CREATE A CHATBOT IN PYTHON

Team Leader:

962821106033 : JERRIK PRAYWIN RAJ J

Team Members:

962821106004 : ABINANDH R S

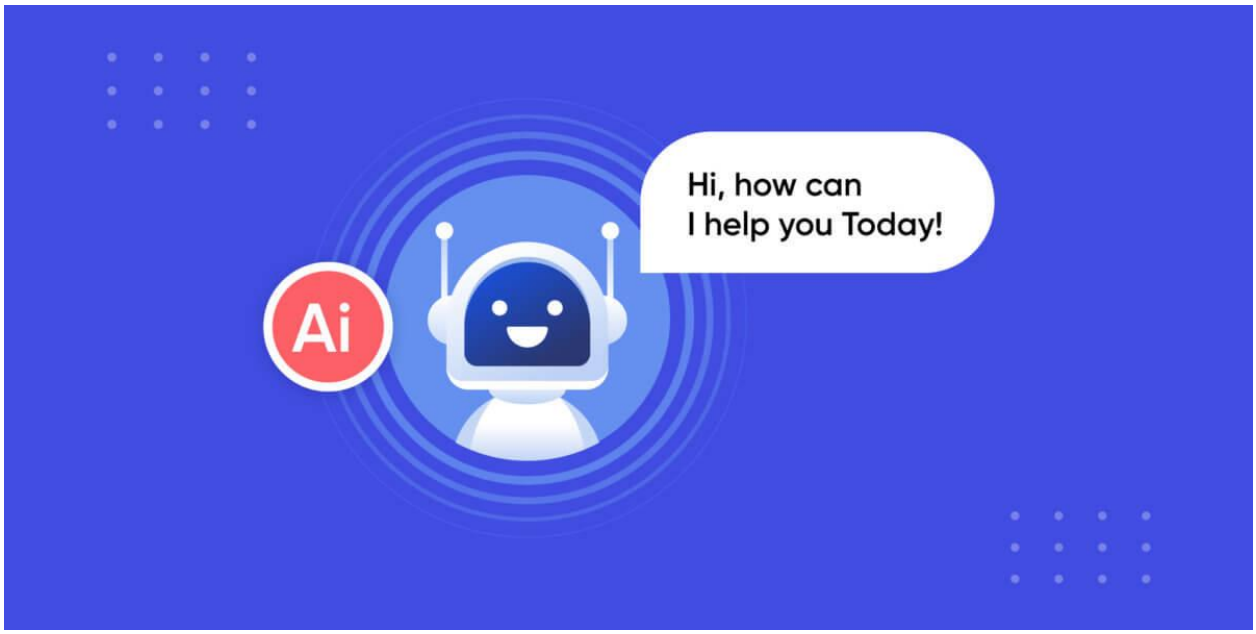
962821106014 : ARSHATH SHARIEF S

962821106017 : BALASUBRAMANIAN K

962821106018 : BASKARAN K

Phase - 5: Project Documentation & Submission

TOPIC: CREATE A CHATBOT



INTRODUCTION:

Chatbots are used in various applications across industries, including customer support, e-commerce, healthcare, and education. They are employed to streamline processes, provide instant responses, and enhance user experiences. Chatbots can be

simple rule-based systems that follow predefined scripts, or more advanced AI-driven models that learn and adapt from user interactions, becoming more proficient over time.

OBJECTIVE:

The objective of this project is to create a chatbot using python that provides exceptional customer service, answering user queries on a website or application. The objective is to deliver high-quality support to users, ensuring a positive user experience and customer satisfaction.

PHASES OF DEVELOPMENT:

PHASE-1: *Problem Definition and Design Thinking*

In this phase, we discussed and understand about the given problem statement and also discussed about the design.

PHASE-2: *Innovation*

In this phase, we considered few innovation ideas to implement on our chatbot.

PHASE-3: *Development Part 1*

In this phase, we began building our project by loading and preprocessing the dataset.

PHASE-4: *Development Part 2*

In this phase, we continued building the project by performing different activities like feature engineering, model training, evaluation etc as per the instructions in the project. And integrated our integrated our project with website using Flask.

DESIGN THINKING:

Design thinking is a problem-solving approach that's centered around understanding the needs of the people who will be using the product, service, or system being developed. It's a human-centric, iterative process that encourages creativity and innovation to come up with solutions.

Design thinking can be effectively applied to the creation and improvement of chatbots. Here's a breakdown of the design thinking process tailored to chatbot development:

1. Empathize:

Understand User Needs: Gather insights into the target audience's preferences, pain points, and behavior regarding the chatbot's intended purpose.

Analyse Existing Conversations: Review past chat logs and feedback to comprehend user interactions and common queries.

2. Define:

Identify Chatbot Goals: Define specific objectives the chatbot should achieve (e.g., customer support, information delivery, sales assistance).

Compile User Personas: Create user personas representing different user groups, their preferences, and needs.

3. Ideate:

Brainstorm Conversational Flows: Generate a variety of conversation flow ideas that address different user needs and scenarios.

Design Chatbot Features: Think of innovative features or functionalities the chatbot could offer, such as interactive elements or personalized responses.

4. Prototype:

Create Chatbot Mockups: Develop low-fidelity prototypes or wireframes of the chatbot's interface and conversation structure.

Develop Conversational Scripts: Write sample dialogues and responses for the chatbot to test its interactions.

5. Test:

User Testing: Conduct usability tests with real users to assess how well the chatbot meets their needs.

Iterate and Refine: Gather feedback on user experiences, and refine the chatbot's design, conversation flow, and features based on the test results.

6. Implement:

Develop the Chatbot: Build the chatbot using the insights and refined design from the testing phase.

Deploy and Monitor: Launch the chatbot and continuously monitor its performance, user feedback, and engagement.

LIBRARIES USED:

Some basic libraries used in our chatbot,

- TensorFlow:

  TensorFlow is an open-source framework by Google for building and training machine learning models using computational graphs. It offers flexibility, scalability, and various abstraction levels, allowing for the development and deployment of complex models across multiple platforms.

  TensorFlow in chatbots can be used to implement natural language processing (NLP) models, enabling the chatbot to understand and generate responses based on user input. It helps in training models for language understanding, sentiment analysis, intent recognition, and dialogue generation, enhancing the chatbot's conversational abilities.

- NumPy:

  NumPy is a powerful Python library for numerical computing that provides support for arrays, matrices, and a wide range of mathematical functions. It's widely used for handling numerical data efficiently and is fundamental for scientific and mathematical operations in Python.

  In a chatbot, NumPy can be used for efficient handling and manipulation of numerical data. While not directly for the chatbot's conversational abilities, NumPy can support various data processing tasks, such as statistical analysis, handling user data, or managing internal numerical computations within the chatbot's backend.

- Seaborn:

  Seaborn is a data visualization library built on top of Matplotlib in Python. It provides an easy-to-use interface for creating attractive and informative statistical graphics. Seaborn simplifies the process of creating aesthetically pleasing visualizations, particularly for statistical data exploration and understanding.

Seaborn, a data visualization library in Python, isn't directly used within a chatbot's functionalities. However, it could be utilized for data analysis and visualization related to a chatbot's performance or user interaction statistics. For instance, if the chatbot collects data that needs visual representation, Seaborn can be employed to create plots or charts to understand user engagement, conversation trends, or other relevant metrics.

- Pandas:

Pandas is a powerful Python library widely used for data manipulation and analysis. It provides data structures and tools for handling structured data, mainly in tabular form, such as CSV files, SQL databases, Excel sheets, or other data sources.

In a chatbot, Pandas can be employed for data management and analysis tasks related to user interactions. It helps in organizing, processing, and analyzing data collected from chatbot conversations, enabling insights into user behavior, sentiment, or conversation patterns for chatbot performance improvement and reporting.

- Matplotlib:

Matplotlib is a popular Python library used for creating high-quality static, interactive, and publication-ready visualizations.

In a chatbot context, matplotlib might not be directly used in the chatbot's functionality. However, it could be employed for data visualization related to the chatbot's performance metrics, user interaction statistics, or any collected data requiring graphical representation. For example, matplotlib can be utilized to create plots or charts illustrating trends in user engagement, conversation volumes, sentiment analysis, or other relevant metrics for analysis and reporting.

Example,

```
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.layers import TextVectorization
import re,string
from tensorflow.keras.layers import LSTM,Dense,Embedding,Dropout,LayerNormalization
```

Natural Language Processing (NLP): Natural Language Processing (NLP) is a subfield of artificial intelligence (AI). It helps machines process and understand the human language so that they can automatically perform repetitive tasks.

The basic text processing in NLP are:

1. Sentence Segmentation
2. Normalization
3. Tokenization

1.Sentence Segmentation:

The process of deciding from where the sentences actually start or end in NLP or we can simply say that here we are dividing a paragraph based on sentences. This process is known as Sentence Segmentation.

Program:

formatting data to be in a question answer format

```
#reading data
data=open('/kaggle/input/simple-dialogs-for-chatbot/dialogs.txt','r').read()

#paried list of question and corresponding answer
QA_list=[QA.split('\t') for QA in data.split('\n')]
print(QA_list[:5])
```

```
[['hi, how are you doing?', "i'm fine. how about yourself?"], ["i'm fine. how about yourself?", "i'm pr
etty good. thanks for asking."], ["i'm pretty good. thanks for asking.", 'no problem. so how have yo
u been?'], ['no problem. so how have you been?', "i've been great. what about you?"], ["i've been
great. what about you?", "i've been good. i'm in school right now."]]
```

```
questions=[row[0] for row in QA_list]
answers=[row[1] for row in QA_list]
```

```
print(questions[0:5])
print(answers[0:5])
```

```
['hi, how are you doing?', "i'm fine. how about yourself?", "i'm pretty good. thanks for asking.", 'no
problem. so how have you been?', "i've been great. what about you?"]
["i'm fine. how about yourself?", "i'm pretty good. thanks for asking.", 'no problem. so how have y
ou been?', "i've been great. what about you?", "i've been good. i'm in school right now."]
```

2.Normalization:

Text normalization is a pre-processing step aimed at improving the quality of the text and making it suitable for machines to process. Four main steps in text normalization are case normalization, tokenization and stop word removal, Parts-of-Speech (POS) tagging, and stemming.

Program:

To reduce its randomness, bringing it closer to a predefined “standard”

```
def remove_diacritic(text):
    return ''.join(char for char in unicodedata.normalize('NFD',text)
                   if unicodedata.category(char) !='Mn')
```

```
def preprocessing(text):

    #Case folding and removing extra whitespaces
    text=remove_diacritic(text.lower().strip())

    #Ensuring punctuation marks to be treated as tokens
    text=re.sub(r"([?.!,¿])", r" \1 ", text)

    #Removing redundant spaces
    text= re.sub(r'[" "]+', " ", text)

    #Removing non alphabetic characters
    text=re.sub(r"[^a-zA-Z?.!,¿]+", " ", text)

    text=text.strip()

    #Indicating the start and end of each sentence
    text='<start> ' + text + ' <end>'
```

```
    return text

preprocessed_questions=[preprocessing(sen) for sen in questions]
preprocessed_answers=[preprocessing(sen) for sen in answers]

print(preprocessed_questions[0])
print(preprocessed_answers[0])
```

```
<start> hi , how are you doing ? <end>
<start> i m fine . how about yourself ? <end>
```

3.Tokenization:

Tokenization is used in natural language processing to split paragrap hs and sentences into smaller units that can be more easily assigned meaning. The first step of the NLP process is gathering the data (a se ntence) and breaking it into understandable parts (words).

Program:

```
def tokenize(lang):
    lang_tokenizer = tf.keras.preprocessing.text.Tokenizer(
      filters='')

    #build vocabulary on unique words
    lang_tokenizer.fit_on_texts(lang)

    return lang_tokenizer
```

Dataset:

We are using the dataset provided in the problem. They provide some basic dialogues in question and answer which are commonly used conversation in day-to-day life.

Dataset Link: https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot

INNOVATION:

In this section we explored innovative techniques such as ensemble methods and deep learning architectures to improve the prediction system's accuracy and robustness.

Ensemble methods:

Ensemble methods in machine learning involve constructing multiple models and combining their predictions to produce a more accurate and robust final prediction. The primary goal is to improve the overall performance compared to using a single model.

In the context of chatbots, ensemble methods can be applied to enhance the bot's performance and accuracy in various ways:

Response Aggregation:

Instead of relying on a single model to generate chatbot responses, ensemble methods can combine predictions from multiple models, potentially resulting in more diverse and robust answers. For instance, using multiple language generation models and aggregating their outputs can improve response quality.

Context Fusion:

Chatbots often need to maintain context across conversations. Ensemble methods can help in preserving context by integrating various models that specialize in different aspects of conversation, memory, or intent recognition.

Intent Recognition:

For understanding user intents, combining predictions from multiple classifiers using ensemble methods can improve the accuracy of identifying user intentions, enhancing the chatbot's ability to provide relevant and accurate responses.

Model Diversity:

Employing ensemble methods with diverse models (e.g., different architectures, training data, or pre-processing techniques) can reduce biases and cover a broader range of scenarios, making the chatbot more adaptable and versatile.

Error Correction:

Ensemble methods can be used to detect and rectify errors in responses. If multiple models generate different responses, ensemble methods can help identify and choose the most suitable response, potentially improving accuracy and consistency.

Overall, ensemble methods in chatbot development offer a means to improve accuracy, robustness, and the overall performance of the chatbot by leveraging the strengths of multiple models or approaches. The specific application of ensemble methods would depend on the nature of the chatbot, the available data, and the requirements of the application.

Deep learning architectures:

Several deep learning architectures are commonly used in chatbot development to enhance language understanding, context retention, and response generation. Here are a few of these architectures:

Recurrent Neural Networks (RNNs):

RNNs are used to maintain sequential information in conversations. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) variants of RNNs are often employed to address the vanishing gradient problem and capture long-term dependencies in text data.

Sequence-to-Sequence (Seq2Seq) Models:

These models, based on RNNs or more advanced versions like Transformers, are designed for tasks like language translation and chatbot conversations. They consist of an encoder-decoder architecture to transform input sequences into output sequences.

Attention Mechanisms:

Commonly used in conjunction with sequence-to-sequence models, attention mechanisms improve the ability to focus on different parts of the input sequence when generating the output, leading to better performance in understanding and generating responses.

Transformers:

These models, popularized by architectures like BERT and GPT (Generative Pre-trained Transformer), utilize self-attention mechanisms and have shown significant advancements in language understanding and generation tasks. They are used for various natural language processing tasks, including chatbot development.

Memory Networks:

These architectures help chatbots retain context across conversations by utilizing an external memory component that can be read and written, allowing the chatbot to remember previous dialogue turns.

These deep learning architectures are implemented to varying degrees within chatbot frameworks to enhance the bot's ability to

understand context, remember information across conversations, and generate more contextually relevant and coherent responses. The choice of architecture often depends on the specific requirements of the chatbot's task and the complexity of the conversation it needs to handle.

CODINGS:

Code for website:

HTML and CSS:

```
<!DOCTYPE html>
<html>
<head>
   <title>Ultron</title>
</head>
<style>
   .Heading{
      color: grey;
   }
   body{
      background:linear-gradient(to right ,rgb(59, 3, 7),rgb(226, 74, 74));


   }
   .inp{
     height: 25px;
     width:90%;
```

```
    }
    .send{
        height:25px;
        width:60px;
        border-radius: 50px;
        background-color: rgb(63, 2, 2);
        color:rgb(180, 151, 151);
        border:none;
    }
    .res{
        background:linear-gradient(to right, rgb(255, 255, 106),rgb(107,
107, 7));
        border-radius: 50px;
        padding:5px;


    }
    .cc{
        color:grey;
    }
    .im{
        height: 450px;
    }
</style>
<body>
```

```
    <center>
    <h1 class="Heading">Hi I'm Ultron Here for your
service</h1></center>
    <p class="cc">Welcome TONY:</p>
    <p class="cc">Plz enter below</p>
    {% if response %}
        <p class="res"><b>{{ response }}</b></p>
    {% endif %}
    <center><img src="{{ url_for('serve_static', filename='img-
removebg-preview.png') }}" alt="My Image" class="im">
    </center>
    <form action="/get_destination" method="post" class="ak">
        <input type="text" name="user_input"  class="inp">
        <input type="submit" value="Ask" class="send">
    </form>



</body>
</html>
```

Coding for chat-bot using python:

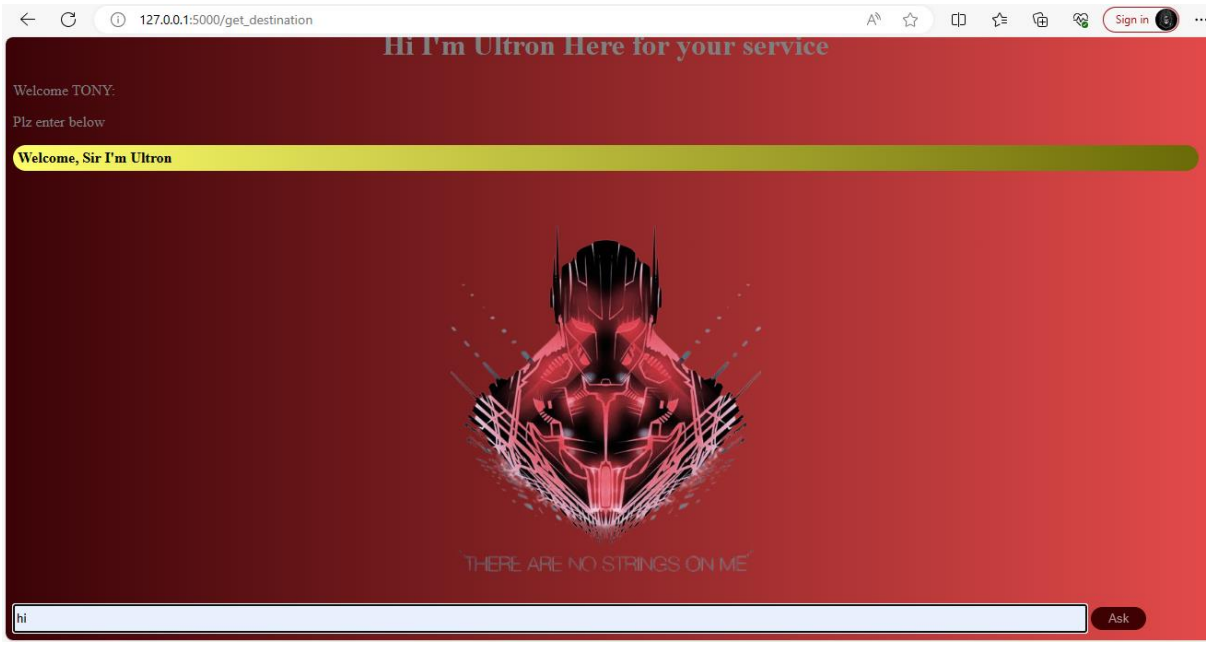
```
import csv
from flask import Flask, request, render_template
import difflib
```

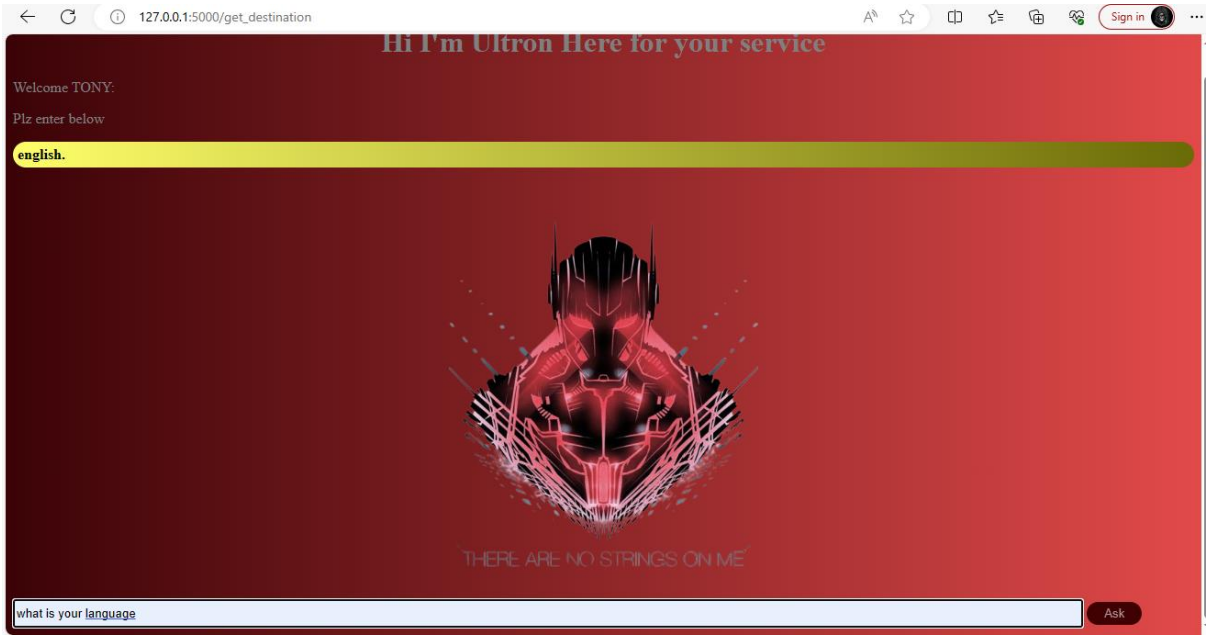
```
app = Flask(__name__)
@app.route('/static/<path:filename>')
def serve_static(filename):
    return send_from_directory('static', filename)

# Read destinations from CSV
destinations = {}
with open('output.csv', 'r', newline='', encoding='utf-8') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        destinations[row['Question'].lower()] = row['Answer']

def find_approximate_match(user_input):
    # Tokenize and normalize the user input
    user_tokens = [word.strip().lower() for word in user_input.split()]

    # Initialize variables for the best match
    best_match = ""
    best_match_score = 0

    # Find the best match using fuzzy string matching
    for question in destinations.keys():
```

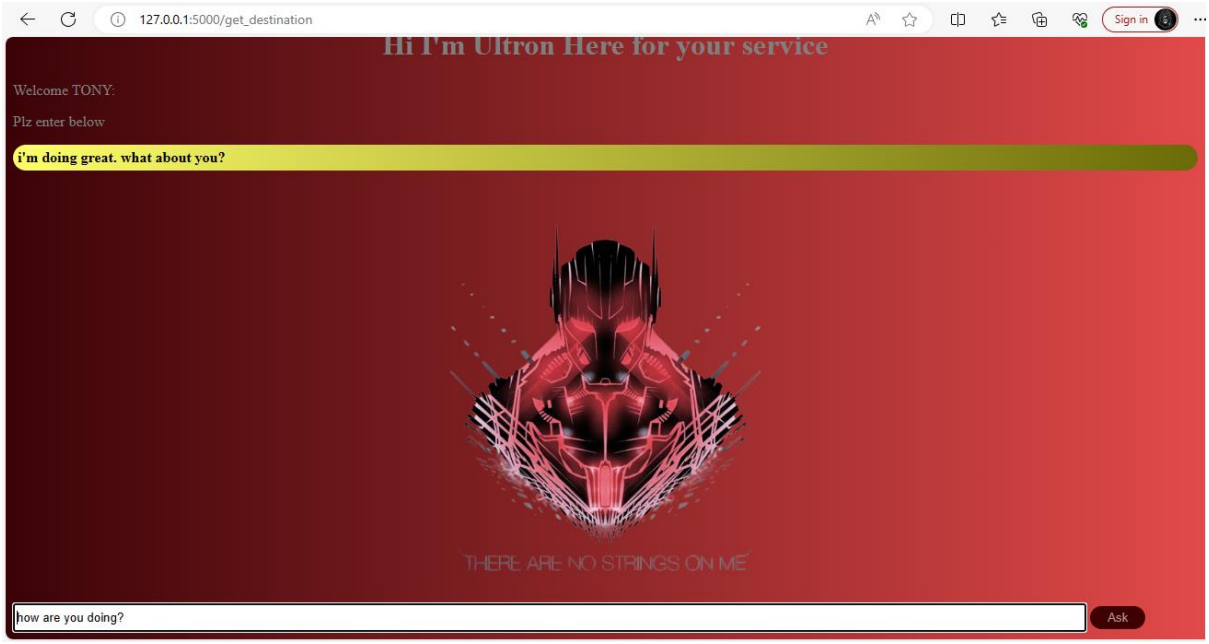
```
        matcher = difflib.SequenceMatcher(None, user_tokens,
question.split())
        match_score = matcher.ratio()
        if match_score > best_match_score:
            best_match_score = match_score
            best_match = question

    return best_match, destinations.get(best_match, 'what are you
blabbering!!!')

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/get_destination', methods=['POST'])
def get_destination():
    user_input = request.form['user_input']
    best_match, response = find_approximate_match(user_input)
    return render_template('index.html', response=response)

if __name__ == '__main__':
    app.run(debug=True)
```

STEPS TO USE THE CODE:

The steps are given in github read me file.


127.0.0.1:5000/get_destination
Sign in
Hi I'm Ultron Here for your service
Welcome TONY:
Plz enter below
Welcome, Sir I'm Ultron
THERE ARE NO STRINGS ON ME
hi
Ask


127.0.0.1:5000/get_destination
Sign in
Hi I'm Ultron Here for your service
Welcome TONY:
Plz enter below
english.
THERE ARE NO STRINGS ON ME
what is your language
Ask



CONCLUSION:

Creating a chatbot using Python can be a rewarding and insightful endeavor. Python, with its versatile libraries and frameworks like NLTK, TensorFlow, or PyTorch, offers an excellent foundation for building chatbots. However, developing a chatbot involves more than just coding; it requires a comprehensive approach that encompasses various elements, from natural language understanding to user experience design.