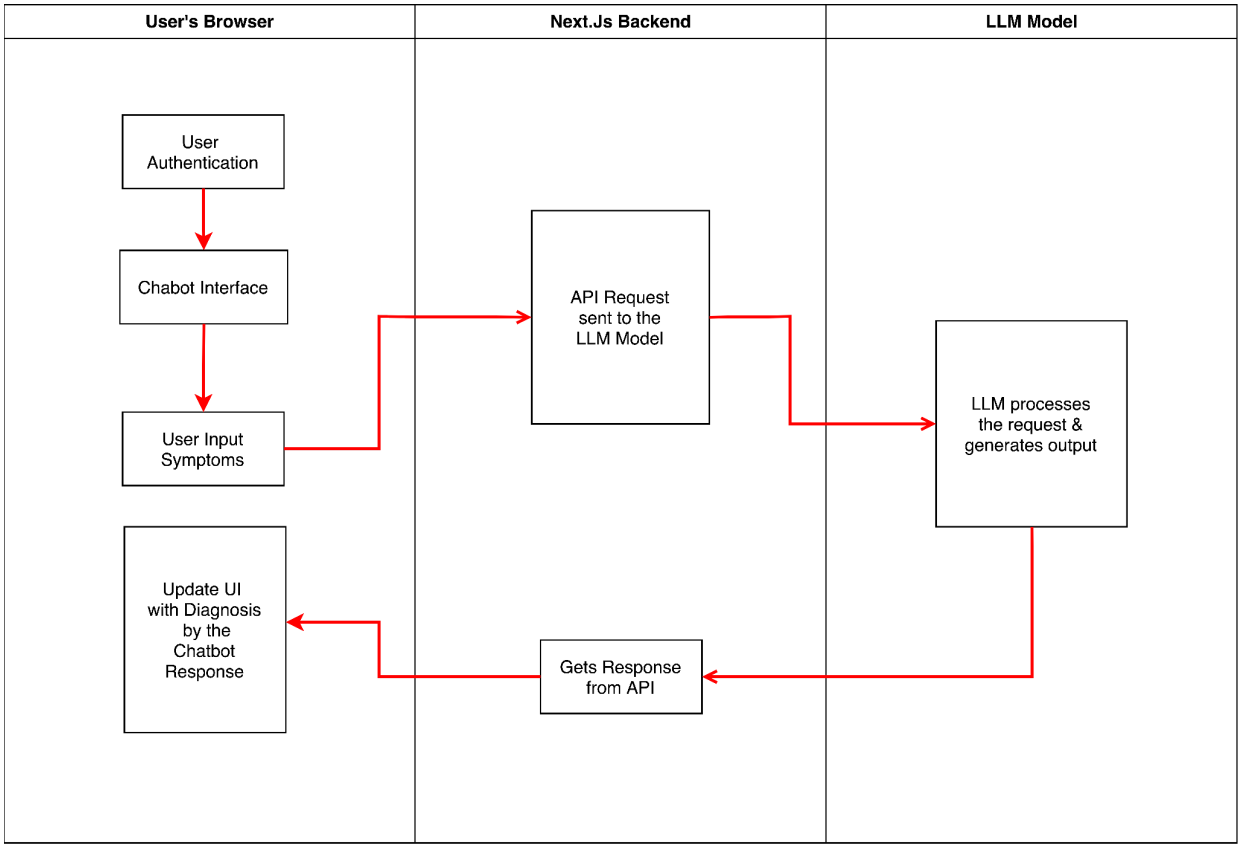

Figure 4.3 WORKFLOW DIAGRAM

Our entire application can be divided into 3 faces, namely-

1. User's Browser or Frontend
    a. The user will login in using his credentials on our website
    b. There he will be able to chat using our chatting interface
    c. The query entered by the user will be sent to the backend
    d. The response received from the LLM will be shown on the frontend

2. Backend (handling the frontend)
    a. Receives the query prompt from the frontend
    b. Send the query prompt to LLM Model (API)
    c. Receives the response from the LLM Model (API)
    d. Sends back to the User's Browser
3. LLM Model (API)
    a. LLM Model is deployed on AWS as an API
    b. Gets the user prompt to generate response
    c. Sends the generated response back as a Response to the API request

CHAPTER 5

TESTING

5.1 TESTING PLAN AND RESULTS

At the end, 2 machines were created and the results were compared for a set of questions (19)-

BLUE - (Ref- https://huggingface.co/hkunlp/instructor-large)

GPU Memory - 16gb
GPU 0- Tesla T4
23 Prompts
Embeddings used - **`hkunlp/instructor-large (variating here)`**
Model used - **`Llama 2 13B, 128 group size model - GPTQ`**
Database used - **Faiss (constant)**

ORANGE- (Ref- https://huggingface.co/hkunlp/instructor-xl)

CPU Memory - 192gb
CPU- Intel Xeon Platinum 8275CL (Cascade Lake)
23 Prompts
Embeddings used - **`hkunlp/instructor-xl (variating here)`**
Model used - **`Llama 2 13B, 128 group size model-GGUF`**
Database used - **Faiss (constant)**

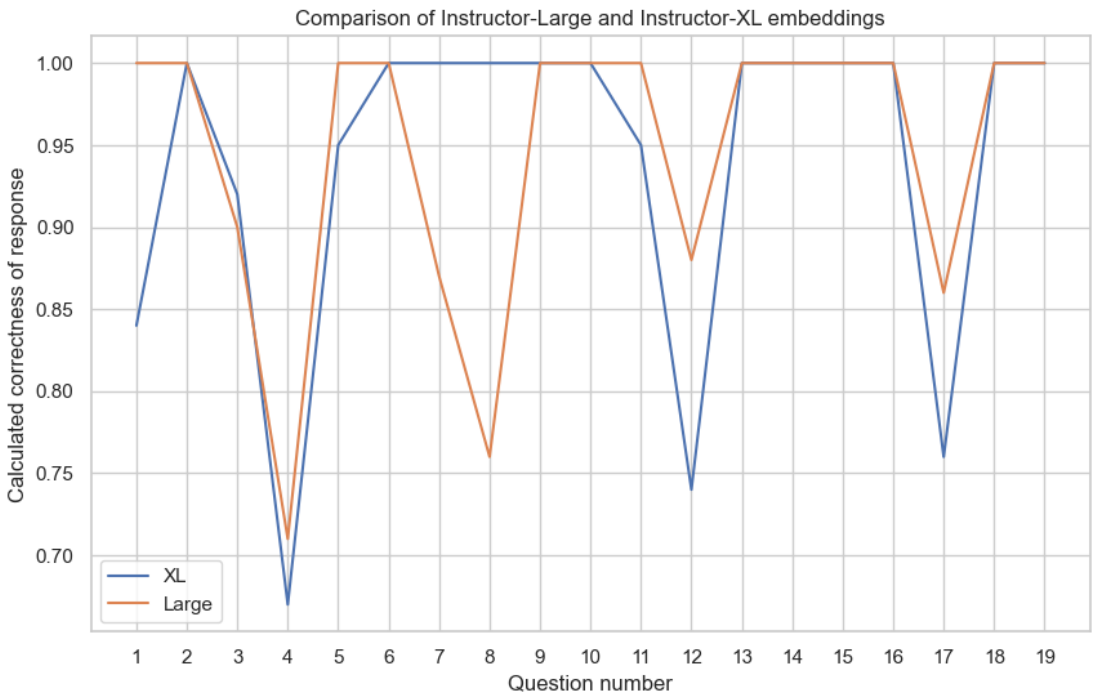

Figure 5.1 Comparison of Setups

Even though there is no clear winner, combining a more refined model (Instructor-XL) with a better Large Language Model (we are using Llama 2-70B-chat-GGML model in practice) we can achieve higher consistency and accuracy of results than we see in this graph.

5.2 WORKING SCREENS

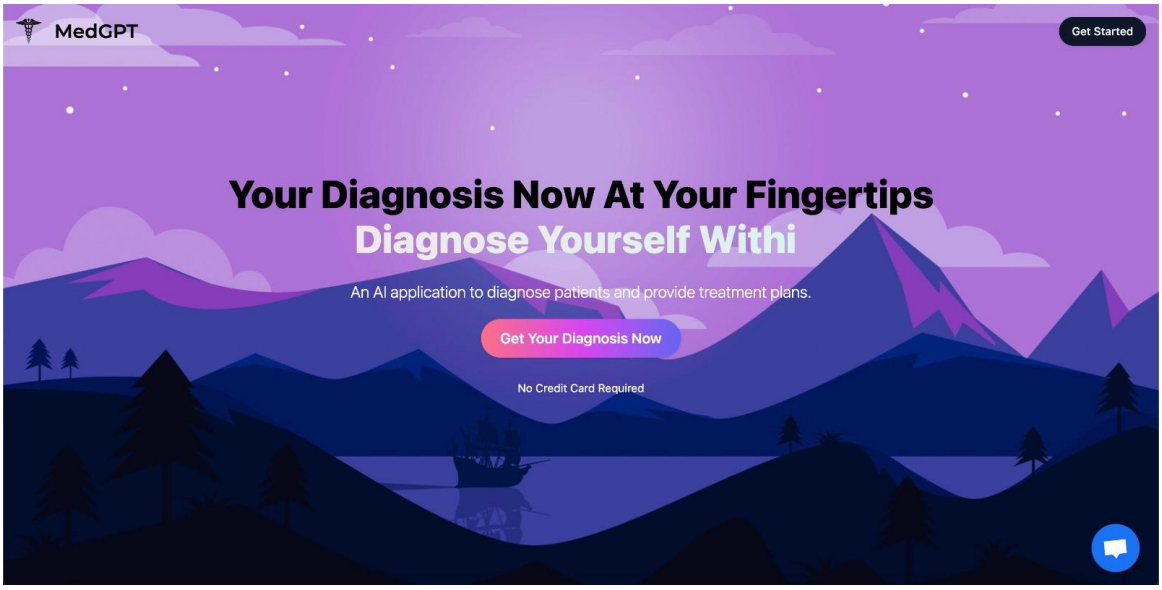

Figure 5.2 Home Page

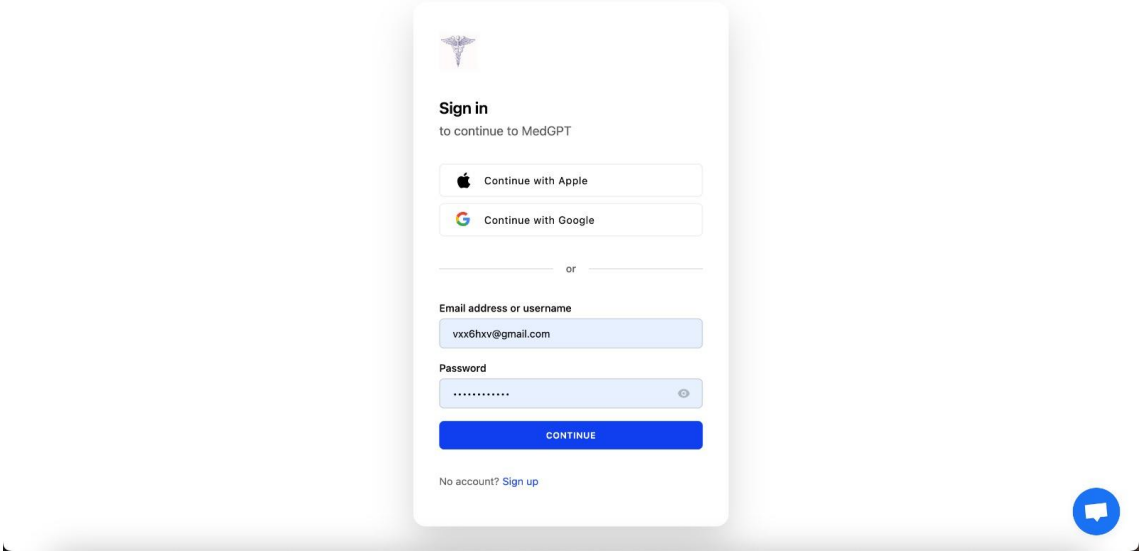

Figure 5.3 Login/SingUp Screen

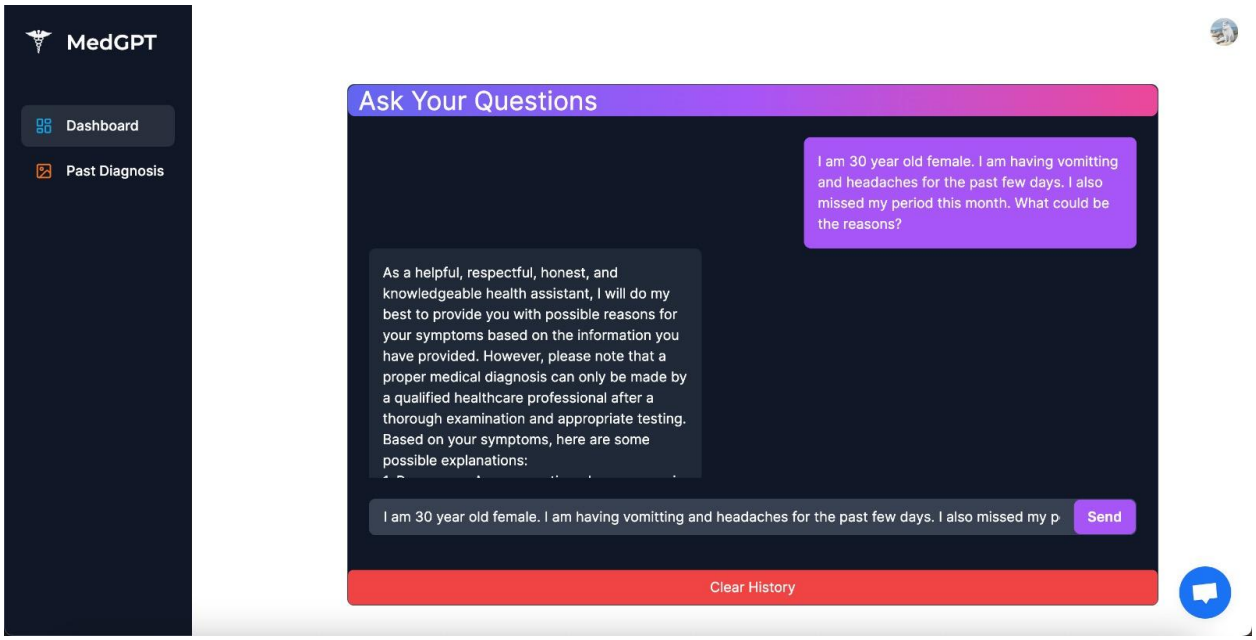

Figure 5.4 Chatbot interface

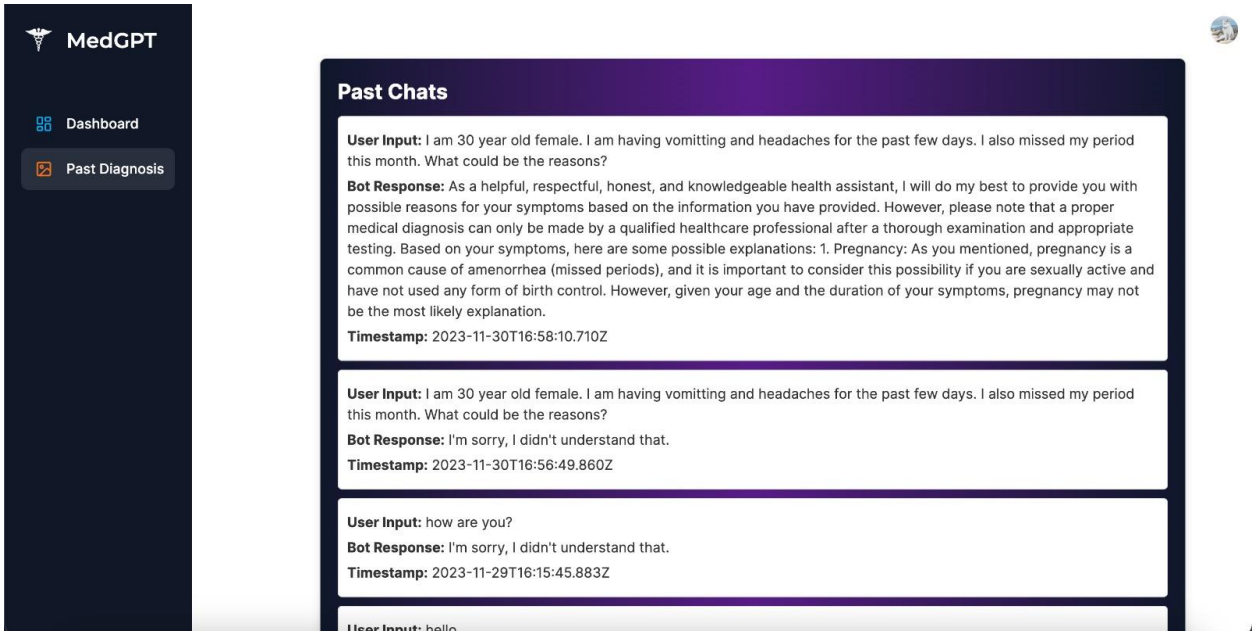

Figure 5.5 User's Past Chats

5.3 LIMITATIONS OF THE SOLUTION

The main limitation in our approach is that we are totally dependent on the vector embeddings model to create the embeddings properly. If the embeddings chosen is unable to properly relate different medical terms properly then it may fail in certain cases. That's the only limitation of this solution.

CHAPTER 6
CONCLUSION, FUTURE WORK

6.1 CONCLUSION

The development process involved creating a highly advanced eXplainable Artificial Intelligence (XAI) chatbot. This chatbot is equipped with an expansive knowledge base capable of handling both diagnosis and treatment recommendations. It's integrated seamlessly within a sophisticated web application that boasts an exceptional user interface and user experience. The web app is designed to offer users a smooth login experience and the convenience of accessing their previous chat conversations effortlessly. This integrated system ensures a comprehensive and user-friendly platform for obtaining accurate medical guidance and maintaining a record of interactions for users' convenience and continuity of care.

6.2 FUTURE WORK

There is an immense possibility of future work possible -

1. Create a custom embedding model, and train it rigorously
2. Fine tune the LLM chatbot with various clinical conversations, so that it doesn’t completely depend on the embeddings Vector database for the answers and has some of its own brain

REFERENCES

- https://arxiv.org/abs/1706.03762
- http://jalammar.github.io/illustrated-bert/
- Amin, D.; Garzón-Orjuela, N.; Garcia Pereira, A.; Parveen, S.; Vornhagen, H.;
- Vellinga, A. Artificial Intelligence to Improve Antibiotic Prescribing: A Systematic
- Review. Antibiotics 2023, 12, 1293. https://doi.org/10.3390/antibiotics12081293
- Das, S., Biswas, S., Paul, A., Dey, A.: AI doctor: an intelligent approach for medical diagnosis. In: Bhattacharyya, S., Sen, S., Dutta, M., Biswas, P., Chattopadhyay, H. (eds.) Industry Interactive Innovations in Science, Engineering and Technology. LNNS, vol. 11, pp. 173–183. Springer, Singapore (2018).
- Arsene Fansi Tchango, Rishab Goel, Zhi Wen, Julien Martel, Joumana Ghosn: DDXPlus: A New Dataset For Automatic Medical Diagnosis https://doi.org/10.48550/arXiv.2205.09148
- https://faiss.ai/index.html
- https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/
- https://huggingface.co/hkunlp/instructor-large
- https://huggingface.co/hkunlp/instructor-xl
- https://huggingface.co/spaces/mteb/leaderboard
- https://gist.github.com/VictorTaelin/d293328f75291b23e203e9d9db9bd136
- https://ruslanmv.com/blog/How-to-connect-to-Sagemaker-Notebook-via-SSH
- https://accessmedicine.mhmedical.com/book.aspx?bookID=3095