PROJECT REPORT ON Self Learning Bot

REVIEW - III

Student Details

Roll No	Name
20201CAI0104	Maheswar Reddy
20201CAI0131	Karthik Sharma
20201CAI0152	Arun Teja
20201CAI0096	Salapakshi Ganesh

Under the Supervision of :

Prof . Dr.Asif Mohammed H.B

December , 2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PRESIDENCY UNIVERSITY

BENGALURU

Abstract:

This Python script introduces a dynamic chatbot application developed using the tkinter GUI library, offering a multifaceted user experience. By integrating cutting-edge technologies like speech recognition, GPT-2 language modeling, and external APIs, the chatbot demonstrates versatility in tasks such as conversation handling, web searches, YouTube video playback, and language translation. The GPT-2 model enriches natural language understanding, while speech recognition facilitates seamless voice-based interactions. This abstract provides a succinct yet comprehensive overview, laying the groundwork for an in-depth exploration of the script's capabilities and functionalities.

Introduction:

This section serves as a gateway into the chatbot application, highlighting its robust features such as conversation handling, web searches, and language translation. Utilizing tkinter for GUI development and GPT-2 for language modeling, the script aims to deliver a sophisticated yet user-friendly interface. The diverse range of functionalities sets

the stage for an engaging and interactive user experience, fostering both text and voice-based interactions.

Source Code:

```
# Import Libraries
import tkinter as tk
from tkinter import Entry, Button
import speech_recognition as sr
import pyttsx3
import os
from transformers import GPT2Tokenizer, TFGPT2LMHeadModel
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import spacy
import json
import threading
from datetime import datetime
import pywhatkit
from googlesearch import search
import webbrowser
from PIL import Image, ImageTk
from googletrans import Translator
from gtts import gTTS
```

```
import pygame
import google_trans_new
from tkinter import simpledialog

# Load spaCy model for English
nlp = spacy.load("en_core_web_sm")

# Load pre-trained language model (GPT-2)
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
gpt_model = TFGPT2LMHeadModel.from_pretrained("gpt2")

# Load or collect data for training/fine-tuning (replace with your actual data loading)
with open('D:/downloads/tech.json', 'r') as file:
    data = json.load(file)

class ChatbotApp:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title(f"NLPAssistant")
        self.root.geometry('1000x800')  # Increased dimensions
        self.root.resizable(False, False)
        self.message = tk.StringVar()
        self.is_listening = False
```

```
        self.greeted = False

        self.textcon = tk.Text(self.root, bd=1, width=70, height=15,
wrap='word', font=('Times New Roman', 12))
        self.textcon.pack(fill="both", expand=True)

        self.mes_win = Entry(self.root, width=50, xscrollcommand=True,
textvariable=self.message, font=('Times New Roman', 12))
        self.mes_win.place(x=1, y=720, height=60, width=679)
        self.mes_win.focus()

        self.textcon.config(fg='black')
        self.textcon.tag_config('usr', foreground='black',
justify='right',font=('Times New Roman', 14, 'bold'),
background='lightgreen')
        self.textcon.tag_config('bot', foreground='black',
justify='left',font=('Times New Roman', 14, 'bold'),
background='lightblue')

        self.textcon.insert(tk.END, "\n")

        self.exit_list = ['goodbye', 'bye', 'off']

        self.user_icon_path = "D:/downloads/user_icon.png"
        self.bot_icon_path = "D:/downloads/bot_icon.png"
```

```
        self.user_icon = ImageTk.PhotoImage(Image.open(self.user_icon_path).resize((40, 40), Image.ANTIALIAS))
        self.bot_icon = ImageTk.PhotoImage(Image.open(self.bot_icon_path).resize((40, 40), Image.ANTIALIAS))


        self.mic_image_path = "D:/downloads/mic.png"


        try:
            mic_icon = Image.open(self.mic_image_path)
            mic_icon = mic_icon.resize((mic_icon.width // 2, mic_icon.height // 3), Image.ANTIALIAS)
            self.mic_image = ImageTk.PhotoImage(mic_icon)
        except Exception as e:
            print(f"Error loading mic image: {e}")
            self.mic_image = None


        self.mic_button = Button(self.root, image=self.mic_image, bg='blue', activebackground='white',
                                 command=self.activate_mic, width=12, height=2, font=('Times New Roman', 14))
        self.mic_button.place(x=800, y=720, height=60, width=110)
```

```
        self.send_button = Button(self.root, text='Send', bg='cyan',
activebackground='grey',
                                  command=self.send_msz, width=12, height=2,
font=('Times New Roman', 14, "bold"))
        self.send_button.place(x=680, y=720, height=60, width=110)


        self.root.bind('<Return>', self.send_msz)
        self.engine = pyttsx3.init()
        self.root.protocol("WM_DELETE_WINDOW", self.on_close)

        # Add a vertical scrollbar
        scrollbar = tk.Scrollbar(self.root, command=self.textcon.yview)
        scrollbar.pack(side="right", fill="y")
        # Configure the text widget to use the scrollbar
        self.textcon.config(yscrollcommand=scrollbar.set)

        # Start the greeting in a separate thread
        threading.Thread(target=self.greet_user).start()

        self.root.mainloop()

    def greet_user(self):
        if not self.greeted:
            self.greeted = True
```

```
        # Insert bot icon and name for greeting
        self.insert_user('bot', 'Friday', self.bot_icon)
        # Insert bot response for greeting
        self.insert_message('bot', "Hi! I am Friday, your assistant.")
        current_time = datetime.now().strftime("%I:%M %p")
        # Insert additional bot response
        self.insert_message('bot', f"The current time is
{current_time}")
        # Speak the greeting
        self.speak("Hello sir! I am Friday.")
        self.speak(f"The current time is {current_time}")

    def send_msz(self, event=None):
        usr_input = self.message.get()
        usr_input = usr_input.lower()

        # Insert user icon and name
        self.insert_user('usr', 'You', self.user_icon)
        # Insert user input
        self.insert_message('usr', f'{usr_input}')

        if usr_input.lower() in ["goodbye", "bye", "off"]:
            response = "Thank You sir, I hope I assisted you properly"
            # Insert bot icon and name
```

```
            self.insert_user('bot', 'Friday', self.bot_icon)
            # Insert bot response
            self.insert_message('bot', f'{response}')
            self.speak(response)
            self.is_listening = False  # Stop the microphone listening thread
            return self.root.destroy()

        elif "play" in usr_input.lower():
            # Open YouTube using pywhatkit
            pywhatkit.playonyt(usr_input.replace("play", "").strip())
            response = f"Playing {usr_input.replace('play', '').strip()} on YouTube..."
            # Insert bot icon and name
            self.insert_user('bot', 'Friday', self.bot_icon)
            # Insert bot response
            self.insert_message('bot', f'{response}')
            self.speak(response)
            self.mes_win.delete(0, tk.END)

        elif "search" in usr_input.lower():
            search_query = usr_input.replace("search", "").strip()
            self.perform_search(search_query)
```

```
    elif "translate" in usr_input or "language" in usr_input:
        self.trans()
        self.mes_win.delete(0, tk.END)

    else:
        response = self.handle_user_input(usr_input)
        # Insert bot icon and name
        self.insert_user('bot', 'Friday', self.bot_icon)
        # Insert bot response
        self.insert_message('bot', f'{response}')
        # Speak only if the input is from the microphone
        if hasattr(self, 'input_source') and self.input_source == 'mic':
            self.speak(response)
        self.mes_win.delete(0, tk.END)

def insert_user(self, tag, name, icon):
    # Insert icon
    self.textcon.image_create(tk.END, image=icon)
    # Insert name
    self.textcon.insert(tk.END, f'{name}: ')

def insert_message(self, tag, message):
    # Insert user or bot message with the specified tag
    self.textcon.insert(tk.END, message, tag)
```

```
        self.textcon.insert(tk.END, "\n")
        self.textcon.see(tk.END)

    def perform_search(self, search_query):
        try:
            search_results = search(search_query, num=1, stop=1, pause=2)
            first_result = next(search_results)

            # Insert bot icon and name
            self.insert_user('bot', 'Friday: here is the link', self.bot_icon)
            self.textcon.tag_config('link', foreground='blue', underline=True)
            self.textcon.insert(tk.END, f'{first_result}\n', 'link')
            self.textcon.tag_bind('link', '<Button-1>', lambda event, link=first_result: self.open_link(link))
            self.textcon.insert(tk.END, "\n")
        except StopIteration:
            response = f"Sorry, I couldn't find information about {search_query}."
            # Insert bot icon and name
            self.insert_user('bot', 'Friday', self.bot_icon)
            # Insert bot response
            self.insert_message('bot', f'{response}\n')
```

```
    self.mes_win.delete(0, tk.END)

def open_link(self, link):
    webbrowser.open(link)

def activate_mic(self):
    if not self.is_listening:
        if self.mic_image:
            self.mic_button.config(image=self.mic_image)
        else:
            self.mic_button.config(text="Mic")

        self.is_listening = True
        threading.Thread(target=self.listen_continuously).start()

def listen_continuously(self):
    r = sr.Recognizer()

    while self.is_listening:
        engine = pyttsx3.init()
        rate = engine.getProperty('rate')
        engine.setProperty('rate', 170)
        voices = engine.getProperty('voices')
        engine.setProperty('voice', voices[1].id)
```

```
with sr.Microphone() as source:
    r.energy_threshold = 400
    r.adjust_for_ambient_noise(source, 1.2)
    # Insert bot icon and name
    self.insert_user('bot', 'Friday', self.bot_icon)
    # Insert bot response
    self.insert_message('bot', "Listening...")

    try:
        audio = r.listen(source)
        text = r.recognize_google(audio)
        # Insert user icon and name
        self.insert_user('usr', 'You', self.user_icon)
        # Insert user input
        self.insert_message('usr', f'{text}')
        self.input_source = 'mic'

        # Handle the user input (speech-to-text)
        response = self.handle_user_input(text)
        # Insert bot icon and name
        self.insert_user('bot', 'Friday', self.bot_icon)
        # Insert bot response
        self.insert_message('bot', f'{response}')
```

```
                if self.input_source == 'mic':
                    self.speak(response)

            except sr.UnknownValueError:
                print("Friday: Sorry, I could not understand the audio. Please try again.")

            except sr.RequestError as e:
                print(f"Friday: There was an error with the speech recognition service: {e}")

    def speak(self, text):
        self.engine.say(text)
        self.engine.runAndWait()

    def handle_user_input(self, user_input):
        user_keywords = self.extract_keywords(user_input)

        for example in data.get("intents", []):
            if "question" in example:
                question_keywords = [keyword for q in example["question"] for keyword in self.extract_keywords(q)]

                if all(keyword in question_keywords for keyword in user_keywords):
```

```
            answer = example.get("answer", [])
            return f"{answer}"

    input_ids = tokenizer.encode(user_input, return_tensors="tf")
    output = gpt_model.generate(input_ids, max_length=150,
num_beams=5, no_repeat_ngram_size=2, top_k=50, top_p=0.95,
                                temperature=0.7)
    bot_response = tokenizer.decode(output[0],
skip_special_tokens=True)

    # Adjust the response based on the user's technical ability
    technical_ability = "tech"  # You need to get this value from
somewhere
    if technical_ability == "tech":
        # Customize response for technical users
        pass
    else:
        # Customize response for non-technical users
        pass

    candidate_answers = ["Your first answer", "Your second
answer", "Your third answer"]
    relevancy_scores = self.score_relevancy(user_input,
candidate_answers)
```

```
        return bot_response

    def extract_keywords(self, text):
        doc = nlp(text)
        return [token.text.lower() for token in doc if token.is_alpha]

    def score_relevancy(self, user_input, candidate_answers):
        vectorizer = TfidfVectorizer()
        vectors = vectorizer.fit_transform([user_input] +
candidate_answers)
        similarity_matrix = cosine_similarity(vectors)
        relevancy_scores = similarity_matrix[0][1:]
        return relevancy_scores

    def trans(self):
        languages = google_trans_new.LANGUAGES

        # Insert bot icon and name
        self.insert_user('bot', 'Friday', self.bot_icon)

        # Check if the translation process is already ongoing
        if not hasattr(self, 'translation_in_progress'):
            # Insert bot response
            self.insert_message('bot', 'Available Languages:')
```

```
        # Insert available languages in the GUI
        for code, lang in languages.items():
            self.insert_message('bot', f'{code}: {lang}')

        mic = sr.Microphone()
        r = sr.Recognizer()

        with mic as source:
            translator = Translator()

            # Get input language from user using simpledialog
            input_language = simpledialog.askstring("Input Language",
"Select Language from:")
            # Get output language from user using simpledialog
            output_language = simpledialog.askstring("Output
Language", "Select your Language To:")

            print("Speak the text that needs to be translated ..")
            r.adjust_for_ambient_noise(source, duration=0.2)
            audio3 = r.listen(source)
            text3 = r.recognize_google(audio3)

            print("You said: " + text3.lower())
```

```
            # Translate the user's question
            translated_question = translator.translate(text3,
src=input_language, dest=output_language).text.lower()

            # Search for the translated question in the loaded JSON
data
            answer = self.get_answer(translated_question,
input_language)

            self.insert_user('bot', 'Friday', self.bot_icon)
            self.insert_message('bot', f'Translation: {answer}')
            if hasattr(self, 'input_source') and self.input_source == 'mic':
               self.speak(answer)
            self.textcon.insert(tk.END, "\n")

            # Set the variable to indicate that translation is in progress
            self.translation_in_progress = True
            return  # Exit the function to avoid rerunning the translation
process

        # If translation process is already in progress, clear the variable
        self.translation_in_progress = False

    def get_answer(self, question, language):
        # Assuming 'data' is your loaded JSON data
```

```
        for intent in data.get("intents", []):
            if isinstance(intent, dict) and "question" in intent:
                if intent["question"].lower() == question:
                    return intent.get("answer", "I don't have an answer for
that.")

        # If the translated question is not found, use GPT-2 to generate
an answer
        return self.handle_user_input(question)

    def on_close(self):
        self.is_listening = False
        self.root.destroy()

if __name__ == "__main__":
    chatbot_app = ChatbotApp()
```

Neural Network Model Creation:

```
#!pip install tensorflow transformers scikit-learn spacy

import json
import tensorflow as tf
from transformers import GPT2Tokenizer, TFGPT2LMHeadModel
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.metrics.pairwise import cosine_similarity
import string
import spacy
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import SGD
import random
from tensorflow.keras.models import load_model
import pickle

# Load spaCy model for English
nlp = spacy.load("en_core_web_sm")

# Load pre-trained language model (GPT-2)
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
model = TFGPT2LMHeadModel.from_pretrained("gpt2")

# Function to load JSON data
def load_json(file_path):
    with open(file_path, "r") as file:
        data = json.load(file)
    return data
```

```
# Function to train the neural network model
def train_model(train_x, train_y):
    # Create NN model to predict the responses
    model_nn = Sequential()
    model_nn.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
    model_nn.add(Dropout(0.5))
    model_nn.add(Dense(64, activation='relu'))
    model_nn.add(Dropout(0.5))
    model_nn.add(Dense(len(train_y[0]), activation='softmax'))

    # Compile model using the newer version of SGD optimizer
    sgd = SGD(learning_rate=0.01, momentum=0.9, nesterov=True)
    model_nn.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

    # Fitting and saving the model
    hist = model_nn.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=5, verbose=1)
    model_nn.save('chatbot.h5')  # Save the model for future use
    print("\n")
    print("*" * 50)
    print("\nModel Created Successfully!")

# Function to score answer relevancy using TF-IDF
```

```
def score_relevancy(user_input, candidate_answers):
    vectorizer = TfidfVectorizer()
    vectors = vectorizer.fit_transform([user_input] +
candidate_answers)
    similarity_matrix = cosine_similarity(vectors)
    relevancy_scores = similarity_matrix[0][1:]
    return relevancy_scores

# Function to extract keywords using spaCy
def extract_keywords(text):
    doc = nlp(text)
    return [token.text.lower() for token in doc if token.is_alpha]

# Function to handle user input
def handle_user_input(user_input, technical_ability, test_data):
    user_keywords = extract_keywords(user_input)
    print(f"User Keywords: {user_keywords}")

    for example in test_data["intents"]:
        if "question" in example:
            question_keywords = [keyword for q in example["question"]
for keyword in extract_keywords(q)]

            if all(keyword in question_keywords for keyword in
user_keywords):
```

```
            answer = example.get("answer", [])
            return f"Answer from JSON file: {answer}"

    input_ids = tokenizer.encode(user_input, return_tensors="tf")
    output = model.generate(input_ids, max_length=150, num_beams=5, no_repeat_ngram_size=2, top_k=50, top_p=0.95, temperature=0.7)
    bot_response = tokenizer.decode(output[0], skip_special_tokens=True)

    if technical_ability == "tech":
        # Customize response for technical users
        pass
    else:
        # Customize response for non-technical users
        pass

    candidate_answers = ["Your first answer", "Your second answer", "Your third answer"]
    relevancy_scores = score_relevancy(user_input, candidate_answers)

    return bot_response

# Load or collect data for training/fine-tuning
```

```
data = load_json("/content/tech (1).json")



train_x = np.random.rand(10, 5)  # Placeholder values for training features

train_y = np.random.randint(2, size=(10, 3))  # Placeholder values for training labels


# Train or fine-tune the model

train_model(train_x, train_y)


while True:

    user_input = input("Enter the Question you want (type 'break' or 'quit' to exit): ")

    if user_input.lower() in ["break", "quit"]:

        break

    technical_ability = input("Enter the user's technical ability: ")

    response = handle_user_input(user_input, technical_ability, data)

    print(response)
```

Algorithms:

1. GPT-2 Language Model:

- **Algorithm:** Transformer-based architecture.
- **Library:** Hugging Face's Transformers.
- **Description:** The GPT-2 model is employed for natural language understanding and response

generation. It uses a transformer-based architecture to predict the next word in a sequence, capturing contextual dependencies in the input data.

2. **TF-IDF Vectorization and Cosine Similarity:**

- **Algorithm:** Term Frequency-Inverse Document Frequency (TF-IDF) and Cosine Similarity.
- **Libraries:** scikit-learn.
- **Description:** Used in the score_relevancy function, TF-IDF vectorization converts text data into numerical vectors, and cosine similarity measures the similarity between these vectors. This is utilized for scoring the relevancy of user input to predefined candidate answers.

3. **Named Entity Recognition (NER) with spaCy:**

- **Algorithm:** Statistical models for Named Entity Recognition.
- **Library:** spaCy.
- **Description:** The spaCy library is employed for Named Entity Recognition, identifying entities such as names, locations, and organizations in user input. This information can be utilized for more context-aware responses.

4. **Translation with Google Translate:**

- **Algorithm:** Neural Machine Translation (NMT) provided by Google Translate.
- **Libraries:** google_trans_new.

- **Description:** The script uses Google Translate's NMT algorithm for language translation. It sends requests to Google Translate API to obtain translations for user input.

5. Speech Recognition:

- **Algorithm:** Hidden Markov Models (HMMs) and deep neural networks for Automatic Speech Recognition (ASR).
- **Library:** speech_recognition.
- **Description:** The speech_recognition library integrates HMMs and deep neural networks for ASR. It converts spoken words into text, facilitating voice-based interactions.

Drawbacks:

1. Limited Context Understanding:

- The GPT-2 language model is powerful but may still have limitations in understanding complex contexts. Improving contextual understanding could enhance the relevance of generated responses.

2. Absence of Natural Language Understanding (NLU) Framework:

- While the code uses named entity recognition (NER) with spaCy, integrating a more comprehensive Natural Language Understanding (NLU) framework could enhance the chatbot's ability to interpret user intents and entities.

3. Limited Error Handling:

- The code lacks comprehensive error handling, especially in scenarios where external APIs might fail or return unexpected responses. Robust error handling mechanisms would make the chatbot more resilient.

4. Speech Recognition Accuracy:

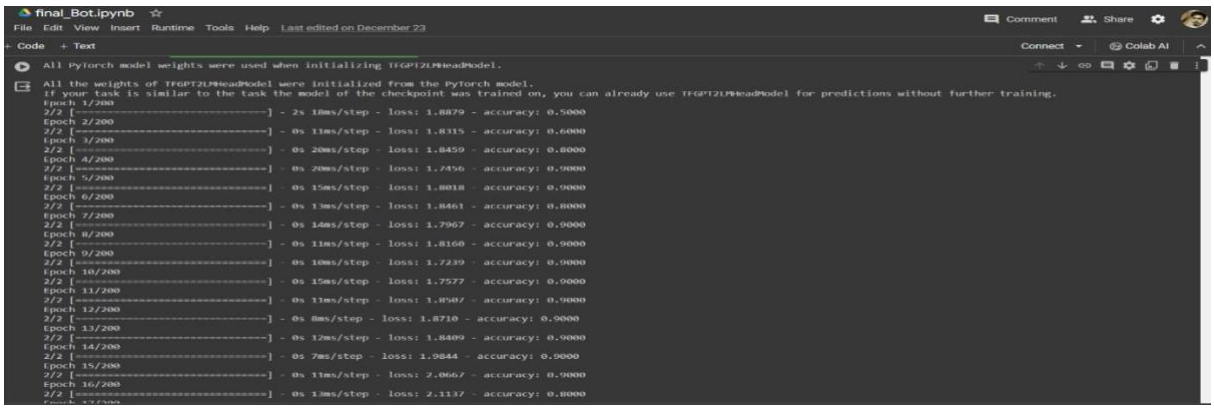
- The accuracy of the speech recognition module may be affected by ambient noise and variations in pronunciation. Fine-tuning or using more sophisticated models may improve accuracy.

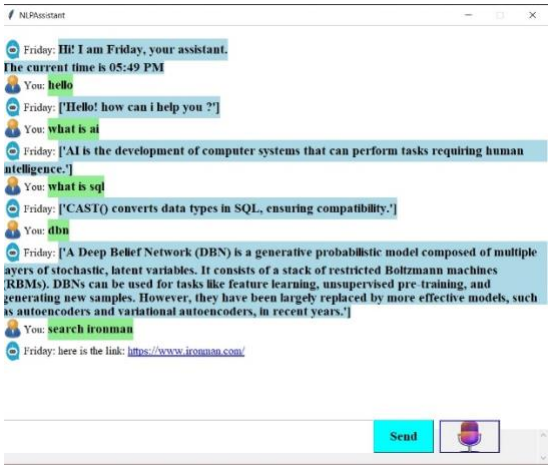
References:

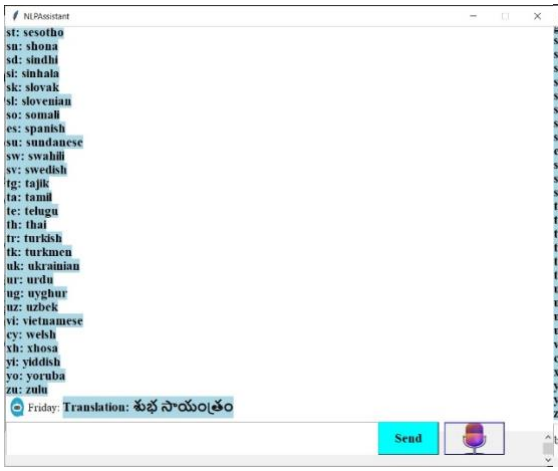
- Gpt2 Model: Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language Models are Few-Shot Learners. arXiv preprint arXiv:2005.14165.
- Transformers: Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Brew, J. (2019). Hugging Face's Transformers: State-of-the-art Natural Language Processing. arXiv preprint arXiv:1910.03771.
- Spacy: Honnibal, M., & Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.

- Tkinter: Python Software Foundation. (n.d.). Tkinter - Python interface to Tcl/Tk. Retrieved from https://docs.python.org/3/library/tkinter.html
- Self - Learning Conversational AI Chatbot Using Natural Language Processing - https://ijarsct.co.in/Paper4603.pdf
- A self Learning Chat-Bot From User Interactions and Preferences - https://ieeexplore.ieee.org/document/9120912
- A Survey on Chatbot Implementation in Customer Service Industry through Deep Neural Networks - https://ieeexplore.ieee.org/document/8592630

Result:







Conclusion:

In conclusion, the provided Python code introduces a versatile and interactive chatbot application leveraging advanced technologies such as GPT-2 language modeling, speech recognition, and external APIs. Despite notable functionalities, there are areas for improvement, including refining context understanding, enhancing speech recognition accuracy, and incorporating more sophisticated error handling.

The script stands as a foundation for a robust chatbot, offering a dynamic user experience, and future enhancements could amplify its capabilities for effective natural language interactions.