Project Documentation & Submission

Project : CREATE A CHATBOT USING PYTHON

Problem Definition :

- ☐ The problem at hand is to design and implement a chatbot using Python that can effectively engage in natural language conversations, understand and interpret user queries, and provide contextually relevant responses.

- ☐ This chatbot should be capable of various functionalities, including answering questions, providing recommendations, offering assistance with tasks, and even performing actions such as setting reminders or sending emails.

- ☐ It should maintain context throughout the conversation, remembering prior interactions and ensuring a coherent dialogue. Additionally, data privacy and security are crucial considerations, ensuring that user data is handled in compliance with relevant regulations.

☐ The success of this project will be measured by user satisfaction, the accuracy of responses, task completion rates, and the bot's ability to continuously improve through user feedback and data analysis.

Design thinking :

☐ Empathize:

- Begin by understanding the needs, desires, and challenges of the users who will interact with the chatbot. This step may include conducting user interviews, surveys, and observing user behaviors to gain insights into their preferences and pain points.

☐ Define:

- Define a clear problem statement that the chatbot aims to address. For example, if the chatbot is designed for healthcare, the problem statement could be about improving patient engagement and providing medical information through a conversational interface.

☐ Ideate:

- Generate creative ideas and brainstorm potential solutions to the defined problem. Consider how RNN-based models can enhance the chatbot's conversational capabilities, context management, and user engagement.

☐ Prototype:

- Create a prototype of the chatbot using Python and the RNN model. This initial version serves as a tangible representation of the concept, focusing on the conversation flow, user interface, and response generation using simplified models.

☐ Test:

- Gather a group of real users to interact with the chatbot prototype. Collect feedback on their experiences, the quality of responses, and the chatbot's ability to understand and generate natural language text.

☐ Iterate:

- Use the feedback received during testing to refine and improve the chatbot prototype. Consider refining the RNN model, enhancing the user interface, and making user experience improvements.

☐ Implement:

- Develop the full-fledged chatbot using Python and integrate the RNN model. Ensure that the chatbot can generate context-aware responses and engage users in meaningful conversations.

☐ Test (Again):

- Conduct thorough testing to verify that the chatbot functions as intended. Test the RNN model to ensure it generates human-like responses and maintains context throughout the conversation.

☐ Deploy and Monitor:

- Deploy the chatbot to the intended platforms (e.g., websites, messaging apps, voice interfaces). Continuously monitor its performance, gather user feedback, and analyze data on user interactions.

☐ Feedback and Improvement:

- Maintain an ongoing feedback loop with users and stakeholders. Use user feedback and data analysis to make incremental improvements to the chatbot. This may involve fine-tuning the RNN model, enhancing user interfaces, and expanding the chatbot's capabilities based on evolving user needs.

Throughout this design thinking process, the emphasis is on a user-centric approach, ensuring that the chatbot not only addresses the defined problem but also provides a valuable and engaging user experience. The iterative nature of this approach allows the chatbot to evolve over time, making it more effective and user-friendly.

Code Implementation :

- **Step 1:** Setup an Environment

  Make sure that Flask and the necessary Python libraries installed:

  pip install flask tensorflow

- **Step 2:** Data Preparation

  Prepare a text dataset for training the chatbot.

- **Step 3:** Build and Train the RNN Model

Here's the code for building and training a simple character-level RNN model using Keras. The following code that assumes a dataset in a file named **conversations.txt**, where each line contains a conversation.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import
pad_sequences

# Load a dataset
with open("conversations.txt", "r") as file:
    conversations = file.read().splitlines()

# Preprocess your data
tokenizer = Tokenizer(char_level=True)
tokenizer.fit_on_texts(conversations)
total_words = len(tokenizer.word_index) + 1

input_sequences = []
for line in conversations:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)

max_sequence_length = max([len(x) for x in
input_sequences])
input_sequences = pad_sequences(input_sequences,
maxlen=max_sequence_length, padding='pre')

# Create the RNN model
model = Sequential()
```

```
model.add(Embedding(total_words, 100,
input_length=max_sequence_length))
model.add(LSTM(100))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy',
optimizer='adam')

# Prepare the training data
X, y = input_sequences[:, :-1], input_sequences[:, -1]
y = tf.keras.utils.to_categorical(y,
num_classes=total_words)

# Train the model
model.fit(X, y, epochs=50, verbose=1)

# Save the model for later use
model.save("chatbot_rnn_model.h5")
```

Step 4: *Create the Flask Application*

Here's a basic Flask application that loads the trained model and provides a simple web interface for chat:

```
from flask import Flask, request, render_template
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import load_model

app = Flask(__name)
```

```
model = load_model("chatbot_rnn_model.h5")

# Load the tokenizer used for preprocessing
tokenizer = Tokenizer(char_level=True)
tokenizer.fit_on_texts(conversations)

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

@app.route('/chat', methods=['POST'])
def chat():
    user_message = request.form['user_message']

   # Preprocess user's input
   input_sequence =
tokenizer.texts_to_sequences([user_message])[0]
    input_sequence = pad_sequences([input_sequence],
maxlen=max_sequence_length-1, padding='pre')

   # Generate a response using the RNN model
   predicted_word_index =
np.argmax(model.predict(input_sequence))
    response = tokenizer.index_word[predicted_word_index]

   return response

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

Step 5: HTML Template

Create an HTML template (**templates/index.html**) for a chat interface:

```
<!DOCTYPE html>
<html>
<head>
    <title>Chatbot</title>
</head>
<body>
    <h1>Chatbot</h1>
    <div id="chat-container">
        <div class="user-message">User's message</div>
        <div class="bot-message">Bot's response</div>
    </div>
    <form action="/chat" method="post">
        <input type="text" name="user_message"
id="user_message" placeholder="Type a message..."
autocomplete="off">
        <input type="submit" value="Send">
    </form>
</body>
</html>
```

Step 6: Run the Application

Run the Flask application :

```
python mychatbot.py
```

Conclusion :

The chatbot Flask application should now be up and running. Users can input messages, and the RNN-based chatbot will generate responses based on the training data.