# 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 propose an SMTS (Speech-to-Machine Translation System) as a solution 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**

## 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.

**Outline**. The outline of this report is as follows: Section II delves into a brief overview of our proposed architecture, which incorporates state-of-the-art technology from leading technology companies such as OpenAI, Google, and MarianMT. 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

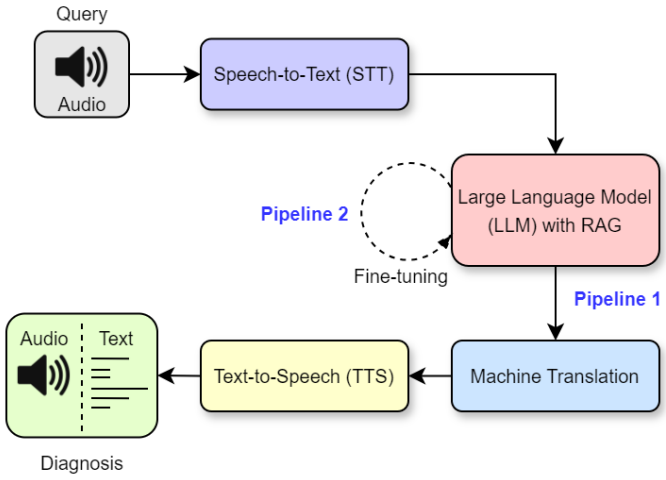


Fig. 1. Proposed system architecture

The first step in our system (Figure 1) is receiving a user's query as an audio input. This is expected to be a 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. Now, in case of the first pipeline, these standardized input texts are next passed on to Mistral 7B, a pre-trained large language model, that is employed to fetch medically accurate and appropriate diagnoses as responses to the patients' symptoms. In case of the second pipeline, the only additional detail is that the large language model has been fine-tuned using a set of curated 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 Marian, a neural machine translation framework, and then converted to audio using Google's Cloud Translation API. This final output is then played for the patient, allowing him or her to smoothly receive an expert's diagnosis

## III. Experiments

Moving on from the functionality point of views of the various modules we have used in our proposed system, 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. We ultimately chose to integrate Chroma into our system. Our decision was heavily influenced by its open-source nature, which aligns with our commitment to supporting and utilizing open-source technology whenever feasible. Chroma proved to be exceptionally user-friendly and adaptable to our needs without imposing the financial strain that Pinecone would have entailed. Given our budget constraints, Chroma was a clear choice, providing the functionality we needed without the associated costs.

### B. Large Language Models (LLMs)

Our experimentation with LLMs was extensive, utilizing 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, 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 LLM's on a vast amount of corpus, as it gives provides 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 but for Pipeline 2

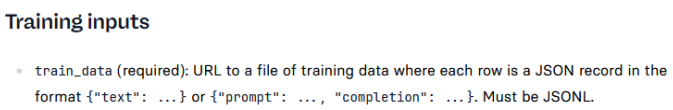


Fig. 2. Format for Fine Tuning LLM on MeDAL Dataset

we had to find appropriate medical datasets such that the pre-trained models can be fine tuned without any hindrance. This will allow more contextualized answers to the user's query rather than the model fetching over the entire corpus its trained upon hence theoretically achieves better performance. Upon further research, we were able to access the MeDAL dataset (Dataset) while other potential datasets included Shaip and MS². The MEDAL dataset is curated for NLP pre-training in the medical domain, focusing on abbreviation disambiguation. It's designed for enhancing models' understanding of medical texts, making it suitable for projects aiming to generate or process medical documentation accurately.

However its imperative to note that this dataset is a collection of corpus on various medical studies rather than a typical symptom-diagnosis feature dataset per the problem we have already specified. However, in the context of defining the machinery of the model we went ahead with this solely for test purposes. We are on the lookout for finding such datasets with the above features however, as has been the theme throughout, we have to keep in mind the limited compute and funding resources we have. We can not possible expect to train on a typical layman CPU a model with 4billion+ parameters and achieve remarkable performance. More details regarding fine-tuning below. Hence from this point onward, we were set on focusing on the machinery of the SMTS model rather than its performance.

### 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 MS2 as we found it most suitable to fine-tune an LLM on without extensive preprocessing. 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 did not perform very well. Figure 3 shows a sample screen shot from when we were fine-tuning it.

### 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

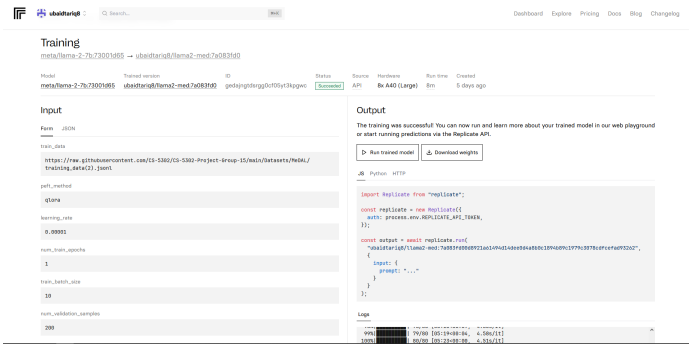


Fig. 3. Llama 7B being fine tuned on MeDAL Dataset

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. 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 of 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. As explained in the github, it has achieved upto $10\times$ 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 of Whisper. The wall-time to transcribe the audio into English was computed using `%time` command. The results are summarized in Figure 4.

| | tiny | base | small | medium | large |
|---|---|---|---|---|---|
| whisper | 1.83 s | 3.6 s | 4.82 s | 14.1 s | 26.4 s |
| faster-whisper | 909 ms | 1.16 s | 2.73 s | 7.29 s | 2.6 s |

Fig. 4. Whisper transcription results

### G. Machine Translation

fter 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 lingustic 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`.

We observed nice quick and accurate inference for target languages pertaining to Arabic, French, Deustch however other languages are yet to be tested upon. 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.

### H. 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 recieved 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 package that allows fast deployment of web application for our machine learning model without any CSS, JavaScript or web hosting experience. 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 by Google Cloud's TTS. The text response text accompanied by speech in user's language is displayed in the output block of the web application.

## 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.

## 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.

## 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.

*B. Faizan Ali (24100065)*

- **Machine Translation Model Selection and Integration**:
  - Evaluated 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.

*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.

*D. Nehal Ahmed Shaikh (24020001)*

Will write later...

*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 after consulting 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.

*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.