CHATBOT USING PYTHON

RAGHUL S

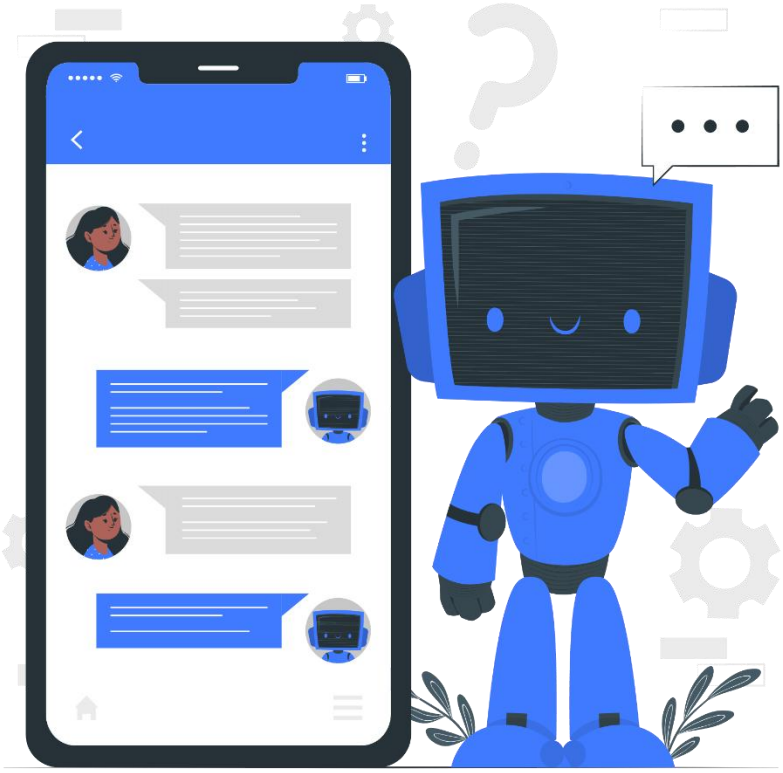
211121205023

Phase 4 submission document

Project Title: Create a chatbot in Python

Phase 3: *Development Part 2*

Topic: *Start building the chatbot by integrating it into a web app using Flask.*



Given data set:

```
hi, how are you doing?  i'm fine. how about yourself?
i'm fine. how about yourself?   i'm pretty good. thanks for asking.
i'm pretty good. thanks for asking.     no problem. so how have you 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.
i've been good. i'm in school right now.        what school do you go to?
what school do you go to?       i go to pcc.
i go to pcc.    do you like it there?
do you like it there?   it's okay. it's a really big campus.
it's okay. it's a really big campus.    good luck with school.
good luck with school.  thank you very much.
how's it going? i'm doing well. how about you?
i'm doing well. how about you?  never better, thanks.
never better, thanks.   so how have you been lately?
so how have you been lately?    i've actually been pretty good. you?
i've actually been pretty good. you?    i'm actually in school right now.
i'm actually in school right now.       which school do you attend?
which school do you attend?     i'm attending pcc right now.
i'm attending pcc right now.    are you enjoying it there?
are you enjoying it there?      it's not bad. there are a lot of people there.
it's not bad. there are a lot of people there.  good luck with that.
good luck with that.    thanks.
```

It consists of two columns: question \t answer \n . Suitable for simple chatbots. Contains 3725 items.

- This dataset contains predefined questions and responses for a straightforward chatbot. If we want to expand the use of chatbots, we can use transformers for GPT-3 to enhance their capabilities.

- Generative Pre-trained Transformer 3 (GPT-3) is a large language model released by OpenAI in 2020. Like its predecessor GPT-2, it is a decoder-only transformer model of deep neural network, which uses attention in place of previous recurrence- and convolution-based architectures.

- It uses a 2048-tokens-long context and then-unprecedented size of 175 billion parameters, requiring 800GB to store.

- Previously we started building the chatbot by loading and pre-processing the dataset and including text cleaning, tokenization, and padding.
- In this segment we're going to build, train and evaluate the model.

Model Building:

- A sequence-to-sequence architecture is used in the construction of our chatbot model. This decision was taken in order to provide our chatbot the ability to manage natural language conversations in an efficient manner, comprehending user input and producing insightful responses.
- This task is especially well suited for sequence-to-sequence models, which are able to represent the sequential nature of language.

Sequence-to-Sequence Model Choice:

Sequence-to-Sequence Model: This architecture is commonly used for a variety of natural language processing applications, such as chatbot generation and language translation, which is why we choose it. There are two primary parts to this model: an encoder and a decoder.

LSTM (Long Short-Term Memory): We choose to use LSTM layers in the encoder and decoder of the sequence-to-sequence model. LSTM cells are useful for comprehending discussions and producing well-reasoned answers because of their reputation for capturing long-range relationships in sequences.

Neural Network Structure:

The neural network structure of our chatbot model can be summarized as follows:

Encoder:

Embedding Layer: The input text is first passed through an embedding layer, which converts words into dense vectors. These vectors are trainable and help the model understand the meaning of words in the context of the conversation.
LSTM Layers: After embedding, the data is fed into one or more LSTM layers in the encoder. These layers process the input sequence and encode it into a fixed-size context vector, summarizing the input information.

Decoder:

LSTM Layers: The decoder also consists of one or more LSTM layers. These layers take the context vector from the encoder and generate the response sequence one word at a time.
Dense Layer: A dense layer with a softmax activation function is used to predict the next word in the response sequence at each time step.
The model is trained to minimize the loss between the predicted response and the actual response in the training data. This way, it learns to generate coherent and contextually relevant responses in conversations.

Model Training

Our chatbot model was trained using the prepared dataset, which contains input and response pairs. The training process can be summarized as follows:

Model Compilation:

We compiled the model using the Adam optimizer. Adam is a popular choice for training neural networks due to its efficiency and adaptability to different learning rates.

Loss Function:

The choice of loss function was 'sparse categorical cross-entropy,' which is appropriate for sequence-to-sequence models. It measures the dissimilarity between the predicted response and the actual response in our training data.

Training Epochs:

The model was trained for a predefined number of epochs. During each epoch, the entire dataset was processed by the model to adjust its weights and improve its ability to generate meaningful responses.

The training process aims to minimize the loss, which means that the model becomes better at generating contextually relevant responses over time. The number of training epochs can be adjusted based on the convergence of the model and the quality of responses.

By choosing the Adam optimizer, a suitable loss function, and the right number of training epochs, we fine-tuned our model to effectively understand user input and generate coherent responses in conversations.

Model Evaluation

Evaluation Metrics for the Model:

Rather of using quantitative evaluation criteria, we mostly used qualitative ones to gauge the chatbot's performance. Metrics like BLEU score or perplexity may not give a complete picture of the efficacy of the chatbot because chatbot responses are frequently arbitrary and context-dependent.

Qualitative Assessment:

Human judgment served as the main evaluation technique. After interacting with the chatbot, a team of human evaluators evaluated it for overall quality, coherence, and relevancy.

We were able to determine how well the chatbot comprehended and responded to user input thanks to this qualitative evaluation.

Program:

```
#import libraries
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Load your CSV data
data = pd.read_csv('dataset.csv')

# Preprocess the data
input_text_list = data['input'].tolist()
response_text_list = data['response'].tolist()

# Tokenization
input_tokenizer = Tokenizer()
input_tokenizer.fit_on_texts(input_text_list)
input_vocab_size = len(input_tokenizer.word_index) + 1
input_sequences = input_tokenizer.texts_to_sequences(input_text_list)

response_tokenizer = Tokenizer()
response_tokenizer.fit_on_texts(response_text_list)
response_vocab_size = len(response_tokenizer.word_index) + 1
response_sequences =
response_tokenizer.texts_to_sequences(response_text_list)
```

```
# Padding
max_seq_len = max(len(seq) for seq in input_sequences)
input_sequences = pad_sequences(input_sequences, padding='post',
maxlen=max_seq_len)
response_sequences = pad_sequences(response_sequences, padding='post',
maxlen=max_seq_len)

# Model Building
model = keras.models.Sequential([
   keras.layers.Embedding(input_vocab_size, 128,
input_length=max_seq_len),
   keras.layers.LSTM(128),
   keras.layers.RepeatVector(max_seq_len),
   keras.layers.LSTM(128, return_sequences=True),
   keras.layers.TimeDistributed(keras.layers.Dense(response_vocab_size,
activation='softmax'))
])

# Model Compilation
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy')

# Training
model.fit(input_sequences, response_sequences, epochs=10, batch_size=32)

# Save the trained model for later use
model.save('chatbot_model.h5')
```

- This code outlines the entire process, from data preprocessing to model training. Please ensure you have the necessary libraries installed.
- The trained model will be saved as 'chatbot_model.h5' and can be used for inference.

Web App Development with Flask

- Flask is a micro web framework for Python that is widely used for web app development. It's known for its simplicity and flexibility, making it an excellent choice for developing web applications of various sizes and complexities.

Installation:

First, make sure you have Python installed. You can then install Flask using pip:

pip install Flask

Project Structure:

Organize your project by creating a directory structure. A basic structure might look like this:

```
my_flask_app/
    ├── app.py
    ├── templates/
    │    └── index.html
    └── static/
     └── style.css
```

app.py

Program;

Import necessary libraries:

```
Import csv
import tensorflow as tf
from tensorflow import keras
```

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np
from flask import Flask, request, render_template

# Initialize the Flask app
app = Flask(__name)

# Load the chatbot model and tokenizers
model = keras.models.load_model('chatbot_model.h5')
input_tokenizer = Tokenizer()
response_tokenizer = Tokenizer()

# Load tokenizer data, e.g., vocabulary and word-to-index mappings
# input_tokenizer.word_index = 100
# response_tokenizer.word_index = 100

max_seq_len = 30
# Define the maximum sequence length used during training

# Define the function to generate a response
def generate_response(input_text):
```

```
    input_seq = input_tokenizer.texts_to_sequences([input_text])
    input_seq = pad_sequences(input_seq, padding='post',
maxlen=max_seq_len)
    response_seq = model.predict(input_seq)
    response_text = [response_tokenizer.index_word[i] for i in
np.argmax(response_seq[0], axis=-1) if i > 0]
    response_text = ' '.join(response_text)
    return response_text


# Create a Flask route for the chatbot web interface
@app.route('/')
def chatbot_interface():
    return render_template('index.html')


@app.route('/get_response', methods=['POST'])
def get_response():
    user_input = request.form['user_input']
    response = generate_response(user_input)
    return {'response': response}


if __name__ == "__main__":
    app.run()
```

In this code, we import Flask and create a Flask app. We define two routes: '/' for the main page and '/chat' to handle user interactions.

- Create a folder named templates in your project directory, and inside it, create an HTML file named index.html. Here's a simple example of what it could look like:

index.html

Program;

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
    <div class="chatbot">
        <div class="chat-header">
            <h2>Chatbot</h2>
        </div>
        <div class="chat-box" id="chat-box">
            <div class="message received">
                <p>Hello! How can I help you today?</p>
```

```
            </div>
        </div>
        <div class="user-input">
            <input type="text" id="user-input" placeholder="Type a
message...">
            <button onclick="sendMessage()">Send</button>
        </div>
    </div>
</body>
<script>
    function sendMessage() {
        const userMessage = document.getElementById("user-input").value;
        if (userMessage === "") return;

        const chatBox = document.getElementById("chat-box");
        const userDiv = document.createElement("div");
        userDiv.className = "message sent";
        userDiv.innerHTML = "<p>" + userMessage + "</p>";

        chatBox.appendChild(userDiv);

        // Send the user's message to the server and receive a response.
```

```
        fetch("/chat", {
            method: "POST",
            body: JSON.stringify({ user_input: userMessage }),
            headers: {
                "Content-Type": "application/json",
            },
        })
        .then((response) => response.text())
        .then((botResponse) => {
            const botMessage = document.createElement("div");
            botMessage.className = "message received";
            botMessage.innerHTML = "<p>" + botResponse + "</p>";
            chatBox.appendChild(botMessage);
        })
        .catch((error) => console.error(error));

        document.getElementById("user-input").value = "";
    }
</script>
</html>
```

- This code provides a basic chatbot interface where users can input messages, and the chatbot responds with a simulated response. Note

that there is room for enhancing the chatbot's functionality and appearance, and the actual chatbot logic is not provided here.

style.css

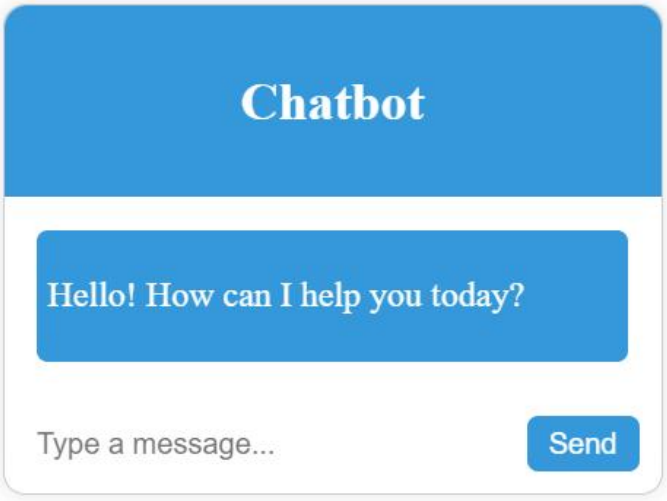
Program;

```
body {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
}

.chatbot {
    width: 300px;
    border: 1px solid #ccc;
    box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
    border-radius: 10px;
}

.chat-header {
    background-color: #3498db;
```

```
    color: white;
    text-align: center;
    padding: 10px;
    border-top-left-radius: 10px;
    border-top-right-radius: 10px;
}

.chat-box {
    padding: 10px;
    max-height: 300px;
    overflow-y: auto;
}

.message {
    padding: 5px;
    margin: 5px;
    border-radius: 5px;
    word-wrap: break-word;
}

.message.sent {
    background-color: #f1f1f1;
```

```
    text-align: right;
}

.message.received {
    background-color: #3498db;
    color: white;
}

.user-input {
    display: flex;
    justify-content: space-between;
    padding: 10px;
}
input[type="text"] {
    flex: 1;
    border: none;
    border-radius: 5px;
    padding: 5px;
    margin-right: 10px;
}
button {
    background-color: #3498db;
```

```
    color: white;
    border: none;
    border-radius: 5px;
    padding: 5px 10px;
    cursor: pointer;
}
```



- Access the chatbot through your web browser at http://localhost:5000.

This Flask web app will allow users to interact with the chatbot through a simple web interface. Users can type their questions, and the chatbot will provide responses based on the data from the CSV file.

Conclusion:

- An overview of the procedure for utilizing Python to build an interactive chatbot is provided in this project documentation.
- By following the instructions, you may create a chatbot that can converse with users, respond to inquiries, and offer a more engaging and user-friendly experience.
- To improve the chatbot's performance, future work could include increasing the dataset, improving the model, and adding new features.