Mimic Like Human Chatbot

Darshil Mavadiya (22BEC508)
Dept. of Electronics and Communication Engineering
Institute of Technology,Nirma University
Ahmedabd,India
22bec508@nirmauni.ac.in

Anant Kothivar (22BEC506)
Dept. of Electronics and Communication Engineering
Institute of Technology,Nirma University
Ahmedabad, India
22bec506@nirmauni.ac.in

***Abstract*—In this paper we discussed about This development of a human mimic chatbot using Python, employing machine learning paradigms, notably neural networks, and other related concepts. The study investigates the efficacy of leveraging these techniques to imbue the chatbot with human-like conversational abilities. Through meticulous experimentation and analysis, the paper elucidates the intricacies of training the chatbot to emulate human speech patterns, responses, and emotional intelligence. The findings shed light on the feasibility and challenges of creating naturalistic interactions in conversational agents. This research contributes to advancing the field of natural language processing and offers insights for future developments in chatbot technology.**

***Index Terms*—Chatbot, Machine Learning, Natural Language Processing**

I. INTRODUCTION

Our chatbot represents a computerized entity capable Of engaging in natural language conversations, mirroring human interaction patterns. Designed to assume various roles, such as answering queries, it serves as a surrogate for human engagement in tasks across multiple domains, including customer service and call centers. Powered by Artificial Intelligence Markup Language, chatbots operate as conversational agents, striving to emulate human-like intelligence. The evolution of chatbot architecture and capabilities aims to enhance their resemblance to human conversational partners, although they currently face limitations in real-time knowledge expansion and conversation tracking. Leveraging machine learning, chatbots employ artificial intelligence to decipher user queries and deliver contextually relevant responses, thus facilitating seamless human-machine interaction.

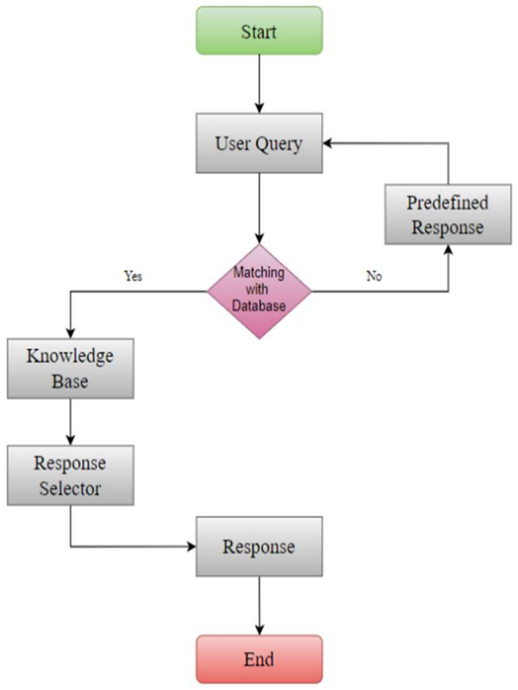
II. METHODOLOGY

The incremental build model, a software development approach, involves iterative design, implementation, and testing, with each iteration adding incremental functionality until completion. Integrating aspects of the waterfall model and prototyping philosophy, this model ensures gradual refinement and enhancement of the product. The proposed methodology for our chatbot system is outlined as follows:

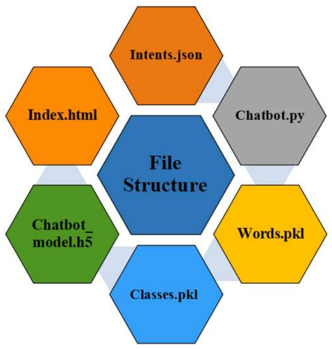
1) Initiate the process.
2) Obtain user query (input).
3) Pre-process the query, eliminating stop words like "is" and "the".
4) Extract relevant keywords from the processed query.
5) Compare keywords with those in the knowledge base using a keyword matching algorithm.
6) Provide an appropriate response based on the matched keywords.
7) Output the query response to the user.
8) Conclude the process.

This structured approach facilitates the systematic development and refinement of the chatbot system, ensuring optimal performance and user satisfaction.

III. FLOW-CHART



IV. FILE STRUCTURE

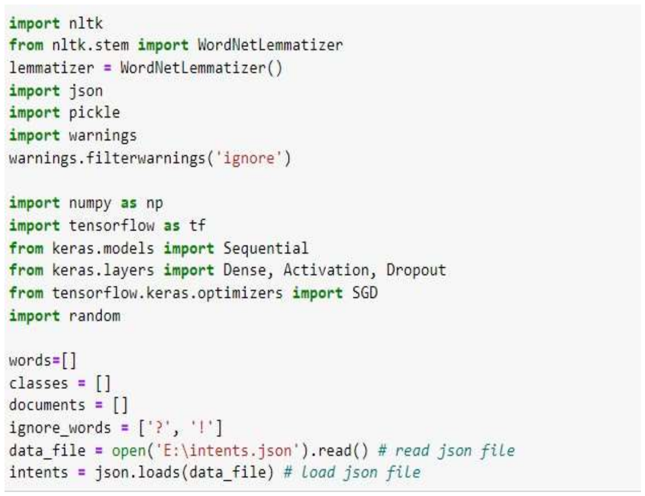


V. Implementation

A. *Import and load the data file:*

First, make a file name as train_chatbot.py. We import the necessary packages for our chatbot and initialize the variables we will use in our Python project.

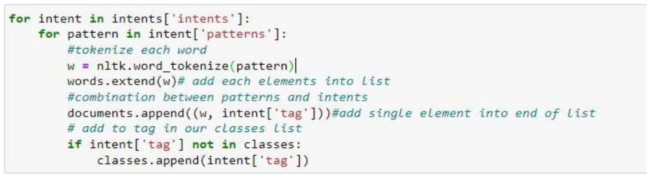
The data file is in JSON format so we used the json package to parse the JSON file into Python.



```
import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import json
import pickle
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import SGD
import random

words=[]
classes = []
documents = []
ignore_words = ['?', '!']
data_file = open('E:\intents.json').read() # read json file
intents = json.loads(data_file) # load json file
```

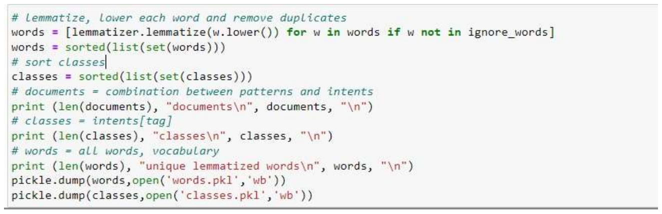
B. *Preprocess Data:*

When working with text data, we need to perform various preprocessing on the data before we make a machine learning or a deep learning model. Based on the requirements we need to apply various operations to preprocess the data.Tokenizing is the most basic and first thing you can do on text data. Tokenizing is the process of breaking the whole text into small parts like words.Here we iterate through the patterns and tokenize the sentence using nltk.word_tokenize() function and append each word in the words list. We also create a list of classes for our tags.



```
for intent in intents['intents']:
    for pattern in intent['patterns']:
        #tokenize each word
        w = nltk.word_tokenize(pattern)
        words.extend(w)# add each elements into list
        #combination between patterns and intents
        documents.append((w, intent['tag']))#add single element into end of list
        # add to tag in our classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])
```

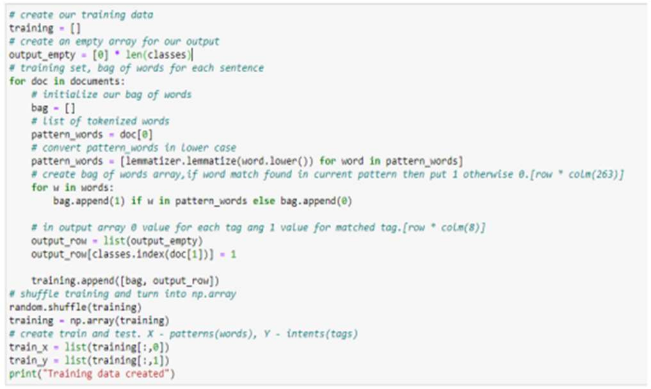
Now we will lemmatize each word and remove duplicate words from the list. Lemmatizing is the process of converting a word into its lemma form and then creating a pickle file to store the Python objects which we will use while predicting



```
# lemmatize, lower each word and remove duplicates
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]
words = sorted(list(set(words)))
# sort classes
classes = sorted(list(set(classes)))
# documents = combination between patterns and intents
print (len(documents), "documents\n", documents, "\n")
# classes = intents[tag]
print (len(classes), "classes\n", classes, "\n")
# words = all words, vocabulary
print (len(words), "unique lemmatized words\n", words, "\n")
pickle.dump(words,open('words.pkl','wb'))
pickle.dump(classes,open('classes.pkl','wb'))
```

C. *Create training and testing data:*

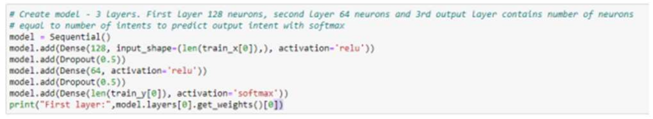
Now, we will create the training data in which we will provide the input and the output. Our input will be the pattern and output will be the class our input pattern belongs to. But the computer doesn't understand text so we will convert text into numbers.



```
# create our training data
training = []
# create an empty array for our output
output_empty = [0] * len(classes)
# training set, bag of words for each sentence
for doc in documents:
    # initialize our bag of words
    bag = []
    # list of tokenized words
    pattern_words = doc[0]
    # convert pattern_words in lower case
    pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern_words]
    # create bag of words array,if word match found in current pattern then put 1 otherwise 0.[row * colm(263)]
    for w in words:
        bag.append(1) if w in pattern_words else bag.append(0)

    # in output array 0 value for each tag ang 1 value for matched tag.[row * colm(8)]
    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1

    training.append([bag, output_row])
# shuffle training and turn into np.array
random.shuffle(training)
training = np.array(training)
# create train and test. X - patterns(words), Y - intents(tags)
train_x = list(training[:,0])
train_y = list(training[:,1])
print("Training data created")
```

D. *Build the model:*

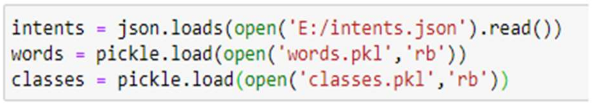
We have our training data ready, now we will build a deep neural network that has 3 layers We use the Keras sequential API for this. After training the model for 200 epochs, we achieved 100% accuracy on our model. Let us save the model as 'chatbot_model.h5'.



```
# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd output layer contains number of neurons
# equal to number of intents to predict output intent with softmax
model = Sequential()
model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(train_y[0]), activation='softmax'))
print("First layer:",model.layers[0].get_weights()[0])
```

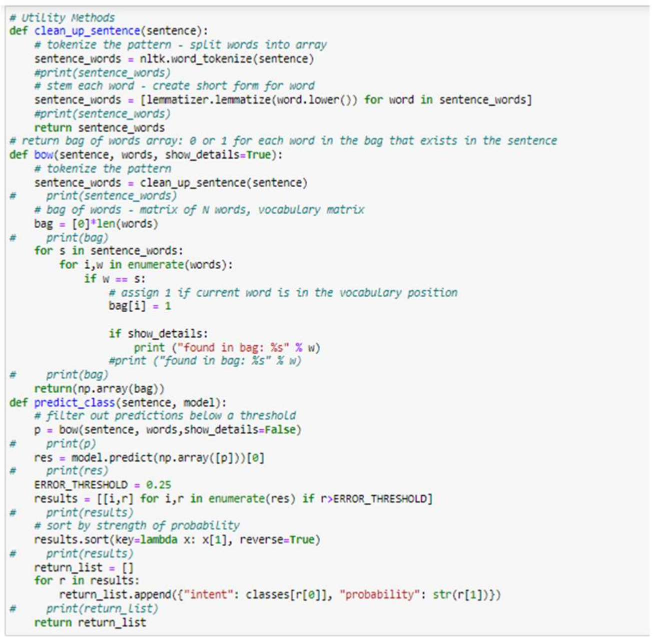
E. Predict the responce:

To predict the sentences and get a response from the user to let us create a new file ‘chatapp.py’. We will load the trained model and then use a graphical user interface that will predict the response from the bot. The model will only tell us the class it belongs to, so we will implement some functions which will identify the class and then retrieve us a random response from the list of responses. Again we import the necessary packages and load the ‘words.pkl’ and ‘classes.pkl’ pickle files which we have created when we trained our model:



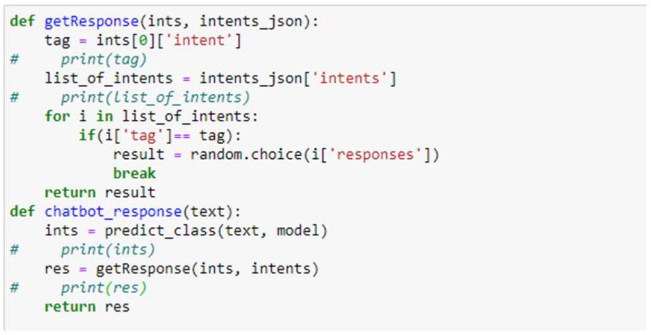
```
intents = json.loads(open('E:/intents.json').read())
words = pickle.load(open('words.pkl','rb'))
classes = pickle.load(open('classes.pkl','rb'))
```

To predict the class, we will need to provide input in the same way as we did while training. So we will create some functions that will perform text preprocessing and then predict the class.



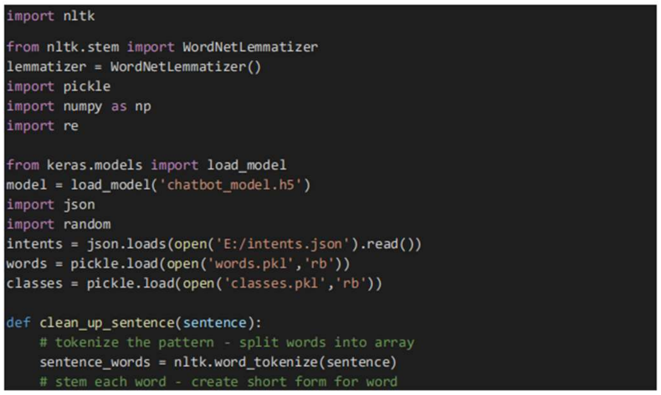
```
# Utility Methods
def clean_up_sentence(sentence):
    # tokenize the pattern - split words into array
    sentence_words = nltk.word_tokenize(sentence)
    #print(sentence_words)
    # stem each word - create short form for word
    sentence_words = [lemmatizer.lemmatize(word.lower()) for word in sentence_words]
    #print(sentence_words)
    return sentence_words
# return bag of words array: 0 or 1 for each word in the bag that exists in the sentence
def bow(sentence, words, show_details=True):
    # tokenize the pattern
    sentence_words = clean_up_sentence(sentence)
#     print(sentence_words)
    # bag of words - matrix of N words, vocabulary matrix
    bag = [0]*len(words)
#     print(bag)
    for s in sentence_words:
        for i,w in enumerate(words):
            if w == s:
                # assign 1 if current word is in the vocabulary position
                bag[i] = 1

                if show_details:
                    print ("found in bag: %s" % w)
                #print ("found in bag: %s" % w)
#     print(bag)
    return(np.array(bag))
def predict_class(sentence, model):
    # filter out predictions below a threshold
    p = bow(sentence, words,show_details=False)
#     print(p)
    res = model.predict(np.array([p]))[0]
#     print(res)
    ERROR_THRESHOLD = 0.25
    results = [[i,r] for i,r in enumerate(res) if r>ERROR_THRESHOLD]
#     print(results)
    # sort by strength of probability
    results.sort(key=lambda x: x[1], reverse=True)
#     print(results)
    return_list = []
    for r in results:
        return_list.append({"intent": classes[r[0]], "probability": str(r[1])})
#     print(return_list)
    return return_list
```

After predicting the class, we will get a random response from the list of intents:

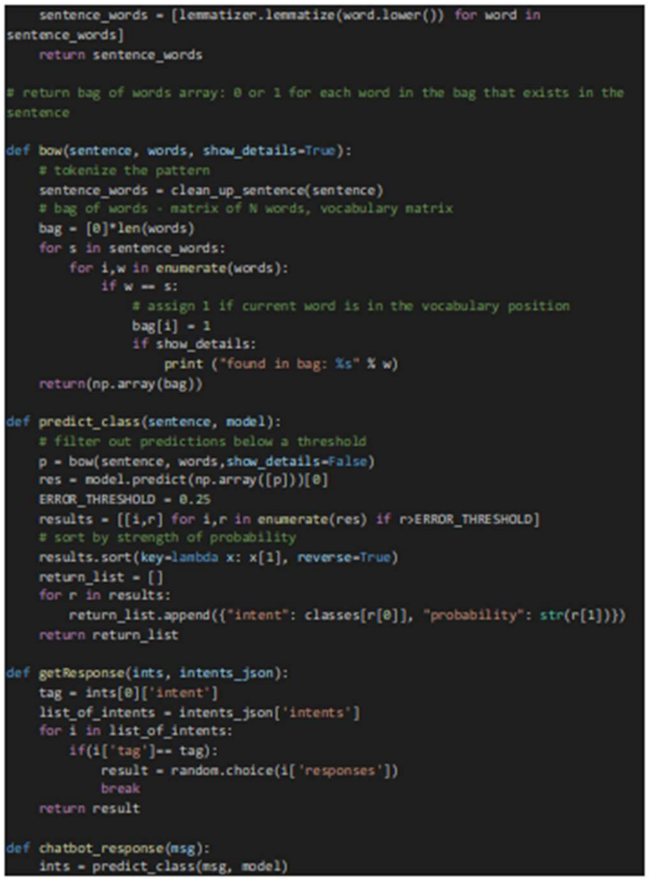


```
def getResponse(ints, intents_json):
    tag = ints[0]['intent']
#     print(tag)
    list_of_intents = intents_json['intents']
#     print(list_of_intents)
    for i in list_of_intents:
        if(i['tag']== tag):
            result = random.choice(i['responses'])
            break
    return result
def chatbot_response(text):
    ints = predict_class(text, model)
#     print(ints)
    res = getResponse(ints, intents)
#     print(res)
    return res
```

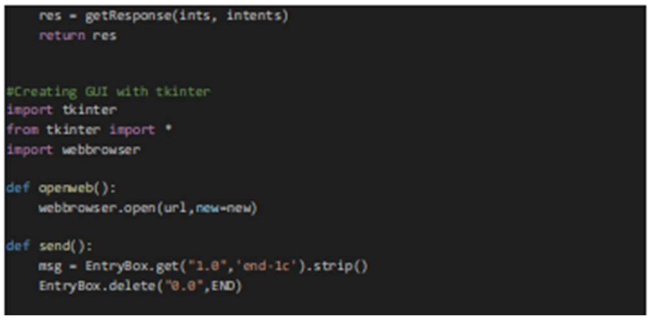
F. Here is the full source code for the GUI:



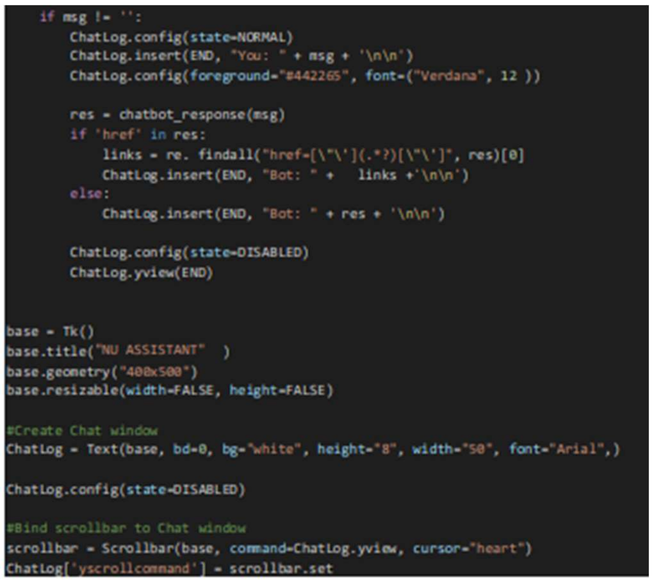
```
import nltk

from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import pickle
import numpy as np
import re

from keras.models import load_model
model = load_model('chatbot_model.h5')
import json
import random
intents = json.loads(open('E:/intents.json').read())
words = pickle.load(open('words.pkl','rb'))
classes = pickle.load(open('classes.pkl','rb'))

def clean_up_sentence(sentence):
    # tokenize the pattern - split words into array
    sentence_words = nltk.word_tokenize(sentence)
    # stem each word - create short form for word
```



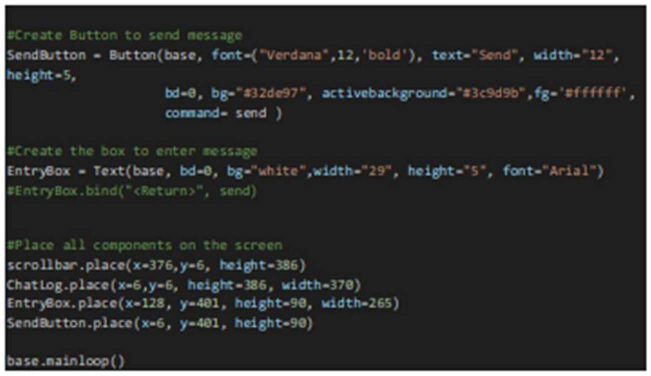
```
    sentence_words = [lemmatizer.lemmatize(word.lower()) for word in
sentence_words]
    return sentence_words

# return bag of words array: 0 or 1 for each word in the bag that exists in the
sentence

def bow(sentence, words, show_details=True):
    # tokenize the pattern
    sentence_words = clean_up_sentence(sentence)
    # bag of words - matrix of N words, vocabulary matrix
    bag = [0]*len(words)
    for s in sentence_words:
        for i,w in enumerate(words):
            if w == s:
                # assign 1 if current word is in the vocabulary position
                bag[i] = 1
                if show_details:
                    print ("found in bag: %s" % w)
    return(np.array(bag))

def predict_class(sentence, model):
    # filter out predictions below a threshold
    p = bow(sentence, words,show_details=False)
    res = model.predict(np.array([p]))[0]
    ERROR_THRESHOLD = 0.25
    results = [[i,r] for i,r in enumerate(res) if r>ERROR_THRESHOLD]
    # sort by strength of probability
    results.sort(key=lambda x: x[1], reverse=True)
    return_list = []
    for r in results:
        return_list.append({"intent": classes[r[0]], "probability": str(r[1])})
    return return_list

def getResponse(ints, intents_json):
    tag = ints[0]['intent']
    list_of_intents = intents_json['intents']
    for i in list_of_intents:
        if(i['tag']== tag):
            result = random.choice(i['responses'])
            break
    return result

def chatbot_response(msg):
    ints = predict_class(msg, model)
```



```
    res = getResponse(ints, intents)
    return res


#Creating GUI with tkinter
import tkinter
from tkinter import *
import webbrowser

def openweb():
    webbrowser.open(url,new=new)

def send():
    msg = EntryBox.get("1.0",'end-1c').strip()
    EntryBox.delete("0.0",END)
```

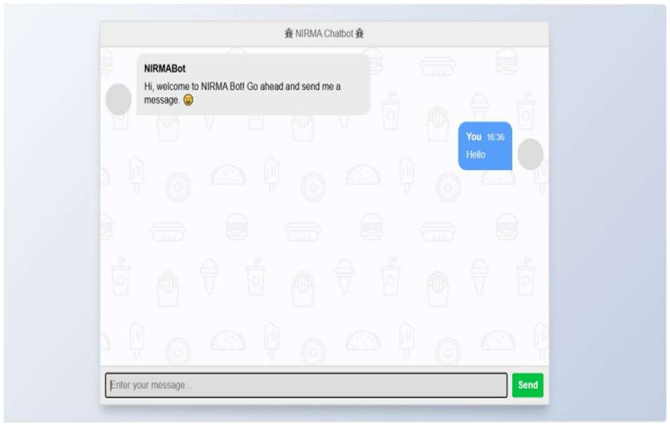


```
    if msg != '':
        ChatLog.config(state=NORMAL)
        ChatLog.insert(END, "You: " + msg + '\n\n')
        ChatLog.config(foreground="#442265", font=("Verdana", 12 ))

        res = chatbot_response(msg)
        if 'href' in res:
            links = re. findall("href=[\"\'](.*?)[\"\']", res)[0]
            ChatLog.insert(END, "Bot: " +   links +'\n\n')
        else:
            ChatLog.insert(END, "Bot: " + res + '\n\n')

        ChatLog.config(state=DISABLED)
        ChatLog.yview(END)


base = Tk()
base.title("NU ASSISTANT"  )
base.geometry("400x500")
base.resizable(width=FALSE, height=FALSE)

#Create Chat window
ChatLog = Text(base, bd=0, bg="white", height="8", width="50", font="Arial",)

ChatLog.config(state=DISABLED)

#Bind scrollbar to Chat window
scrollbar = Scrollbar(base, command=ChatLog.yview, cursor="heart")
ChatLog['yscrollcommand'] = scrollbar.set
```



```
#Create Button to send message
SendButton = Button(base, font=("Verdana",12,'bold'), text="Send", width="12",
height=5,
                    bd=0, bg="#32de97", activebackground="#3c9d9b",fg='#ffffff',
                    command= send )

#Create the box to enter message
EntryBox = Text(base, bd=0, bg="white",width="29", height="5", font="Arial")
#EntryBox.bind("<Return>", send)


#Place all components on the screen
scrollbar.place(x=376,y=6, height=386)
ChatLog.place(x=6,y=6, height=386, width=370)
EntryBox.place(x=128, y=401, height=90, width=265)
SendButton.place(x=6, y=401, height=90)

base.mainloop()
```

VI. Run the Chatbot



NIRMABot
Hi, welcome to NIRMA Bot! Go ahead and send me a message. 😄

You 16:36
Hello

VII. Conclusion

Artificial intelligence (AI) stands as one of the most rapidly advancing technologies globally. Leveraging AI alongside sophisticated databases enables significant advancements in pattern recognition and virtual assistance This project focuses on developing a chatbot system within a web-based framework, merging AI capabilities, comprehensive databases, and virtual assistance functionalities. By integrating neural network models and natural language processing (NLP) techniques in Python, we created a highly accurate deep learning-based chatbot. This chatbot offers flexibility for customization to align with specific business needs and can be trained with exceptional precision. Given the ubiquitous use of chatbots across various sectors, businesses are increasingly seeking to integrate them into their operational workflows.

Acknowledgment

This paper was performed under the guidance of proff. Rutul Patel and Proff. Bhupendra Fataniya of Electronics and Communication Department, Institute of Technology Nirma University. The authors of the paper are thankful to the university and the professors for the guidance in completion of the term paper as a part of the teaching and learning process.

References

[1] For NumPy: Indian AI production https://youtu.be/zeGTmpkdXYU

[2] For Tensorflow: Indian AI production https://youtu.be/x178e1Ykujo , Codebasics https://youtu.be/yfsTZbwgMSE

[3] For Keras: Codebasics https://youtu.be/yfsTZbwgMSE

[4] For Nltk: Sentdex https://youtu.be/FLZvOKSCkxY ,Knowledge Shelf https://youtu.be/a8E2gksDXDQ

[5] For Chatbot implementation: https://data-flair.training/blogs/python-chatbot-project/

[6] For Chatbot GUI: https://youtu.be/a37BL0stIuM