CHENNAI MATHEMATICAL INSTITUTE

Vernacular VoiceBot for Customer Support

Srishti Lakhotia
Roll No. MDS202437

Lucky Kispotta
Roll No. MCS202411

Mentors: Pushkar Sathe
Kotak Mahindra Bank

Sarvesh Bandhaokar
MAN Trucks and Bus India

Shashi Satyam
MAN Trucks and Bus India

Alumni-mentored project

Chennai, August 2025

Acknowledgements

We express our gratitude to our mentors Shashi Satyam, Pushkar Sathe, and Sarvesh Bandhaokar who supported and guided us throughout the duration of this project. We also express our gratitude to Mr. Sudhir Kumar without whose support this would not have been possible. In the end we would like to thank Chennai Mathematical Institute for giving us this opportunity to collaborate on this project.

We would also acknowledge the use of *ChatGPT* which we used to proof read and enhance this report and related works.

Abstract

Accessibility to customer service options for sections of the population that are different in terms of language is a very important problem to tackle and resolve in our linguistically diverse country. In this report, we present our work to bridge the gap between *Customer Service* solutions that only support English options by introducing vernacular languages into such arrangements. Hence, bridging the gap between people and services.

Keywords: VoiceBot, Chatbot, Customer support, LLMs, Speech-to-Text, Text-to-Speech.

Contents

This page intentionally left blank.

1 Introduction

1.1 Motivation

The project is the brainchild of our mentors Pushkar and Sarvesh. Accurate, conversational and efficient automation of Customer Service is a growing demand and a product that could do so in vernacular languages has immense scope in the near future.

1.2 Problem Statement

We try to create a proof of concept (PoC) for an AI Powered Voice Chat application which talks back and forth with a customer (user) in their language, just as a human Customer Service Agent would. The user asks the bot their queries and the bot tries to respond with minimal latency, ideally real-time. Given the user questions, the bot tries to reason the validity and sufficiency of the question. Based on which, it tries to find the most appropriate response for the user.

> The most challenging part of the problem is to deal with latency and choose the best of existing solutions.

1.3 Pipeline Diagram

This was the initial proposed pipeline.

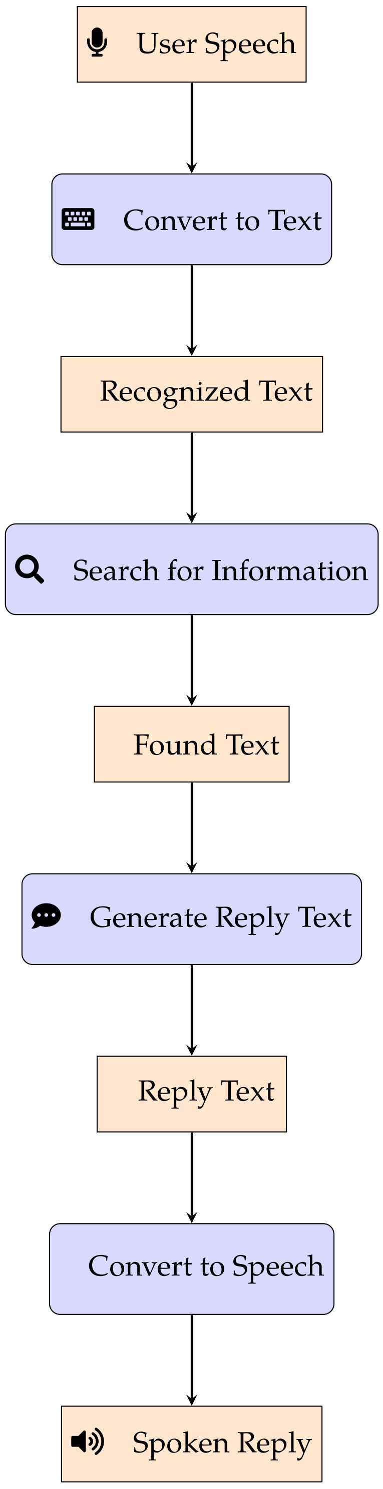


Figure 1.1: *Diagram illustrating the flow of a voice-based chat system using search and generation.*

2 Speech to Text

In this section we discuss the various options and techniques we used for Speech-To-Text (STT) and the relevant research on audio technologies.

2.1 Voice-Activity detection (VAD)

The first step towards recognizing audio from a stream of audio data (array) is to discard chunks of samples which do not contain any human voice. This crucial step ensures that only relevant data is processed decreasing overhead on the system. Additionally, in a chat setup, VAD helps with the problem of interruption: the user may start speaking again before the bot has finished. This should immediately prompt the system to pause or terminate its output speech.

Warning

One can think of a more simple approach to deal with this problem. As one chunk is an array of samples and the array representing the chunk contains the values of amplitudes at that sample. A sample to audio is pixel to image. By averaging the values received in the array of the chunk and comparing it against a threshold one can devise a very simple solution to filter out silent audio chunks. This **fails** as they fail to keep in mind noisy environments.

Here is a list of VAD options we tried:

Table 2.1: *Various VAD tools*

Name	Type	Realtime ?	Offline ?
Google VAD	Proprietary	✓	No.
webrtcvad	Decsision-tree like	✓	✓
Silerovad	ONNX model with convulation and LSTM layers	✓	✓

2.2 Audio formats and datatypes

Pulse code modulation (PCM) is the method used to digitally represent analog signals. Each chunk of an audio consists of multiple samples. A sample can be thought of as a snapshot taken from a continuous audio signal. The value of a sample represents the value of it's amplitude. Two commonly used ways to store PCM samples are using *int16* and *float32* formats. This is a crucial detail as most tools require a specific format and frequency as input.

Table 2.2: *Formats to represent PCM audio*

Type	Range
int16	-32768 to 32767
float32	-1 to 1

Converting one format to another required clipping and dividing by the suitable ranges.

2.3 Audio Preprocessing

We experimented with multiple signal preprocessing packages: *noisereduce*, *RNNoise* and also the default WebRTC preprocessing. When using PyAudio to locally record audio, RNNoise along with a gentle high-pass filter performed the best noise suppression for real-time usecases. The default WebRTC noise and echo suppression seemed to be working fine as well when recording via our browser. However, STT models like Whisper have a lot of noisy data as part of their training data, while also having light internal audio pre-processing explaining why the transcription did not seem to improve significantly when adding noise suppression to raw audio.

2.4 STT Models

We experimented using multiple STT models: whisper, vosk, google. A faster-whisper model fine-tuned on hindi data worked quite accurately. However, the extremely low latency (<2s) offered by the Google Cloud STT model could not be beat.

2.5 Benchmarking STT

We collected *10hrs* of audio data from YouTube to check the accuracy of the model. We used Google's STT API to test. The pipeline achieved an accuracy of *70%* (character-wise). The word-error-rate is around *15%-20%*.

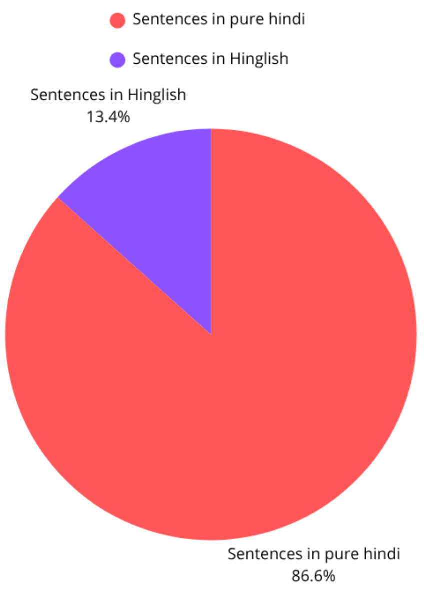


Figure 2.1: *Details about the data used to test the accuracy of the STT pipeline.*

2.6 WebRTC

To enable streaming of audio from client to the user and vice-versa we made use of WebRTC via the *streamlit-webrtc* package. It is an open source solution which addresses the problem of realtime communication between client and the server. It uses **ICE, STUN** and **TURN** servers to set up pathways between two systems (which may or may not be behind firewalls or NATs).

2.7 Translation

The transcribed text is translated to english from Hindi using *Google's* API.

3 LLM

In this section we discuss the *LLM* part of the VoiceBot which deals with given the transcript of the user's query generate appropriate response in text i.e. the intermediate (text-based) chatbot.

Note

To demonstrate a real-world use case. We used publicly available information of *Kotak Mahindra Bank* to create a dummy customer service voicebot for the bank.

3.1 RAG and Vector Storage

For accurate context to the LLM eg. publically available bank information, customer service guidelines, etc., we made use of the popular *Retrieval Augmented Generation* (RAG) technology to add relevant information to the prompt. To implement this we use *ChromaDB* to store data in a vector store and then retrieve them at the time of processing user's query.

The RAG implementation involved the following steps :

3.1.1 Scraping data

We scraped the web and also manually added data in our raw data collection related to *Kotak Mahindra Bank*. We scraped articles from various news sources, Kotak's official site and Wikipedia.

3.1.2 Cleaning data

The steps involved in cleaning the data and preparing them to be stored in the Vector Database included :

1. Break the documents into multiple based on "$\backslash n$ $\backslash n$" which signify double newline.
2. Remove paragraphs which contain less than 20 characters. *Note:* This ensures obsolete and unwanted paragraphs like the options in a website's navigation bar are ignored.

3.1.3 Agentic chunking

We used an *LLM* to make semantic chunks instead of those solely based on punctuation or chunk length. Chunking is important as it breaks big documents into smaller chunks which can be retrieved easily and efficiently. The way we chunk our documents to be stored determines how well the information can later be passed to the LLM for accurate results. The process leverages the power of *AI* to segment paragraphs based on semantics rather than a fixed length. We use *llama3b* model for this purpose. We also make the *LLM* return a **keyword** for each chunk. The keyword may be a newly discovered one or a previously discovered one. Chunks which have the same keyword . This is relevant as instead of searching for the contents in the document while retrieving documents we first filter out documents based on the relevant **keyword** the user is closely related. (Here the notion of closeness is determined by *cosine similarity*).

3.1.4 Vector store

We store the chunked data along with the keyword as metadata in a Vector store (Chroma store). We use the embedding model *all-MiniLM-L6-v2* for this purpose. An embedding model converts text to a vector. The keywords are stored in a seperate vector database.

The embedding model generates vectors of dimension 384.

$$cosine - distance(A, B) = \frac{A.B}{||A||\ ||B||} \tag{3.1}$$

3.1.5 RAG overview

Here is a visual implementation of RAG. *Assume* the user's query is complete and self-contained.

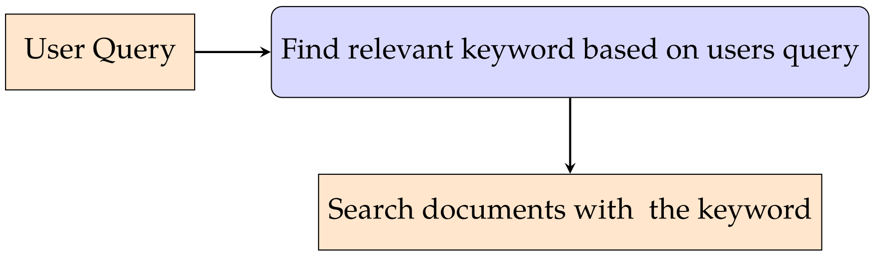

User Query
Find relevant keyword based on users query
Search documents with the keyword

Figure 3.1: *Diagram illustrating the retrieval of documents from vector store based on user's query if the user's query is complete.*

3.1.6 Intermediate Translation

All LLMs seem to perform the best in English. This observation, along with the fact that the bank data that we could web scrape was all in English, we decided to implement intermediate translation to and from English instead of having our entire pipeline work purely in Hindi (or other vernacular tongues). We used Google's translator via the deep-translator package for this purpose.

3.1.7 Conversation

We have implemented a mechanism which makes use of multiple LLM calls to carry conversations. That is, before looking for documents inside the vector store the bot invokes a LLM request with prompt to check if the users query is complete. If not it returns the relevant questions to be asked in the following reply. Internally, a flag is set to signify incomplete ongoing chat thread. If a previously asked question by LLM is answered the flag is reset and normal conversation is carried out. Otherwise it tries clarifying the original sentence till the user asks to quit.
This intermediate chatbot also has a memory. With each user dialogue passed to the LLM, the conversation context is passed alongside for clear understanding and efficient communication.

3.1.8 Tools

The bot allows querying external databases for relevant information. For example, if a user asks for locations near them. The bot asks them their locations and replies with nearby locations. This is done using *MongoDB*'s geolocation query feature.

3.2 Benchmarking

To benchmark the performance of *Text-text* module of the application. We generated artificial questions by inputting the data in vector store in a different *LLM* and asking it to generate questions based on that. About 50 questions were generated. Also, using *LLM* to check if the answers given by the bot for the questions is correct or not. The accuracy achieved was 75%. An example of a response is as follows :

Q: कोटक महिंद्रा बैंक का विजन क्या है?

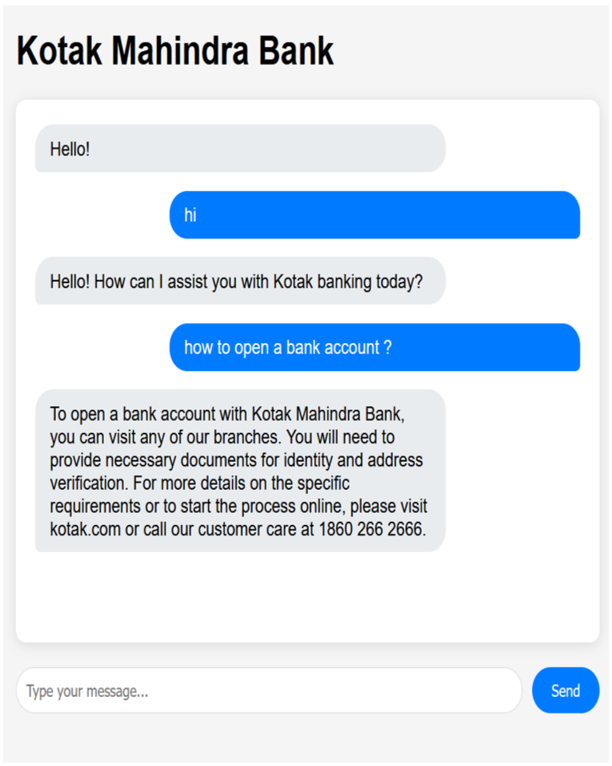
A: कोटक महिंद्रा बैंक की दृष्टि एक वैश्विक भारतीय वित्तीय सेवा ब्रांड है जो अपने ग्राहकों के लिए अनुकूलित, व्यावहारिक समाधान प्रदान करता है। बैंक का उद्देश्य एक विश्व स्तरीय भारतीय वित्तीय सेवा समूह है, जो भारतीय ग्राहकों की जरूरतों की गहरी समझ को बनाए रखते हुए अंतरराष्ट्रीय सर्वोत्तम प्रथाओं के खिलाफ बेंचमार्क किया गया है। कोटक बचत के एक भंडार से अधिक होने की आकांक्षा रखता है; यह एक ग्राहक के ब्रह्मांड में प्रत्येक वित्तीय सेवा के लिए एक ही विंडो होना चाहता है।

3.3 Cloud

We also uploaded the *LLM* part to Cloud. We are using *Google's cloud run* for this purpose. We have deployed the *API* endpoint which can be requested using **POST** request on the *API* and sending the below as body :

```
{
        "messages": ["tell me your locations near chennai india"],
        "thread_id": "2"
}
```

Here the *thread_id* is the unique id of the current chat thread.


Kotak Mahindra Bank

Hello!

hi

Hello! How can I assist you with Kotak banking today?

how to open a bank account ?

To open a bank account with Kotak Mahindra Bank, you can visit any of our branches. You will need to provide necessary documents for identity and address verification. For more details on the specific requirements or to start the process online, please visit kotak.com or call our customer care at 1860 266 2666.

Figure 3.2: *Demo of text-text module on cloud.*

4 Text To Speech

In this section we discuss the final part of the VoiceBot which deals with voicing the LLM output: Text-to-Speech.

4.1 TTS Engines

We initially experimented with the default System Engine and the CoquiTTS Engine. In our final pipeline, we used the gTTS Engine i.e. Google's TTS Engine via the RealTimeTTS package. The gTTS Engine has a few voices to choose from, and an option to set speed of the speech as well, allowing us to give our system a natural customer service feel.

4.2 Major Challenge

The main challenge was figuring out how to properly integrate our TTS pipeline into our streamlit deployment. We were using the *streamlit-webrtc* package for our entire app. Upto this point when there was only incoming audio, the deployment had had no major issues. But to have both incoming and outgoing audio, we realised our then current code would not work. The package has less than enough documentation and there is not much discussion on online forums that fit our problem. It took a while to understand how streamlit-webrtc internally uses threading which leads to it not being able to directly access streamlit's session-state variables.
We settled on a workaround where the TTS output is being saved as a .mp3 file first and then played directly through streamlit, without streamlit-webrtc. This works but it is not truly real-time. However, this way allows the user interrupt the output speech by manually pausing or terminating the speech if needed.

5 Major Challenges

5.1 STT

- Conversion to and from various audio formats.
- Choosing the ”best” STT model, keeping in mind accuracy, latency and cost.

5.2 LLM

- Agentic chunking for our RAG purposes.
- Going from a single LLM call to a more complicated workflow with multiple LLM calls to ensure clear and efficient conversations.

5.3 TTS

- Integration with streamlit-webrtc.

6 Future Improvements

1. **Retry Mechanism:** Currently, if the LLM call fails or the system encounters an error, it defaults to an apologetic output. This could be changed by implementing a retry mechanism where the LLM would iterate a couple more times if it fails the first time.
2. **Prompt Engineering:** We could use interfaces like DSPy which can help iteratively improve prompts till we get the required type of output from our LLM, for a variety of inputs.
3. **Dynamic Prompting:** Having multiple ready prompts, to be used for specific usecases. This would aim to go from a linear workflow to one that resembles a more tree-like structure where another LLM call is made to decide which type of prompt is most suitable.
4. **Interruption mechanism:** The pipeline is currently unable to stop/pause output speech automatically once it detects input speech; it does however allow for manual interruption by the user.
5. **Latency:** The entire pipeline still has more than acceptable latency for it to be truly real-time.
6. **Languages:** Currently, our app is capable of conversing in Hindi. To extend to other vernacular languages, we would likely have to fine-tune STT models in that language. The translation models will also have to be thoroughly tested to check accuracy.
7. **Output speech accent:** For pan-India usage, we could create region-specific accented voices.