SymptoCare: A Speech Recognition and Translation System For Medical Communication

1st Talha Ahmed
24100033@lums.edu.pk

2nd Syed Muqeem Mahmood
24100025@lums.edu.pk

3rd Faizan Ali
24100065@lums.edu.pk

4th Maryam Shakeel
24100295@lums.edu.pk

5th Nehal Ahmed Shaikh
24020001@lums.edu.pk

6th Ubaid Ur Rehman
24020389@lums.edu.pk

***Abstract*—Effective communication between patients and healthcare providers is the cornerstone of quality medical care. Patients, however, hail from diverse cultural backgrounds with unique languages, making linguistic diversity a significant challenge. Lack of access to reliable medical advice and miscommunication with doctors due to language differences can lead to adverse outcomes for refugees, immigrants, and inhabitants of areas lacking adequate healthcare facilities. To address this prevalent issue, we introduce a personal healthcare assistant, SymptoCare. This AI-powered assistant uses an SMTS (Speech-to-Machine Translation + Speech) system to bridge the linguistic gaps in healthcare settings. Our system combines speech recognition for transcription, machine translation for language conversion, and text-to-speech for delivering the generated response as an audio. The development of the SMTS involves selecting relevant vector databases, fine-tuning of large language models (LLMs), and employing retrieval augmented generation (RAG) pipelines. Our SMTS can be expected to enable users to receive real-time state-of-the-art medical advice and information.**

Keywords — LLM, RAG, Fine-Tuning, Gradio, Healthcare

I. Introduction

Language barriers in healthcare are a significant challenge that can lead to misunderstandings, misdiagnosis, and even incorrect treatment plans. This is particularly problematic in multicultural societies or regions where patients and healthcare providers may need to share a common language. Traditional translation methods, such as human interpreters or basic translation software, often fall short. They can be slow, leading to delays in urgent care situations. They can also be inaccurate, especially when dealing with complex medical terminology, potentially leading to severe consequences for patient care. To address these challenges, we propose a Speech to Machine Translation + Speech (SMTS) system that incorporates state-of-the-art algorithms in the fields of Automatic Speech Recognition (ASR), Neural Machine Translation (NMT), and Text-to-Speech (TTS) synthesis to enhance communication in healthcare settings, making it more efficient and reliable.

The outline of this report is as follows: Section II delivers a brief overview of our proposed architecture and visualizes the flow of Pipeline 1 and Pipeline 2. In Section III, we explore various design choices we had to consider during the developmental phase of our architecture, detailing various alternatives, experimentation, and insightful results. Following this, Section IV describes the deployment of the final model on a global server, providing details about the provider and the web application design. Sections X and XI share discussions and the final results of the deployed model. The report concludes by stating the individual contribution of each group member in developing this project, along with acknowledgment from all group members.

II. Proposed Architecture

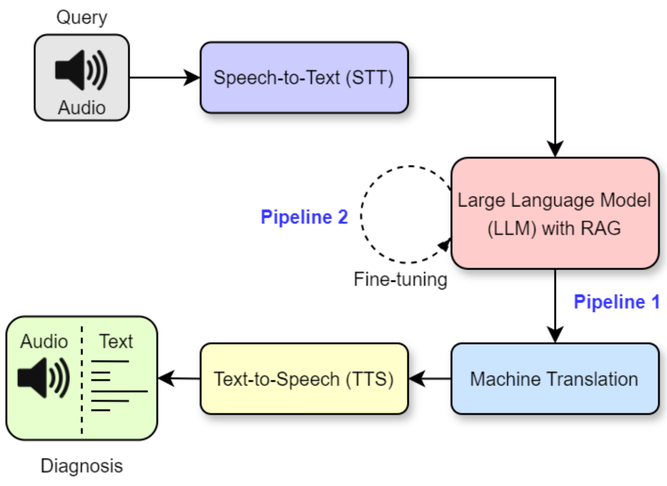


Fig. 1. Proposed system architecture

The first step in our system (Figure 1) is receiving a user's query as an audio input. This is expected to be the patient's symptoms, expressed in the user-specified source language, that are to be used for diagnosis. However, to facilitate later processes and ensure language-independence for our pipelines, we transcribe the text into English via OpenAI's Whisper, a speech-to-text transformer, before feeding it to the model. In Pipeline 1, the standardized input texts undergo processing through Mixtral 8x7B, a pre-trained large language model, that is employed to fetch medically accurate and appropriate diagnoses as responses to the patients' symptoms. Pipeline 2, on the other hand, incorporates the large language model that has undergone fine-tuning using a carefully curated dataset of medical documents. It should be noted that this is expected to yield better responses simply because language models have been decisively shown to perform better when fine-tuned on more context-relevant data. However, it is also important to note that this improvement in performance is crucially dependent on the time and resources invested in fine-tuning (refer to **Experiments** for further details in this regard). Moving on, the English text response is first translated into the user-specified target language through MarianMT, a neural machine translation framework, and then converted to audio using the Google Translate's text-to-speech API.. This final output is then played for the patient, allowing them to receive expert diagnoses tailored for their symptoms.

III. Experiments

Moving on from the functionality point of views of the various modules, it's imperative to know that this wasn't the case from the beginning. We experimented with several vector databases, LLMs, and RAG pipelines, datasets and various Transcription, Machine Translation, Text to Speech modules. Here's a deeper dive into our experimentation process:

A. *Vector Databases*

Initially, we tested several vector databases such as Pinecone and Chroma. Given our budget constraints, Chroma was a clear choice, providing the functionality we needed without the associated costs. Our decision was also heavily influenced by not only its open source nature, but also due to its user-friendliness and flexibility to our needs.

B. *Large Language Models (LLMs)*

Our experimentation with LLMs was extensive, as we utilized APIs from OpenAI, Mistral, and Replicate to explore a range of both open-source and proprietary models. We tried several models, including GPT-3.5, GPT-4, Mistral-8b, Mixtral-8x7b, LLaMA 2 7b, LLaMA 3 8b, and LLaMA 3 70b. Among these, LLaMA 3 70b and GPT-4 stood out for their performance in conjunction with our RAG pipeline. However, considering the computational demands and the potential financial implications of deploying such powerful models in a production environment, we opted for smaller, more manageable models like the tiny LLaMA and Mistral. These models provided a balance between capability and cost, a trade-off crucial for the practical deployment of our system.

C. *Datasets*

The initial setup of the vector databases was necessary because of the amazing flexibility it provides when training LLMs on a vast amount of corpus, as it gives access for cloud storage spaces which is definitely a key factor when storing the embeddings for the data.

As explained above, the Pipeline 1 was not intended to be fine-tuned on any medical curated dataset. For Pipeline 2, however, we had to find appropriate medical datasets such that the pre-trained models can be fine-tuned without any hindrance. This would allow the generation of more contextualized answers to the user's query rather than the model fetching over the entire corpus it's trained upon. Upon further research, we were able to access the (MeDAL), Disease-Symptom Dataset, Disease Symptoms and Patient Profile dataset and MS² datasets.

However, its imperative to note that these datasets are a collection of corpus on various medical studies rather than a typical symptom-diagnosis feature datasets per the problem we have already specified. In the context of defining the machinery of the model, we went ahead with the MeDAL dataset solely for test purposes. We ended up using the Disease Symptoms and Patient Profile and MS² datasets. These datasets were combined to create a single *jsonl* file that served as our final dataset for fine-tuning our model. However, it should be kept in mind that we can not possibly expect to train on a typical layman CPU a model with 4billion+ parameters and achieve remarkable performance.

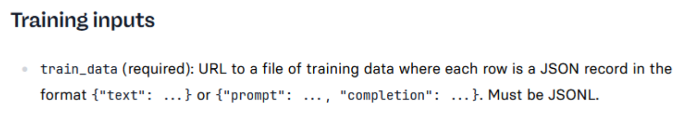
In the Disease Symptoms and Patient Profile dataset, we removed the patient details and only the symptoms columns were kept as features. Further pre-processing on the dataset included dropping irrelevant columns like ”Age” and ”Gender”. The next step was to include two columns in the dataset for ”prompts” and ”results”. This was done by checking all the symptoms the patient had and telling the result of the diagnosis by looking at the ”Outcome” label. If the ”Outcome” was positive the patient was told what disease they have. If the ”Outcome” was negative the patient was labeled as having no disease.

The other data set that was used for fine-tuning the model was the Disease-Symptom Data Set. With 773 unique diseases (e.g., panic disorder and vocal cord polyp), 377 one-hot encoded symptoms (e.g., anxiety and insomnia), and 246,000 samples, this dataset offers valuable information for research related to disease symptoms. The binary nature of symptoms allowed us to easily construct sets of symptoms and a likely diagnosis based on those symptoms. After this pre-processing, we had another sizeable data set that we could use to fine-tune our model.

More details regarding fine-tuning follow, so from this point onward, we are set on focusing on the machinery of the SMTS model and acknowledging the setbacks we faced due to computational or time constraints.

D. *Fine-tuning LLMs*

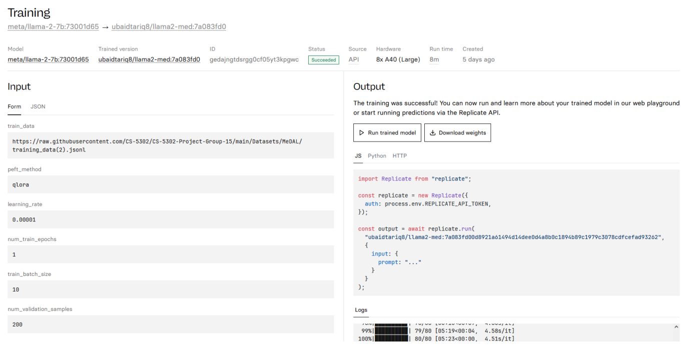
Another important aspect of the project's experimentation was the fine-tuning of LLMs on medical data. For this, we explored 3 datasets as described above, and chose MS^2 as we found it most suitable to fine-tune an LLM on without extensive pre-processing. The dataset was formatted in a JSONL file according to Replicate's requirements as format ”text”: ... (shown in Figure 2). Due to computational constraints, we chose LLaMA-2 7b model to fine-tune using Replicate's API, which uses a cluster of 8xA40 Nvidia GPUs to train the model. We were only able to run 1 epoch over a 1000 training examples due to financial constraints (even that cost us around \$5, almost 1400 PKR). This lack of training time and limited computational power made it so that the trained model itself



Training inputs

- `train_data` (required): URL to a file of training data where each row is a JSON record in the format `{"text": ...}` or `{"prompt": ..., "completion": ...}`. Must be JSONL.

Fig. 2. Format for Fine Tuning LLM on MeDAL Dataset



Training
meta/llama-2-7b:73001d65 → ubaidtariq8/llama2-med:7a083fd0

Model	Trained version	ID	Status	Source	Hardware	Run time	Created
meta/llama-2-7b:73001d65	ubaidtariq8/llama2-med:7a083fd0	gedajngtdsrgg0cf05yt3kpgwc	Succeeded	API	8x A40 (Large)	8m	5 days ago

Input

train_data	`https://raw.githubusercontent.com/CS-5302/CS-5302-Project-Group-15/main/Datasets/MeDAL/training_data(2).jsonl`
peft_method	qlora
learning_rate	0.00001
num_train_epochs	1
train_batch_size	10
num_validation_samples	200

Output

The training was successful! You can now run and learn more about your trained model in our web playground or start running predictions via the Replicate API.

```
import Replicate from "replicate";

const replicate = new Replicate({
  auth: process.env.REPLICATE_API_TOKEN,
});

const output = await replicate.run(
  "ubaidtariq8/llama2-med:7a083fd00d8921a61494d14dee0d4a8b0c1894b89c1979c3078cdfcefad93262",
  {
    input: {
      prompt: "..."
    }
  }
);
```

Logs

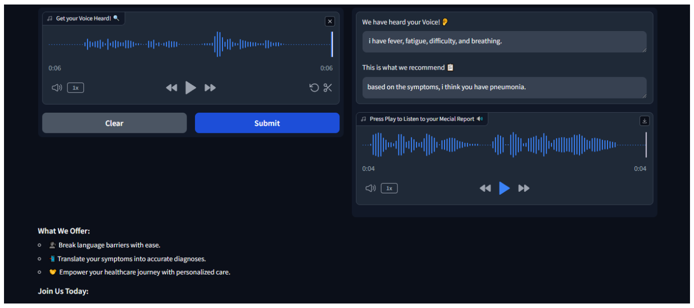
```
 99%|██████████| 79/80 [05:19<00:04,  4.58s/it]
100%|██████████| 80/80 [05:23<00:00,  4.51s/it]
```

Fig. 3. Llama 7B being fine tuned on MeDAL Dataset

did not perform very well. Figure 3 shows a sample screen shot from when we were fine-tuning it.

Another fine-tuning approach we took employed Gradient AI, a web app that lets you train and fine-tune models through its interface. The only problem we faced with Gradient AI was that the maximum number of tokens that could be fine-tuned at once was 100. Our dataset for fine-tuning contained approximately 1300 samples. This issue was resolved by doing batch-wise fine-tuning by taking 100 samples iteratively and storing the results of the previous batch of fine-tuning. This way, we could fine-tune the entire dataset using a clean notebook. The models available on Gradient AI for fine-tuning were Mistral, LlaMa, and nous-hermes). For our fine-tuning, we used nour-hermes as it supports more tokens than the rest of the models. The fine-tuned model was also available to us on Gradient AI's interface, and we could fine-tune it later from the interface if we wanted to. The only reason this was not possible was due to the maximum token limit of training only 100 samples, hence we had to resort to training on Google Colab. In the end, Gradient AI's fine-tuned model was not integrated with our deployed code because of contradictory dependency issues for Gradient AI and Gradio. Hence, the code for Gradient AI's fine-tuning is provided separately for any testing.

Another approach we took for fine-tuning the model was using the Unsloth library, which supports some of the most commonly used large language models, such as LlaMa 3 and Mistral; we used the latter for our project, specifically *mistral-7b-bnb-4bit*. Unsloth allowed us to fine-tune the model more than two times faster while saving more than 70% memory, as it uses QLoRA, a recent fine-tuning approach that backpropagates gradients through a frozen, 4-bit quantized pre-trained language model into low rank adaptors and hence



Get your Voice Heard! 🔍

We have heard your Voice! 👂

i have fever, fatigue, difficulty, and breathing.

This is what we recommend 📋

based on the symptoms, i think you have pneumonia.

Press Play to Listen to your Mecial Report 🔊

What We Offer:

- Break language barriers with ease.
- Translate your symptoms into accurate diagnoses.
- Empower your healthcare journey with personalized care.

Join Us Today:

Fig. 4. A fine-tuned Mistral 7B's performance on our custom merged data set

theoretically saves a phenomenal amount of space and time. We confirmed this practically in our experiments, as this approach lets us fine-tune a 7B-parameter model using a data set consisting of more than 246000 observations for free on a mere T4 GPU within minutes. To be more precise, it supports 4-bit pre-quantized models that enable 4 times faster downloading while avoiding out-of-memory errors. After the fine-tuning was complete, we uploaded the model to Hugging Face to be able to clone it later when we are examining the model's performance using Gradio. The model ultimately worked quite smoothly on Gradio, as can be seen from Figure 4. It is to be noted, however, that, unlike the case in Pipeline 1, we could not fully make the model such that it generates personalized responses primarily due to the absence of such a large and rich data set (at least as far as we know). It is clearly dangerous to provide specific medical guidance via an AI agent, so our model is limited to generic responses and patients are advised to take the model's diagnoses as starting points and consult an actual doctor.

E. RAG Pipelines/Libraries

We constructed two distinct RAG pipelines using LangChain and LlamaIndex to test which would better suit our needs. LangChain, being an older and more established framework, offered extensive online support and a robust community. However, after rigorous testing and evaluation, LlamaIndex emerged as the superior choice for our specific requirements. It is particularly designed for ”connecting custom data sources to large language models,” a feature we found to be not only accurate but incredibly effective. LlamaIndex also excels in user-friendliness and comprehensive support for RAG applications, despite its relative novelty and the lesser availability of external resources compared to LangChain.

Moving on from the stuff that takes the brunt of the computational expense, it was imperative we found quick hacks to the other essential modules that made the essential connection between the user and the AI agent (referring to the LLM) at the backend.

F. Prompt Engineering

In the rapidly advancing field of Artificial Intelligence and large language models (LLMs), the proliferation of data has led to the development of increasingly complex models. However, accessing and fine-tuning these models can pose significant challenges, especially for individuals with limited compute and GPU resources. Despite efforts to fine-tune models using subscription resources like Replicate and open-source datasets, the process remains non-trivial, particularly due to the scale of available data. For instance, fine-tuning on a small fraction of the total dataset, such as 1000 samples out of 256k in the Disease-Symptom Data Set, can limit the model's performance, even when utilizing free resources like the T4 Colab GPU. This underscores the need for alternative strategies, such as prompt engineering, to optimize model performance within the constraints of available resources.

Prompt engineering, in essence, offers a supervised approach to enhancing the performance of large language models (LLMs) compared to the unsupervised fine-tuning method. The core concept involves optimizing model performance not by adjusting billions of parameters to fit a specific dataset, but rather by refining the formulation of user queries.

In our model's architecture, user inputs come in various forms, representing more than just a simple list of symptoms. When these queries are sent to the LLM as-is, it's possible that the symptoms hidden within them are accompanied by unnecessary tokens. Due to the LLM's longer context window, it may not perform optimally relative to a query with a smaller context window, especially in scenarios like ours where the dataset is relatively small.

Therefore, the initial approach to prompt engineering for improving LLM performance involves taking the user's query and passing it through a hypothetical model. This model aims to output a concise and more meaningful representation of the query, focusing on the most likely symptoms. Subsequently, the refined query is fed to the LLM, enabling it to generate a more contextually accurate response for the user.

To initiate the operation of the hypothesized model, we referred to the data described in the *Datasets* section. The features, i.e., the symptoms, totaled around 380, each represented as a binary feature: 1 if the symptom is present and 0 if it's not. However, classical machine learning theory suggests that models with high-dimensional features suffer from the "curse" of dimensionality. Fortunately, considerable work has been done in this area to mitigate this curse, with several algorithms developed to reduce dimensionality. One of the most common methods is Principal Component Analysis (PCA). However, we chose to explore a relatively new algorithm called Uniform Manifold Approximation and Projection (UMAP).

UMAP, or Uniform Manifold Approximation and Projection, is a dimensionality reduction technique that preserves more of the local structure of the data compared to PCA. It achieves this by creating a low-dimensional representation of the data while preserving its global structure, making it well-suited for various machine learning tasks, including clustering and visualization.

The approach was straightforward: we condensed the binary features of each disease into a lower-dimensional subspace, with the dimensionality of the subspace being a hyperparameter. If there were multiple distinct feature vectors for the same disease, we took their mean. This approach aligns with the intuition of emphasizing symptoms closely associated with a particular disease while downplaying relatively infrequent symptoms. By condensing these vectors into a lower-dimensional subspace, we theoretically obtained a latent representation of each disease in our dataset as a lower-dimensional feature vector. This latent representation served as the ground label for each disease we aimed to predict from the user's query.

Next, the user's query, containing unnecessary tokens, was utilized to extract keywords (i.e., symptoms). By mapping these keywords back to the classic binary feature vector using indices, the prediction was determined based on the disease with the highest dot product between its ground label vector and the user's feature vector.

However, despite the apparent simplicity and effectiveness of this approach, it fell short of integrating with the LLM, which is not in line with the ideal concept of prompt engineering. Therefore, adhering to the principle of Occam's Razor, we opted for a simpler solution: extracting keywords from the user's query from the overall symptom list defined in the data and passing them directly to the LLM. Although this solution may lack the dramatic impact initially anticipated, it resulted in better contextualized responses.
Building on this idea and drawing inspiration from how *GPT-Researcher* generates reports from user queries, we decided to personalize the LLM agent. Thus, we provided the LLM with the following fixed text along with the extracted symptoms from the user: 'You are a medical doctor. A patient has come to you in desperate need of help. Provide as accurate a diagnosis as possible using the listed symptoms.' This approach yielded remarkable improvements over the previous method of simply sharing symptoms. Not only did it accurately identify diseases, but it also contextualized responses and provided suggestions.

G. *Speech to Text - Transcription*

The connection begins from the user's plea of her symptoms (ideally) in the form a *.wav* file recording. It was a must that the transcription of this audio file happen as soon as possible for the LLM agent to use **RAG** to fetch the user's diagnosis (ideally). We started off with one of the standard Whisper models (Github Link). However upon experimenting with simple audio recordings, we observed it was not fast enough. Luckily upon further research we found the a variant of Whisper - *FasterWhisper* (Github Link) which is a godsend module reimplemented using CTranslate2, which is a fast inference engine for Transformer models. As explained in the GitHub, it has achieved upto 4× better inference time relative to standard Whisper. To test this hypothesis, we set-up a task to transcribe an audio file spanning 5 seconds by every size of whisper models available in both varients. The tests are performed in Kaggle notebook using GPU T4 x2 accelerator. The wall-time to transcribe the audio into English was computed using `%time` command. The results are summarized in Figure 5.

	tiny (39 M)	base (74 M)	small (244 M)	medium (769 M)	large-v2 (1550 M)
whisper	1.83 s	3.6 s	4.82 s	14.1 s	1min 8s
faster-whisper	909 ms	1.16 s	2.73 s	7.29 s	14.1 s

Fig. 5. Whisper transcription results

H. Machine Translation

After the model has transcribed the user's query and has been received by the AI agent and fetched an appropriate response, next came the step of translating the language in the user's language (assuming its not the language of the AI agent i.e. English) per the project proposals one of the most essential goals of eliminating linguistic barriers between patient and doctors. We opted to use the family of MarianMT short for Machine Translation models. Unlike before, we did not face any computational pressure here and the model used was `Helsinki-NLP/opus-mt`. The Helsinki-NLP models comprise of over 1400 translation models covering hundreds of lanugage pairs.

We observed nice quick and accurate inference for target languages pertaining to Arabic, French, Deustch, and various other common languages. However logically speaking as many languages stem from semantic languages like Arabic e.g Urdu, it is likely to perform up to the expectations set by its parent.

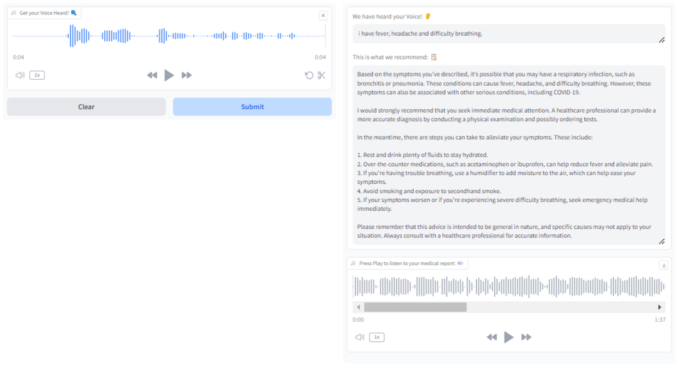
I. Text to Speech - TTS

Essentially the model could have ended its pipeline here and outputted the AI agent's response in the user's language however, we inched one step ahead and decided to translate the diagnosis into speech because it is much possible that people using this app in a ideal perfect scenario might have literary difficulties with Pakistan having nearly 40% of its population living below the poverty line. So having the answer in a speech format not only prevents these kind of pitiful scenarios but also mimics the classic patient - doctor face to face conservation which otherwise is not possible due to multitude of factors.

We experimented with the famous **Text to Speech** model like *WaveNet* (Github Link) and *Tacotron2* (Github Link) however, we received an anti-climactic conclusion. The simple, user-friendly, fast and free *Google Translate Text to Speech* - *gTTs* topped both models. And as per *Ocam's Razor* we decided to keep things simple and use gTTs. However, its imperative to note that it does not cover a wide range of languages which others models most probably do however for the current context scenario it does more than enough on a set of common languages.

IV. Deployment

Once the model architecture is ready, its time to deploy it on a global server for real-time usage. We have used Gradio for this purpose, which is a user-friendly open-source Python



We have heard your Voice!

i have fever, headache and difficulty breathing.

This is what we recommend:

Based on the symptoms you've described, it's possible that you may have a respiratory infection, such as bronchitis or pneumonia. These conditions can cause fever, headache, and difficulty breathing. However, these symptoms can also be associated with other serious conditions, including COVID-19.

I would strongly recommend that you seek immediate medical attention. A healthcare professional can provide a more accurate diagnosis by conducting a physical examination and possibly ordering tests.

In the meantime, there are steps you can take to alleviate your symptoms. These include:

1. Rest and drink plenty of fluids to stay hydrated.
2. Over-the-counter medications, such as acetaminophen or ibuprofen, can help reduce fever and alleviate pain.
3. If you're having trouble breathing, use a humidifier to add moisture to the air, which can help ease your symptoms.
4. Avoid smoking and exposure to secondhand smoke.
5. If your symptoms worsen or if you're experiencing severe difficulty breathing, seek emergency medical help immediately.

Please remember that this advice is intended to be general in nature, and specific causes may not apply to your situation. Always consult with a healthcare professional for accurate information.

Press Play to listen to your medical report:

Fig. 6. Gradio interface: Responce to a sample query

package that allows fast deployment of web application for our machine learning model without any CSS, JavaScript or web hosting experience.

The audio retrieval is done using Gradio's built-in component for audio retrieval. The audio recording through this component is unsuitable for transcription down the pipeline. This audio is pre-processed first by setting its `sampling rate` to `16,000 Hz`, converting it to a single channel, changing its datatype, and normalizing the values before conversion into an array to make it suitable for the Whisper module.

Considering our limited compute resources, we have used the `tiny` faster-whisper model, followed by response generation from the fine-tuned LLaMA-2 7b model on MeDAL dataset as described in **Experiments**. The English-to-any MarianMT family is used for machine translation, followed by speech generation using Google Cloud's TTS. Finally, the text response, accompanied by a generated speech in the user's language, is displayed in the output block of the web application.

An example of the model response in the Gradio app is shown in Figure 6. The response is descriptive and precise to the user's query. There is an additional option to save the diagnosis audio in the local directory.

V. Discussion

Let's further discuss the hurdles encountered in developing our model's architecture and how we resolved these issues. We will also discuss the areas in our development process where our implementations and tools worked well enough not to require additional improvements.

A. FasterWhisper

Initially, we used Whisper for transcription in our model, but later we switched to (FasterWhisper). The results showed by Whisper and FasterWhisper were mostly similar, but as FasterWhisper had lesser performance overheads and decreased inference time, we permanently shifted to the latter. However, due to limitations in the computation power on our end, as we trained our model on a CPU rather than a GPU, our transcription took significant time with the newfound 10x faster version of Whisper.

B. *Machine Translation*

The next step in our model was to convert the language into the user's language; we used MarianMT (a machine learning model for machine translation). The good thing about MarianMT was that it was accurate and fast in the languages we tested, such as French, Arabic, and English. MarianMT's performance and accuracy with other languages is yet to be determined.

C. *GTTs*

After machine translation, our next step was to convert the text obtained to speech. For this, we started off working with Tacotron2 and Wavenet, which are neural network architectures for converting text to speech. The same problem was encountered in both models, i.e., they took a lot of time to process the text, train the model, and give the output as speech. We wanted our model to be quick in converting text to speech; hence, we dropped the idea of using both these models. We encountered the Google translation text-to-speech library in Python and used it to convert text to speech. The library was simple to use and open-source. It was not only fast but worked with multiple languages. However, it still has language limitations but can work on common languages.

D. *Gradio Audio*

One of the problems we were encountering was to record our audio in real-time and work with that. We used many techniques and incorporated our audio recorder in Python, but that did not work well due to issues with real-time audio recording. Fortunately, we found (Gradio Audio), which helped in the quick recording of audio and giving outputs fast.

E. *Chroma DB*

Chroma DB setup helped store our data as embeddings by indexing our documents. The medical dataset we used had millions of data converted into embeddings and used to generate a response. However, the generated response was not accurate and, therefore, affected the training of the model as well.

F. *Fine-tuning*

For fine-tuning our model we used MeDAL on the Cloud server of Replicate. Unfortunately, it did not provide satisfactory results. Therefore, the next step to take is to find a dataset that would not utilize a lot of computational power and is CPU-friendly. Moreover, an app can be used for medical diagnosis, such as (WebMD). As a future research prospect, we can expand our model to multiple languages that can be trained on large language models with rich and diverse datasets. The only setback for us was to not have computationally powered devices and monetary assistance to train large datasets. This is something that can be worked on in the future. However, we did find some success. For example, Unsloth rendered fine-tuning feasible for our model and hence a huge performance-booster, in terms of both accuracy and inference time, simply because it was able to fine-tune a large model on a large data set for a fair number of iterations. This was a huge improvement because initially our main constraints were space and time, which, in turn, were due to financial constraints. Unsloth was able to greatly improve our performance at virtually zero cost.

VI. Conclusion

We successfully managed to build and deploy an SMTS that has the capability to create an effective healthcare environment that breaks linguistic barriers. Our collective insights and knowledge has significantly shaped the architecture of the system. Firstly, we meticulously evaluated vector databases, LLMs, and RAG pipelines. Our decision to integrate Chroma, an open-source vector database, aligned with our commitment to cost-effectiveness. Balancing capability and cost, we opted for smaller LLMs like LLaMA and Mistral. Additionally, while fine-tuning LLMs, we experimented with the MeDAL dataset, focusing on medical abbreviation disambiguation. We also significantly improved the efficiency of the SMTS by transitioning to FasterWhisper for transcription. For machine translation, MarianMT demonstrated accuracy and speed across the languages. And for converting text to speech, we found Google Translate Text-To-Speech (GTTS) to be the most simple and user-friendly solution. Furthermore, despite resource constraints, we addressed real-time audio recording challenges using Gradio Audio. Fine-tuning LLMs remains a future research prospect and expanding our model to multiple languages also holds promise. We did, however, conclude with a successful fine-tuning and deployment of Mistral 7B on a sizeable data set using Unsloth and GradientAI, which is already a stride in that direction. The code is available at the following Github.

VII. Acknowledgements

We hereby acknowledge the following individual contributions of each team member in our project.

A. *Maryam Shakeel (24100295)*

- **Text-to-Speech Implementations**:
  - Explored various TTS models, including Tacotron2, Wavenet, and gTTs. I wrote code for Tacotron2 and Wavenet only to find out that processing them takes a lot of time; hence, I switched to researching a library/API that quickly converts text-to-speech.
  - Found the gTTs library, which I explored and wrote code that worked in real-time; hence, I selected gTTs for its balance between speed and language support.
  - Tested the gTTs library on multiple language datasets from Kaggle.
- **Deployment with Gradio**:
  - Integrated TTS model with Gradio for user-friendly deployment.

    - Utilized Gradio's Audio component for real-time audio conversion.
- **Balancing Functionality and Resource Constraints**:
    - Chose Chroma vector database for adaptability and cost-effectiveness.
- **Pre-processing Kaggle dataset**:
    - Pre-processed the dataset for diseases found on Kaggle to include prompts and results based on the features given in the dataset. This dataset was to be used for fine-tuning of the model.
- **Fine-tuning dataset using Gradient AI**:
    - I combined the two datasets generated by Nehal and me so that fine-tuning can be done on the combined dataset.
    - I used Gradient AI's "nous-hermes2" model to fine-tune the dataset, which consisted of 1349 rows batch-wise, due to the maximum sampling constraints of 100 rows each time. The fine-tuning gave promising results.
    - Talha, Nehal and I tried to integrated this fine-tuned model with our deployment code but due to dependency constraints we could not do that and had to provide the GradientAI fine-tuned code separately.

B. Faizan Ali (24100065)

- **Machine Translation Model Selection and Integration**:
    - Tested different machine translation models including Google Translate API, OpusMT, and MarianMT.
    - Extensively evaluated each of the machine translation models, considering accuracy, speed, and language coverage.
    - Selected the Helsinki-NLP/opus-mt family of models as our choice due to its robust performance and versatility.
- **Organizing GitHub Repository**
    - Took charge of code documentation.
    - Kept track of the various libraries used and documented the requirements.txt file.
    - Wrote and edited the README.md file on GitHub repository.
- **Writing and Editing Deliverables**
    - Edited the Project Proposal, poster, and final report.
- **Code Debugging**
    - Helped the team with debugging code and fixing compatibility issues.

C. Talha Ahmed (24100033)

- **Project Proposal Draft:** Drafted the project proposal from inception to its various pipelines, datasets and various modules.
- **Github Managment:** Organized the github for our project in the way it is currently.
- **Chroma DB setup:** Nehal and I worked on the machinery for the chroma DB setup from storing embeddings, metadatas, etc using Llama Index.
- **Pipeline 1**: Integrated the RAG pipeline developed by Ubaid and I with other modules to construct complete the machinery of Pipeline 1.
- **Dataset Preprocessing**. Filtered out unncessary datasets and chose MeDAL and preprocessed it into the appropiate format needed for fine-tuning. Provided Ubaid with the 1000 observations in fine-tuning.
- **Deployment**. Maryam and I spearheaded the deployment of the app using Gradio using the inbuilt Gradio Audio.
- **Pipeline 2**. After facilitating Ubaid's fine tuning, integrated the complete pipeline 2 with the deployed app.
- **Fine Tuning Complications with Gradio** From the efforts of Nehal and Maryam's fine tuning, it was clear that the dataset to be fine tuned had to be in JSONL files for some. Therefore, Muqeem and I incoporated the Chroma DB to work with JSONL format documents
- **Prompt Engineering** With Muqeem, we spent considerable time brainstoming a way to improve the performance of LLM. From UMAP to the simple utilization of keywords and personification of LLM, we achieved amazing performance.
- **Poster Design** Helped Muqeem with pointers on what to include, involving architecture designs and providing with tables regarding the experiments.
- **Gradio App Interface** Muqeem and I completely revamped the whole gradio app design, with a step by step guide for the user.
- **Fine-Tune Models Integration with Github** Helped Maryam and Nehal to integrate their fine tuned models with Gradio Interface.

D. Nehal Ahmed Shaikh (24020001)

- **Chroma DB:** worked alongside Talha to create a python script for storing embeddings, metadata, and the rest of the requisites for the Chroma DB setup using Llama index.
- **Gradio:** worked with Muqeem to search for a more sophisticated way of deployment, i.e., one that incorporates both audio and text, for our model using Gradio.
- **Deployment:** worked with Talha and Maryam on debugging the code that used Gradio for the model's deployment, which eventually led to successful deployment.
- **Data sets:** Found and preprocessed key data sets for fine-tuning of the model such that its performance can be significantly improved.
- **Fine-tuning:** Fine-tuned Mistral 7B using Unsloth on a data set consisting of more than 246000 observations of symptoms and diagnoses to boost the model's performance in the medical domain and uploaded the model to Hugging Face for ease of use.
- **Final deployment:** Finalized the fine-tuning process to ensure that the model is improved using the largest data set available (insofar as feasibility is concerned) and that the deployment pipeline is general enough to accommodate all three of our models.

E. Syed Muqeem Mahmood (24100025)

- **Speech-to-Text Experimentation:** Tested multiple models on the web including Facebook's Wav2Vec 2.0 and OpenAI's Whisper family. Narrowed down to Whisper for its user-friendly API and intuitive implementation.
- **Migration to Faster-Whisper:** Encouraged to use Faster-Whisper variant due to its faster implementation of transcription task while keeping the same quality as the original models from OpenAI.
- **System Architecture Diagram:** Made the architecture diagram using `draw.io` tool and by taking advice from Talha and Nehal.
- **Brainstorming Gradio Integration:** Spent considerable time with Nehal browsing multiple examples on the internet available explaining Gradio's usage as a Chatbot.
- **Deployment:** Preprocessed Gradio Audio to be incorporated with the existing pipeline. Assisted Talha and Ubaid in debugging the deployment pipeline.
- **Prompt Engineering:** Spent considerable time with Talha to come up with a simple approach of extracting key words from the transcribed query.
- **Gradio App Interface:** Talha and I worked on designing the Gradio Interface page which includes choosing a universal theme alongwith some minor changes to the input and output blocks for accessibility which include on-demand audio player for medical diagnosis.
- **Project Poster:** Designed the poster on LaTeX using the general guidelines from the handout. Re-designed the whole system architecture on `draw.io` tool by consulting Talha on what components to include in the design.

F. Ubaid Ur Rehman (24020389)

- **RAG Pipelines Development**: Collaborated with Talha to design and implement the Retriever-Augmented Generation (RAG) pipeline, integrating LangChain and LlamaIndex for efficient data retrieval and text generation.
- **LLM Testing and Evaluation**: Conducted extensive testing across various large language models (LLMs) to assess output quality and performance metrics, providing critical data for project decision-making.
- **Replicate API and LLM Fine-Tuning**: Spearheaded the integration of Replicate's API and led the fine-tuning of LLaMA-2 7b model on a specially curated dataset of 1,000 medical texts. Evaluated the model's performance to ensure it met project standards.
- **Vector Database Experimentation**: Performed detailed experimentation with multiple vector databases including Pinecone and Chroma. Analyzed their advantages and limitations to guide the team in selecting the optimal database solution for our project requirements.
- **Pipeline Optimization**: Enhanced the efficiency and accuracy of data processing pipelines by experimenting with various RAG improvements (such as reranking models and hybrid approach RAGs) and fine-tuning parameters, significantly boosting the project's output quality.