# Development of Chatbot Using Speech Recognition

CS 584 Final Report

Basani Priyanka(A20513566) pbasani@hawk.iit.edu

Depala Rajeswari (A20526535) rdepala@hawk.iit.edu

Vinitha Inaganti (A20514310) vinaganti@hawk.iit.edu

## Abstract

Nowadays, many businesses are turning towards rule-based chatbot systems due to their ability to automate repetitive tasks and provide personalized customer experiences. These systems are equipped with advanced natural language processing algorithms, which allow them to quickly comprehend and analyze user queries. Through this technology, chatbots can easily understand the nuances of human language, making them an excellent tool for businesses looking to improve their customer service and reduce workload. By implementing these systems, companies can streamline their operations, cut costs, and enhance customer satisfaction, making it a win-win situation for both the business and the consumer.

The quality of rules and NLP algorithms is crucial for the success of rule-based chatbots. It is important to have thorough, accurate, and up-to-date rules, and NLP algorithms that can understand different subtleties and contextual clues in human conversations. Despite their limitations in handling inquiries beyond their predefined rules, rule-based chatbots still offer several advantages, including round-the-clock support and improved customer satisfaction.

Rule-based chatbot systems have proven to be highly beneficial in facilitating seamless communication between humans and computers, which in turn enhances the user experience. With the continuous advancement of technology, we can expect to see an even more widespread implementation of these systems across various sectors. However, in order for these chatbot systems to deliver optimal responses to user queries, it is important to consistently improve the quality of the rules and natural language processing (NLP) algorithms that are utilized. By doing so, we can ensure that users receive accurate and relevant information, while also minimizing the risk of any errors or misunderstandings.

Key words: natural language processing, speech-to-text, Python, Flask, chatbot,tensorflow, NLTK, Speech Recognition , PyAudio , Keras , Tkinter.

## Introduction

The development of software that resembles human speech is still important currently. The collection of requests and replies serves as the simplest depiction of a conversation. The description of the knowledge base and the operation of the interpreter software are issues in this instance. The knowledge base's markup language may include question patterns, matching answer patterns, the context of the conversations that led up to them, and the title of the relevant communication subject. Further obligations that a chatbot can carry out include audio search, image search, fact search, calculator, weather forecast, and exchange rate display. The majority of these features have online implementations and are accessible via external APIs. The chatbot answer is mechanically generated by a program that analyses and parses the user's text. This algorithm considers the text's morphology and relationship subjects.

**Fig 1: Illustration of Speech to Text transformation**

Because of chatbots, live support employees may handle difficult enquiries that require a human touch. The user is immediately satisfied by obtaining an immediate response to their request, which is more important. As a result, we use Machine Learning (also known as Natural Language Processing) technologies to comprehend consumer enquiries via speech recognition. Chatbots employ text recognition technologies to interpret visible data and respond appropriately. These chatbots are widely utilized in apps that only accept text input, which is a typical feature these days. However, programs such as ChatGPT do not support speech input.

## Data Collection

This data structure appears to be a JSON object representing an intent for a chatbot or virtual assistant.

The "root" object contains a single item, which is an array called "intents" that contains 22 objects. Each of these objects represents a specific intent, which is a user's intention or request when interacting with the chatbot.

In this case, the first intent is called "Greeting". It has several properties, including:

- "text": an array of strings representing different ways a user might greet the chatbot.
- "responses": an array of strings representing possible responses the chatbot might give when greeted.
- "extension": an object with additional information about the intent, such as any functions or responses that should be triggered.
- "context": an object representing the context or state of the conversation, including any input or output context.
- "entityType": a string indicating the type of entity that the intent is associated with, if any.
- "entities": an array of entities associated with the intent.

Overall, this data structure allows developers to design and manage a chatbot's many intents and answers, allowing it to read and respond to user input in a more natural and straightforward manner.

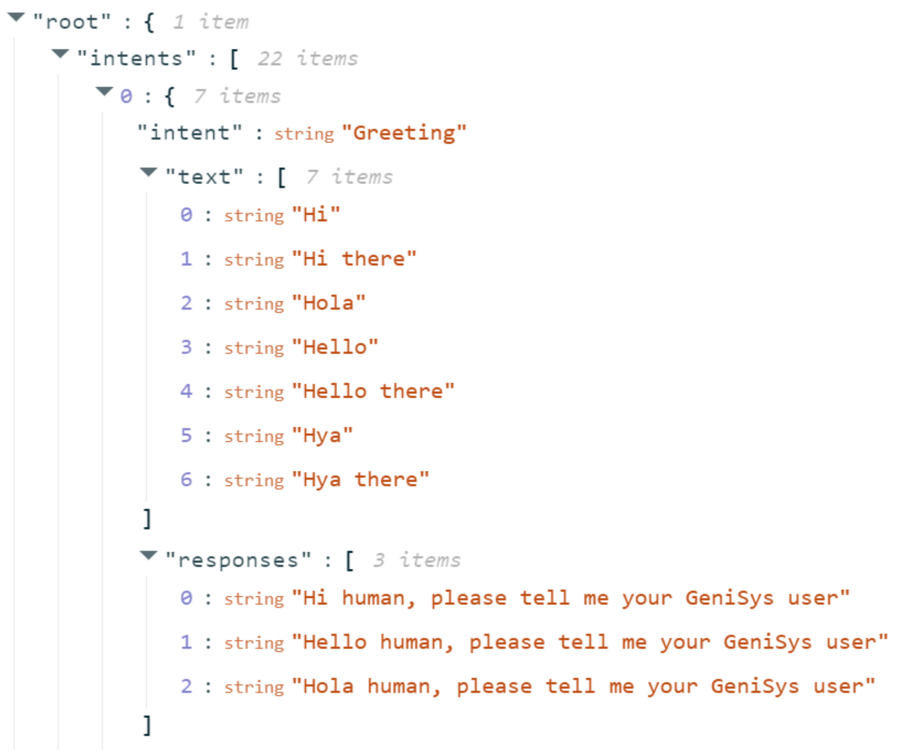

```
▼ "root" : { 1 item
    ▼ "intents" : [ 22 items
        ▼ 0 : { 7 items
            "intent" : string "Greeting"
            ▼ "text" : [ 7 items
                0 : string "Hi"
                1 : string "Hi there"
                2 : string "Hola"
                3 : string "Hello"
                4 : string "Hello there"
                5 : string "Hya"
                6 : string "Hya there"
            ]
            ▼ "responses" : [ 3 items
                0 : string "Hi human, please tell me your GeniSys user"
                1 : string "Hello human, please tell me your GeniSys user"
                2 : string "Hola human, please tell me your GeniSys user"
            ]
```

**Fig : Illustration of dataset .json file**

## Data Preproccessing

The code is intended to preprocess and tokenize a set of text-based intents and labels. It makes use of the nltk library for natural language processing tools, WordNetLemmatizer for word lemmatization, and stopwords for eliminating frequent stop words from text. The preprocess function uses nltk to tokenize the text.word_tokenize, lowercase all the words, delete any non-alphabetic letters, stopwords.words("english"), then lemmatize the remaining words with WordNetLemmatizer. The corresponding intent label is then attached to the labels list, and the resulting preprocessed text is appended to the corpus list.

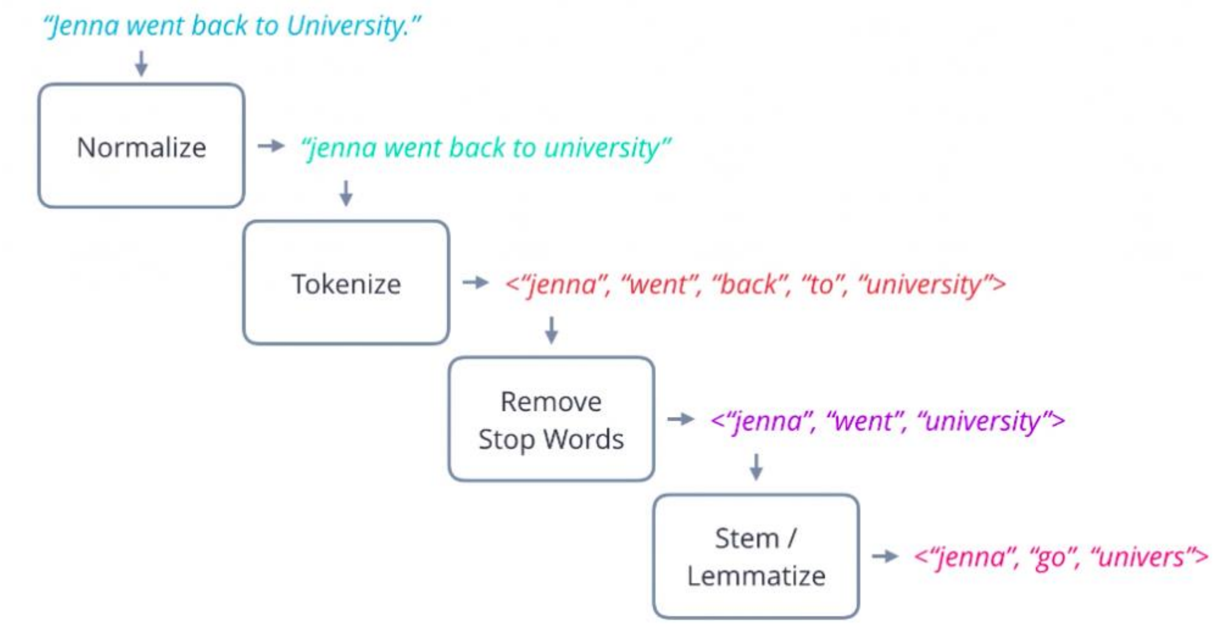
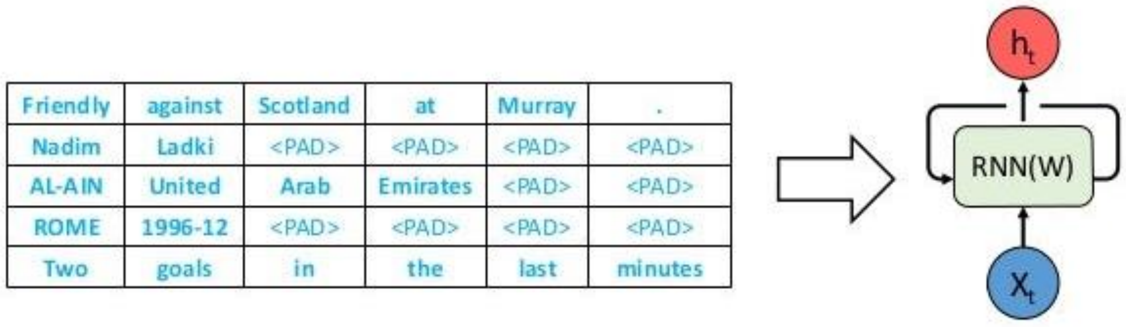

"Jenna went back to University."
Normalize
"jenna went back to university"
Tokenize
<"jenna", "went", "back", "to", "university">
Remove Stop Words
<"jenna", "went", "university">
Stem / Lemmatize
<"jenna", "go", "univers">

**Fig : Data Preprocessing**

After populating the corpus and labels lists, a Tokenizer object is initialized and fitted to the corpus of preprocessed text. This tokenizer creates a vocabulary of unique words and assigns each word a unique index. The Tokenizer object is then used to convert each preprocessed text in the corpus list to a sequence of integers using the texts_to_sequences method. The sequences are then padded with zeros to ensure they are all of the same length using the pad_sequences method from the keras.preprocessing.sequence module.



| | | | | | |
|---|---|---|---|---|---|
| Friendly | against | Scotland | at | Murray | . |
| Nadim | Ladki | <PAD> | <PAD> | <PAD> | <PAD> |
| AL-AIN | United | Arab | Emirates | <PAD> | <PAD> |
| ROME | 1996-12 | <PAD> | <PAD> | <PAD> | <PAD> |
| Two | goals | in | the | last | minutes |

**Fig : Padding Sequences**

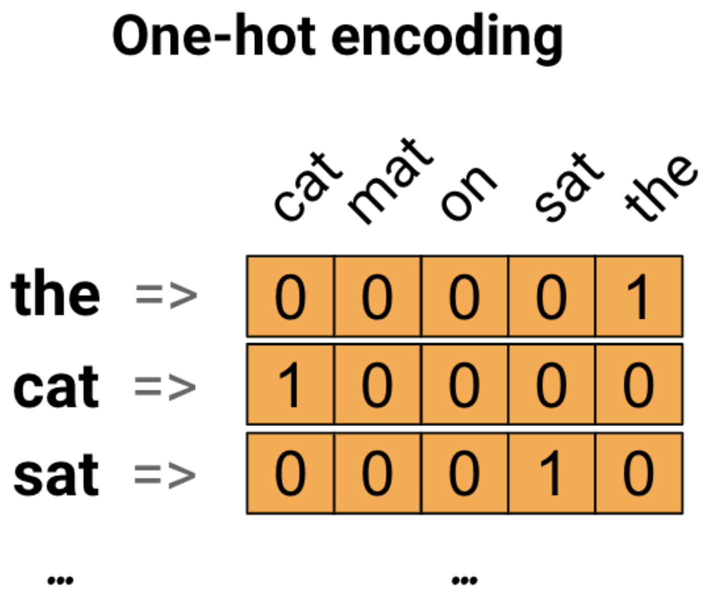
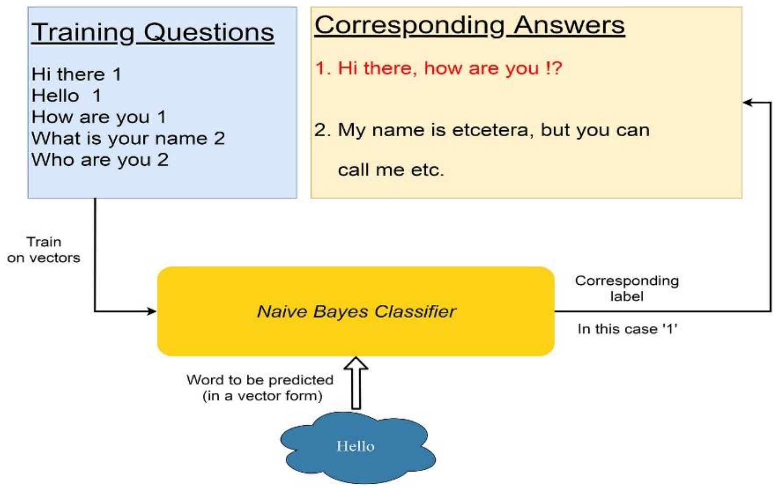


**Fig : One hot encoding**

Finally, the code creates a one-hot encoded version of the labels list. It uses the np.zeros function to create an array of zeros with dimensions (num_samples, num_classes), where num_samples is the number of texts and num_classes is the number of unique intent labels. It then sets the appropriate element in each row to 1 to represent the associated intent label. This one-hot encoded version of the labels list is returned as the output of the preprocessing and tokenization process. Overall, the code provides an efficient and effective way to preprocess and tokenize text-based intent data for use in machine learning models.

## Training The Data

To train a Naive Bayes classifier for text classification, the training data is preprocessed and tokenized into a numerical format. The classifier estimates the class prior probabilities and the class conditional probabilities from the training data. The class prior probabilities are calculated by counting the proportion of training instances that belong to each class. The conditional probabilities are estimated by counting the frequency of individual features in each class and dividing by the total number of instances in that class. Once the model parameters have been estimated, the Naive Bayes classifier can make predictions on new data by calculating the conditional probability of each class given the features of the instance, using the estimated parameters. The class with the highest probability is



selected as the predicted class for the instance.

**Fig : Illustration of queries and responses using Naïve Bayes Classifier**

LSTM networks are a type of RNN that excel in sequence modeling, particularly in natural language processing. They overcome the vanishing gradient issue by having a memory component that maintains crucial context throughout the sequence.

The first stage in training an LSTM network for text classification is to preprocess and tokenize the training data. This entails transforming the raw text into a representation that the network can readily interpret, such as a sequence of tokens or a matrix of word embeddings. The preprocessed data is then divided into training and validation sets, which are used to train and assess the network's performance. The LSTM network is fed sequences of input data and their associated labels during training, and the network parameters are modified to minimize the discrepancy between predicted and true labels. Typically, a loss function such as cross-entropy loss and an optimization algorithm such as stochastic gradient descent are used.

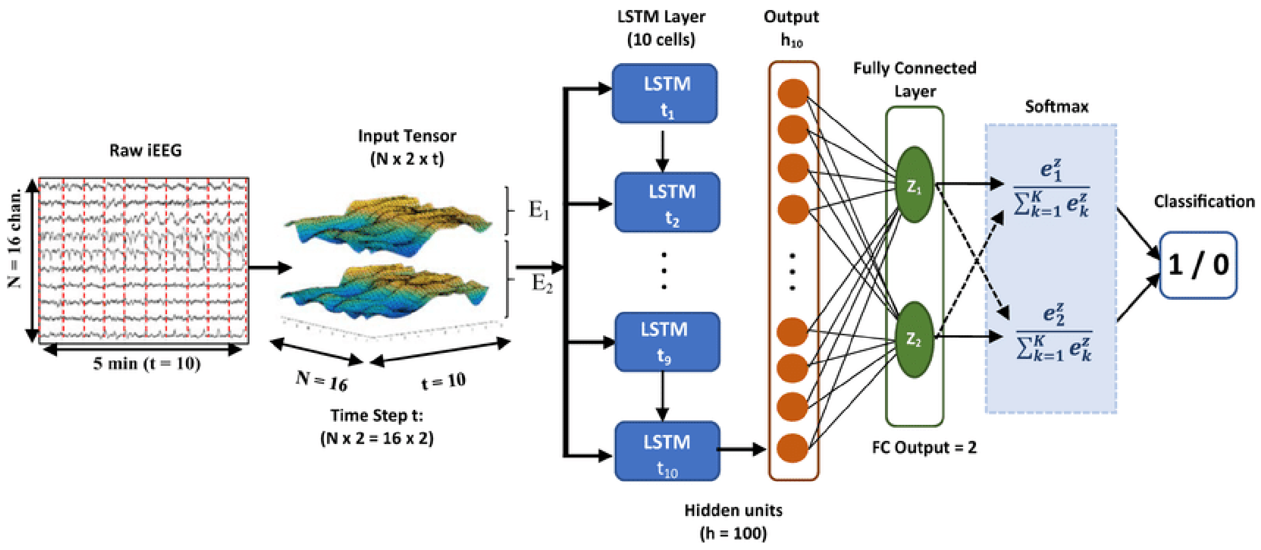


**Fig : Illustration of LSTM model**

Once trained, the LSTM network can be used to categorize fresh text sequences by passing them through the network and making a prediction based on the result at the last timestep. Text classification applications such as sentiment analysis, topic modeling, and spam detection have demonstrated that LSTMs perform well. However, as with all machine learning models, the performance of an LSTM network is determined by the quality of the training data, the selection of hyperparameters, and the task complexity. As a result, it is critical to carefully choose and preprocess the data, as well as tweak the network architecture and hyperparameters to obtain the greatest performance feasible.

## Implementation Using GUI and Speech Recognition

By allowing users to engage with a chatbot using their voice, speech recognition can improve the user experience. The process consists of a few critical phases that must be completed in order for the implementation to be effective.

The first step is to select an API or tool for voice recognition. Google Cloud voice-to-Text, Amazon Transcribe, Microsoft Azure Speech Services, and more voice recognition APIs are available. These APIs can turn audio input into text that the chatbot can process.

For a good chatbot, the conversational flow should handle speech input. This means understanding and responding to various user inputs like greetings, commands, and enquiries. It should be easy for users to interact with.

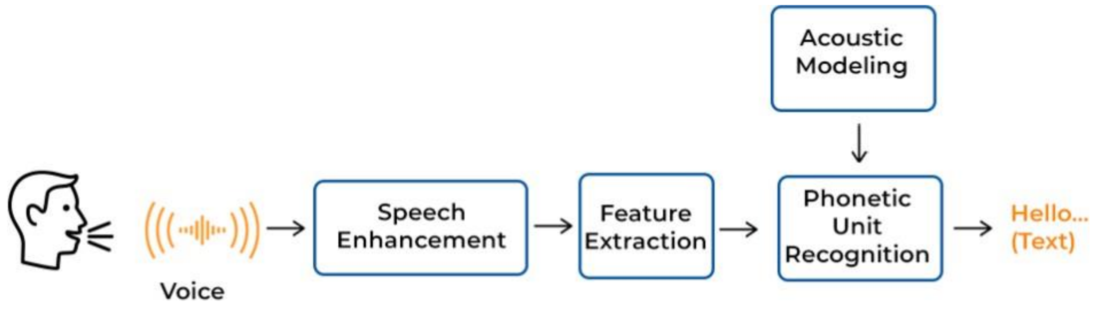
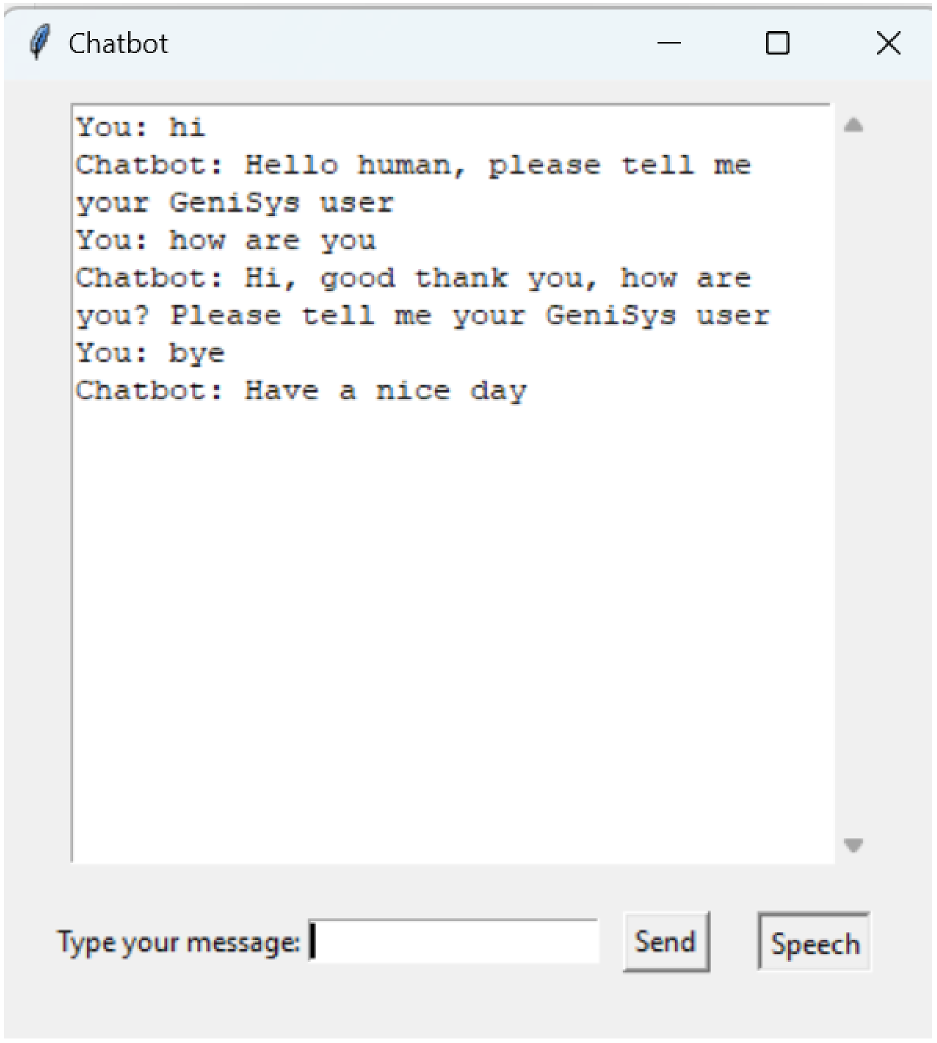


**Fig : Speech recognition process**

To utilize voice recognition, the chatbot requires access to the user's device microphone. The chatbot can simply request permission to access the microphone and subsequently leverage a voice recognition API to convert the user's speech to text. From there, the API can provide a transcription of the user's voice, which the chatbot can then analyze to generate an appropriate response. It is crucial for the chatbot to evaluate and extract key information such as keywords, intent, and entities from the text input provided by the voice recognition API. By doing so, the chatbot can then formulate a relevant response in either text or speech format based on the input received.

Developing a chatbot with voice recognition capabilities is a strategic move that requires careful consideration. But the advantages it can bring to the user experience are undeniable. Voice recognition creates a seamless and natural way for consumers to engage with the chatbot, making it a worthwhile investment for businesses.

## Testing and Validation



```
You: hi
Chatbot: Hello human, please tell me
your GeniSys user
You: how are you
Chatbot: Hi, good thank you, how are
you? Please tell me your GeniSys user
You: bye
Chatbot: Have a nice day
```

**Fig : Chatbot Implementation after taking the valid user response**

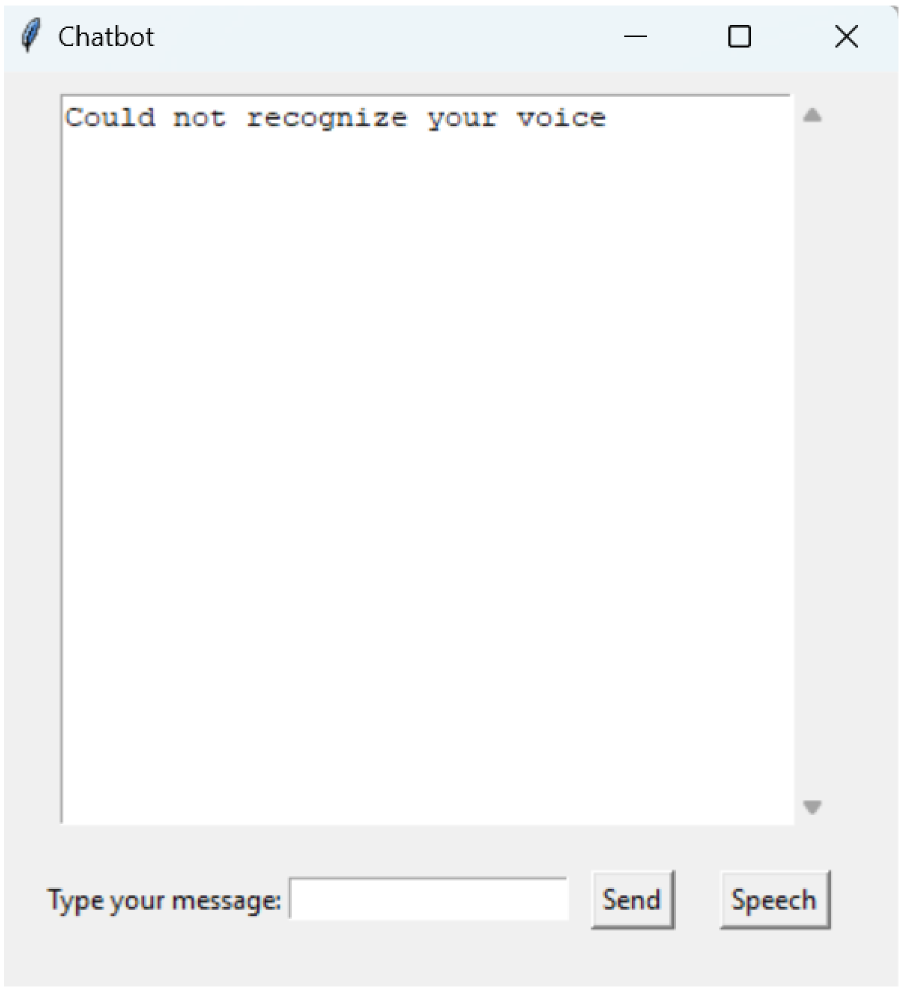

```
Could not recognize your voice
```

**Fig : Chatbot Implementation after taking invalid user response**

Tkinter is a Python library that enables developers to create desktop applications with graphical user interfaces for their Python programs. Tkinter can be used in chatbot development to create a custom graphical user interface for the chatbot, allowing users to interact with the chatbot in a visual way. The chatbot's conversational flow can be integrated into the Tkinter window using Python code to handle user input and generate responses.

Using Tkinter in chatbot development allows developers to create a more engaging and interactive user experience. Tkinter provides a range of GUI elements such as buttons, menus, text boxes, and more that can be used to create a user-friendly interface for the chatbot. Additionally, Tkinter is easy to learn and use, making it accessible to developers with limited experience in GUI programming.

## References

Here are some popular books on chatbot development:

1. "Building Chatbots with Python: Using Natural Language Processing and Machine Learning" by Sumit Pandey and Mohit Sewak (2019)
2. "Practical Artificial Intelligence with Swift: From Fundamentals to Deep Learning" by Jon Manning, Paris Buttfield-Addison, and Tim Nugent (2019)
3. "Designing Bots: Creating Conversational Experiences" by Amir Shevat (2017)
4. "Programming the Microsoft Bot Framework: A Multiplatform Approach to Building Chatbots" by Joe Mayo (2018)
5. "Chatbots: 100 Successful Business and Brand Bots on Facebook Messenger, Skype, Slack, and More" by Andrea Foresi and Roberto Pieraccini (2018)