PROJECT TITLE: <u>CREATE A CHATBOT IN PYTHON</u>

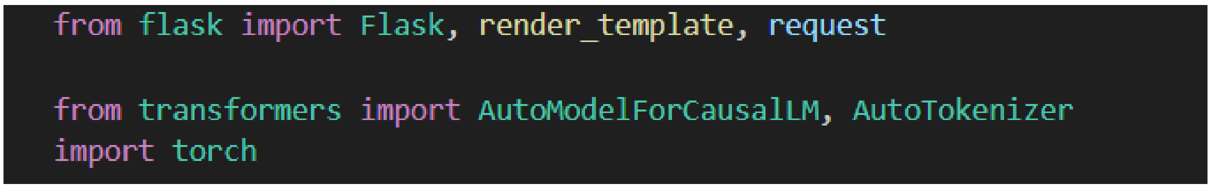
PHASE 4: <u>Development Part 2</u>

Continue building the chatbot by integrating it into a web app using Flask.

INTRODUCTION:

- ✓ Flask is commonly used to create web-based chatbot applications.
- ✓ It provides the necessary tools and libraries to build a web-based interface for chatbots, making it easier to interact with users through a web-based messaging platform.
- ✓ Flask allows developers to create routes and handle HTTP requests, which can be essential for receiving user input and providing responses from a chatbot.
- ✓ You can integrate a chatbot developed with natural language processing capabilities into a Flask-based web application to create a user-friendly interface for users to interact with the chatbot via a web browser.

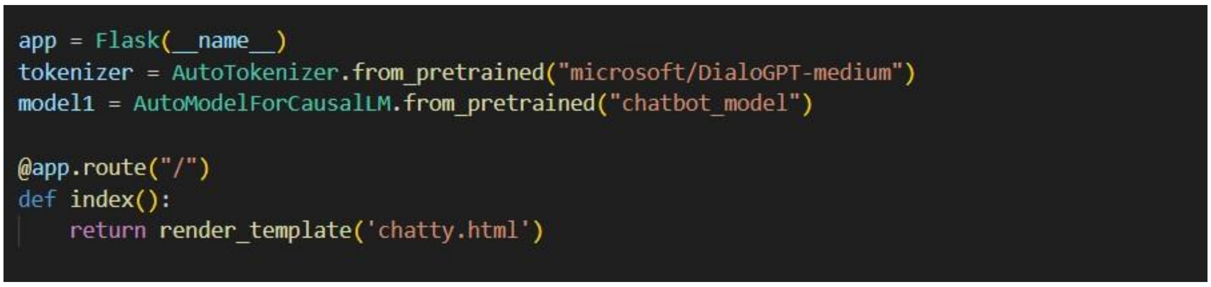
FLASK INTEGRATION:



```python
from flask import Flask, render_template, request

from transformers import AutoModelForCausalLM, AutoTokenizer
import torch
```

- ✓ This line imports necessary modules from the Flask web framework.
- ✓ Flask is used to create a web application, and
    - **Flask** is the main class for that purpose.
    - **Render_template** is used for rendering HTML templates,
    - **request** is used for handling HTTP requests, and **jsonify** is used for returning JSON responses.

- **AutoModelForCausalLM** and **AutoTokenizer** are commonly used for loading pre-trained language models and their corresponding tokenizers.
- **AutoModel** is also imported, although it's not used in your provided code.
- **import torch**: PyTorch is often used in conjunction with Hugging Face Transformers to work with neural networks and machine learning models.



```
app = Flask(__name__)
tokenizer = AutoTokenizer.from_pretrained("microsoft/DialoGPT-medium")
model1 = AutoModelForCausalLM.from_pretrained("chatbot_model")

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

✓ By calling **Flask(__name__)**, you are creating an instance of this class, which represents your web application.

✓ This line initializes a tokenizer using the Hugging Face Transformers library.

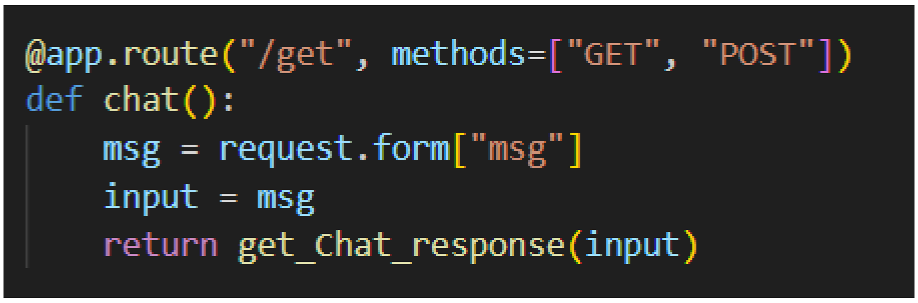
✓ **The AutoTokenizer:**
  - It is a convenient way to load a pre-trained tokenizer for a specific language model.
  - Tokenizers are used to break down text into smaller units (tokens) that can be processed by a language model.

✓ **AutoModelForCausalLM:**
  - This class is used to load a pre-trained language model for causal language modeling.
  - Causal language models are often used for text generation tasks where the order of tokens is important, such as chatbots and text completion.

✓ **@app.route("/")**: This is a decorator in Flask that associates the URL route "/" (the root URL) with the following function, **index()**.

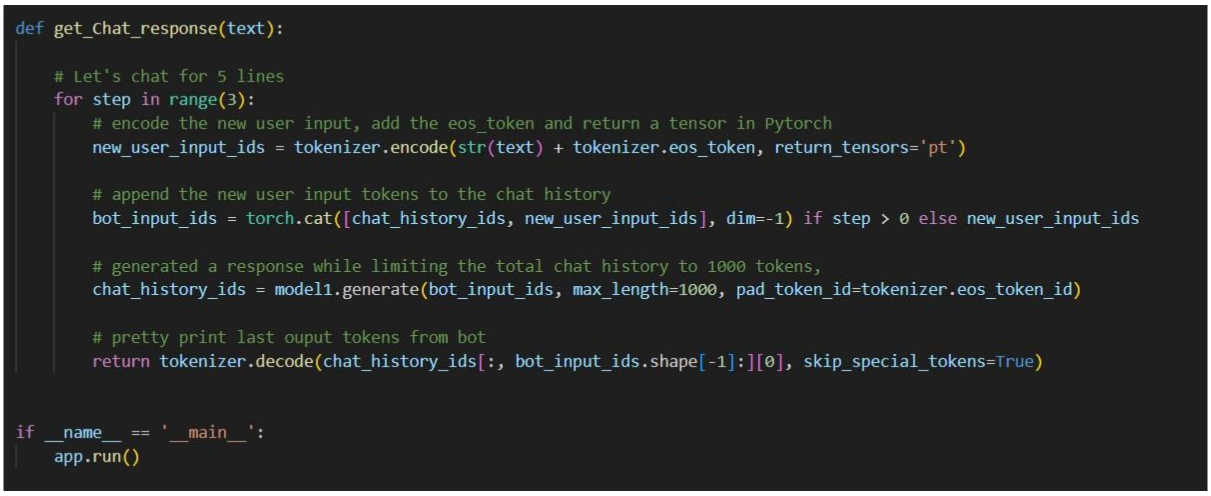
✓ **def index():** This is a Python function definition. It defines a function named index that will be called when a user accesses the root URL.

✓ **return render_template('chatty.html'):** Inside the index function, it returns the result of rendering a template called 'chatty.html'.



```python
@app.route("/get", methods=["GET", "POST"])
def chat():
    msg = request.form["msg"]
    input = msg
    return get_Chat_response(input)
```

- ✓ **@app.route("/get", methods=["GET", "POST"])**:
  - This is a decorator in Flask.
  - It associates the function below it (**chat**) with a specific URL route. In this case, it associates the **/get** route with the **chat** function, and it specifies that this route can handle both GET and POST requests.
- ✓ **def chat()**:
  - This is a Python function definition. It defines a function named **chat** that will be executed when a user accesses the **/get** route of your web application.
- ✓ **msg = request.form["msg"]**:
  - This line retrieves the value of the "msg" parameter from the form data in the request.
  - It assumes that a user has submitted a form with a field named "msg" containing their input message.
  - The value is stored in the **msg** variable.
- ✓ **input = msg**:
  - The value of **msg** is assigned to a variable named **input**.
  - This step is optional and not necessary; you could use **msg** directly in the following line.
- ✓ **return get_Chat_response(input)**:
  - This line calls the **get_Chat_response** function with the user's input (stored in the **input** variable) as an argument.
  - The **get_Chat_response** function generates a response based on the user's input and returns it as an HTTP response.

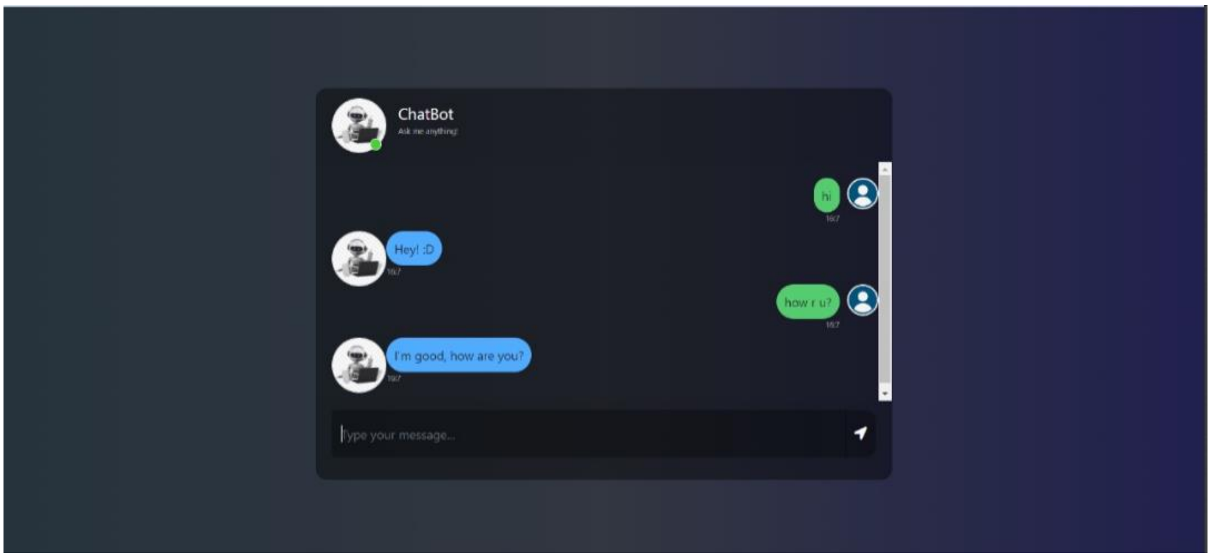


```python
def get_Chat_response(text):

    # Let's chat for 5 lines
    for step in range(3):
        # encode the new user input, add the eos_token and return a tensor in Pytorch
        new_user_input_ids = tokenizer.encode(str(text) + tokenizer.eos_token, return_tensors='pt')

        # append the new user input tokens to the chat history
        bot_input_ids = torch.cat([chat_history_ids, new_user_input_ids], dim=-1) if step > 0 else new_user_input_ids

        # generated a response while limiting the total chat history to 1000 tokens,
        chat_history_ids = model1.generate(bot_input_ids, max_length=1000, pad_token_id=tokenizer.eos_token_id)

        # pretty print last ouput tokens from bot
        return tokenizer.decode(chat_history_ids[:, bot_input_ids.shape[-1]:][0], skip_special_tokens=True)


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

- ✓ **def get_Chat_response(text)**: This is a Python function definition. It takes a **text** parameter, which presumably represents the user's input.
    - For each iteration, it encodes the user's input, adds an end-of-sequence token (**eos_token**), and returns a PyTorch tensor.
    - It appends the new user input tokens to the chat history.
    - It generates a response while limiting the total chat history to 1000 tokens.
    - Finally, it decodes the response and returns it.
- ✓ **if __name__ == '__main__':**
    - This is a common Python idiom used to check if the script is being run as the main program.
    - If it's the main program (not imported as a module in another script), the following code block is executed.
- ✓ **app.run()**:It starts the Flask development server, which allows your web application to listen for incoming HTTP requests and serve web pages.

APPERANCE OF THE CHATBOT:

1. GREETINGS:



ChatBot
Ask me anything!

hi
16:7

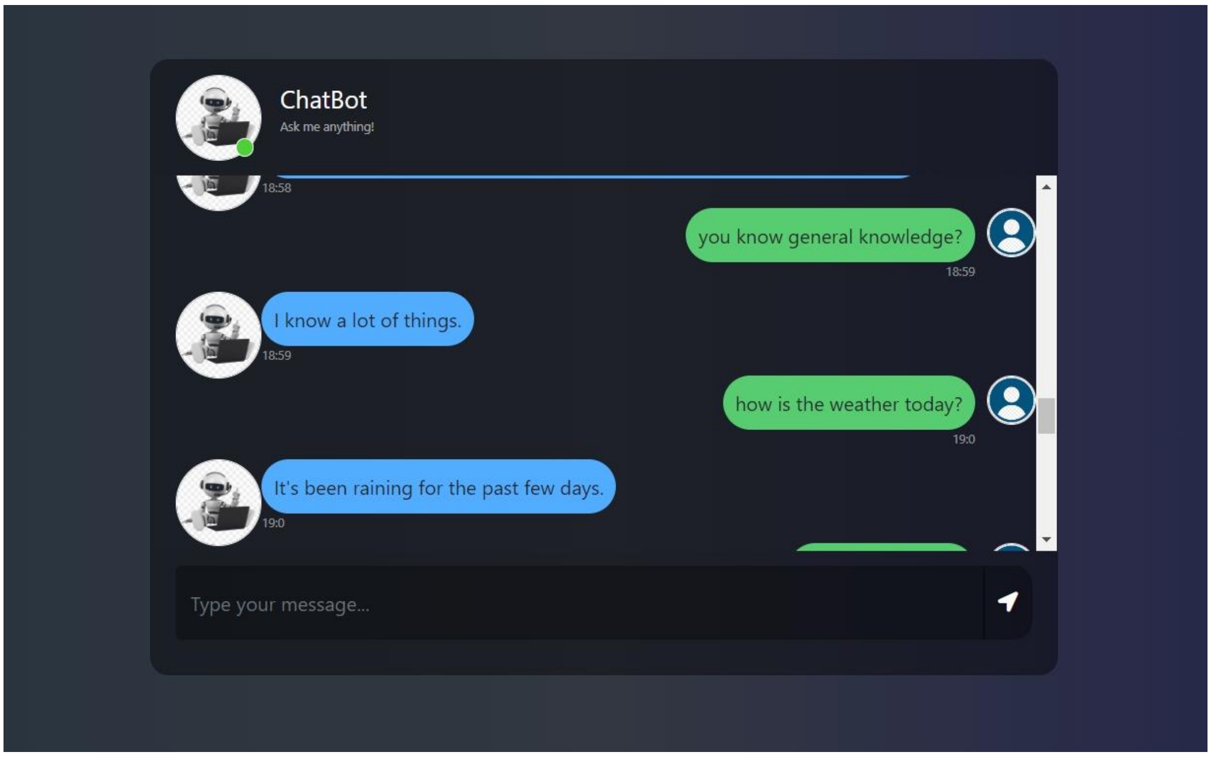
Hey! :D
16:7

how r u?
16:7

I'm good, how are you?
16:7

Type your message...

2. INFORMATION SHARING:



ChatBot
Ask me anything!

18:58

you know general knowledge?
18:59

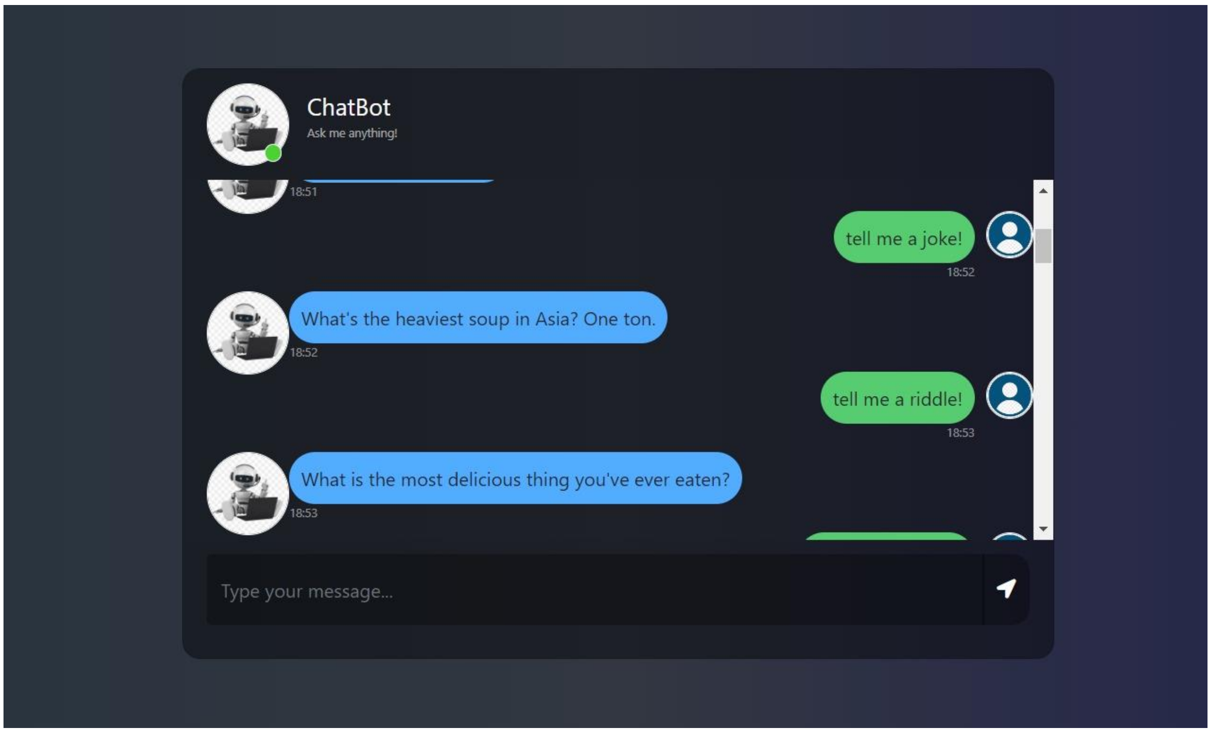
I know a lot of things.
18:59

how is the weather today?
19:0

It's been raining for the past few days.
19:0

Type your message...

3. **ENTERTAINMENT:** can provide riddles and jokes



ChatBot
Ask me anything!

18:51

tell me a joke!
18:52

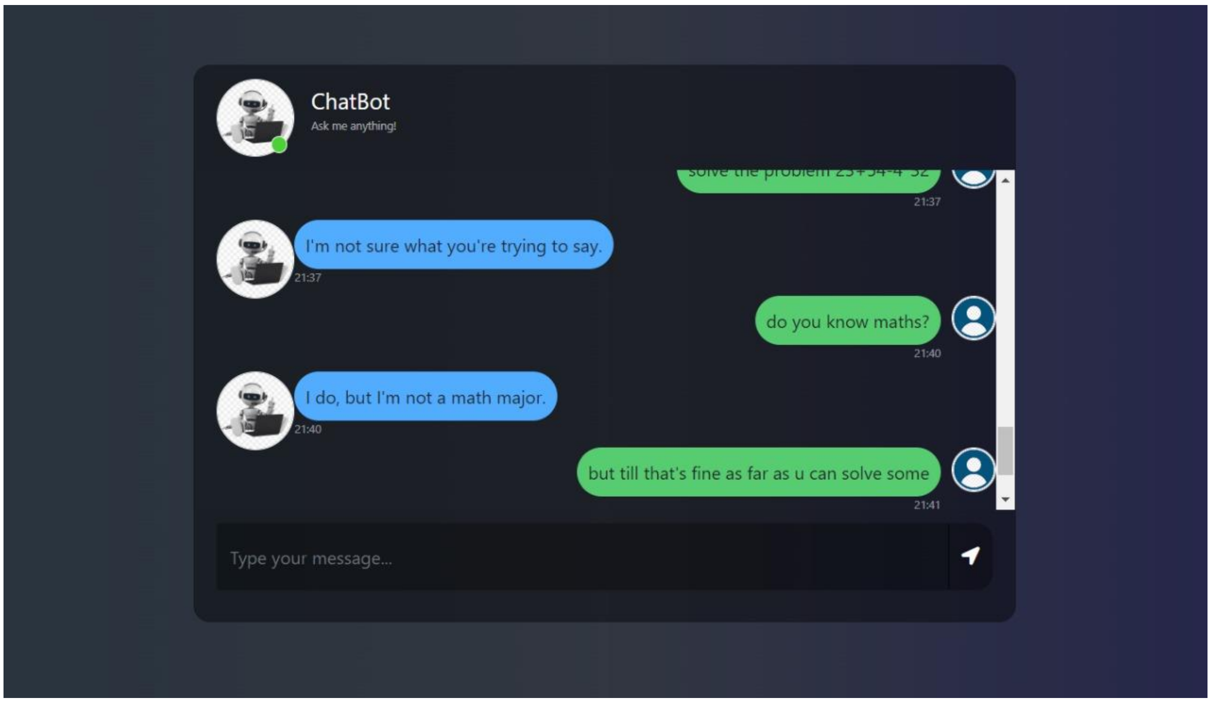
What's the heaviest soup in Asia? One ton.
18:52

tell me a riddle!
18:53

What is the most delicious thing you've ever eaten?
18:53

Type your message...

4. **ACT AS HUMAN INTELLIGENCE:**



ChatBot
Ask me anything!

solve the problem 23+34-4*32
21:37

I'm not sure what you're trying to say.
21:37

do you know maths?
21:40

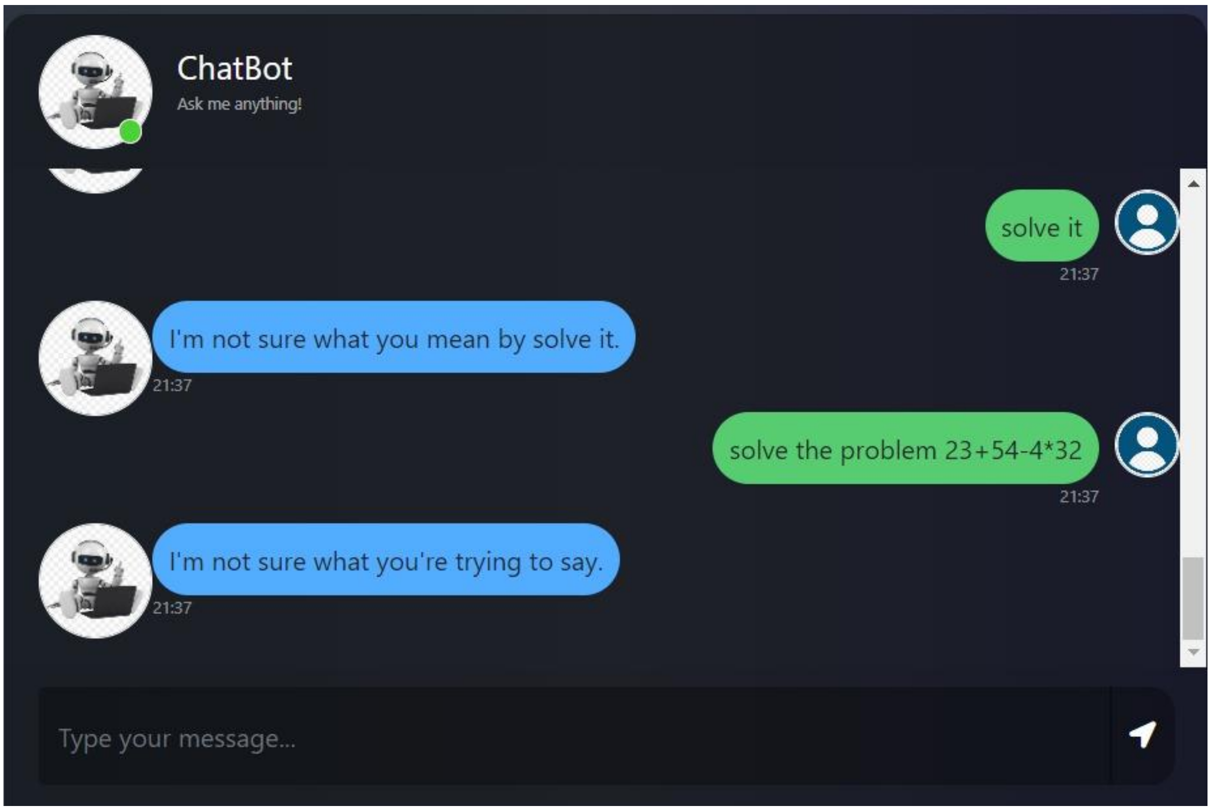
I do, but I'm not a math major.
21:40

but till that's fine as far as u can solve some
21:41

Type your message...

5. ERROR HANDLING:

If the chatbot doesn't understand a query, it should respond with helpful suggestions or ask for clarification.



ChatBot
Ask me anything!

solve it
21:37

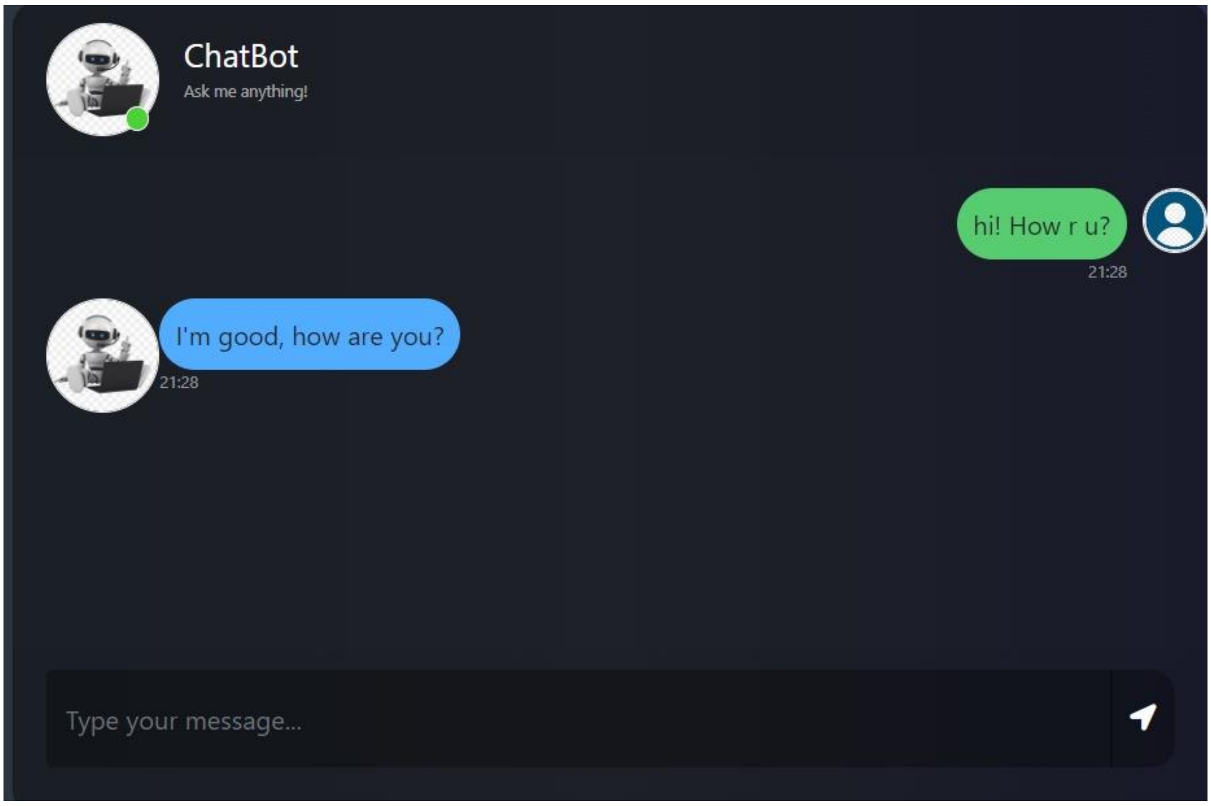
I'm not sure what you mean by solve it.
21:37

solve the problem 23+54-4*32
21:37

I'm not sure what you're trying to say.
21:37

Type your message...

6. INNOVATION:



ChatBot
Ask me anything!

hi! How r u?
21:28

I'm good, how are you?
21:28

Type your message...

This chatbot can provide responses even if it is written in abbreviations as made by human.

CONCLUSION:

The response provided earlier is a result of the capabilities of the pre-training model, which is a sophisticated artificial intelligence language model developed by Dialo-GPT. By utilizing this pre-training model, the chatbot can analyze and interpret natural language input, generating appropriate and contextually relevant responses based on the patterns and information it has learned during its training.