Project Documentation

Name: KRISHNA SAKETH RAM

Reg No: 21BCE7455

Project: Blackbucks Chatbot – AI, ML and DS

Abstract

In this paper, we improved an AI chatbot system that can have interactive discussions with the users that are developed in chatting. With changeable personalities and timestamped chat histories, the chatbot can mimic human-like conversation thanks to its sophisticated natural language processing (NLP) models.

The main driving force behind this effort is the development of an adaptable and user-friendly chatbot that can accommodate a range of conversational tones, from neutral to formal and sardonic, improving user engagement through context awareness and flexibility.

This transformer-based sequence-to-sequence language model, namely “Facebook's blenderbot-400M-distill" model, which has been pre-trained for conversational AI tasks, forms the basis of the system. The project makes use of Gradio to create a web interface, PyTorch as the deep learning framework for effective model inference, and the Hugging Face Transformers library to access the model and its associated tokenizer.

Introduction

A game-changing breakthrough, conversational artificial intelligence (AI) allows robots to recognize, process, and react to human language in a natural and intuitive way. The creation of chatbots—software agents that can mimic text or voice conversations with users—is one of the most well-known uses of conversational AI. Chatbots are essential in today's fast-paced digital world because they provide immediate responses, lessen human labor, and improve user experience in a variety of sectors, including e-commerce, healthcare, education, and customer service. The need for efficiency, individualized contacts, and round-the-clock availability has made them far more important.

The goal of this project is to create a sophisticated AI chatbot with the BlenderBot 400M Distilled model, a cutting-edge transformer-based language model. The chatbot is made to converse with users in a manner similar to that of a human, changing its tone and answers according to certain personalities. The goal is to develop a conversational agent that is adaptable, context-aware, and simple to use so that it can meet a variety of interaction requirements while being coherent and fluent. The project's scope includes using Hugging Face Transformers to develop the model, deploying it through a Gradio-built user-friendly web interface, and turning on personality conditioning and session-based chat history to increase user engagement.

Problem Statement

Despite their widespread use, traditional chatbots frequently have trouble having conversations that are fluid, emotionally compelling, and contextually relevant. Many of the systems in use today are based on strict rule-based procedures or superficial machine learning models that are only capable of handling predetermined answers and limited intentions. Particularly when user input deviates from expected patterns, these restrictions lead to generic, repetitive, and frequently ineffective interactions. These flaws not only make the user experience worse, but they also make it more difficult for chatbots to be used in situations where intelligent and natural conversation is needed.

By deploying a sophisticated chatbot based on the Facebook BlenderBot 400M Distilled model—a transformer-based architecture that has been pre-trained on massive conversational datasets—this research tackles these issues. In contrast to conventional methods, this model makes advantage of deep learning to comprehend context, produce insightful responses, and even modify its personality in response to user preferences. This research is necessary because there is a need for AI conversational agents that are more responsive, natural, and human-like than simple question-answering. The use of chat history, personality training, and a web interface improves usability and qualifies the chatbot for practical uses.

This chatbot is unique because of its versatility and capacity to mimic a range of conversational tenors, from formal to sardonic, which makes it both entertaining and educational. With its adaptable interface and lightweight, deployment-friendly design, the chatbot is a useful tool and scalable basis for upcoming AI-powered communication systems.

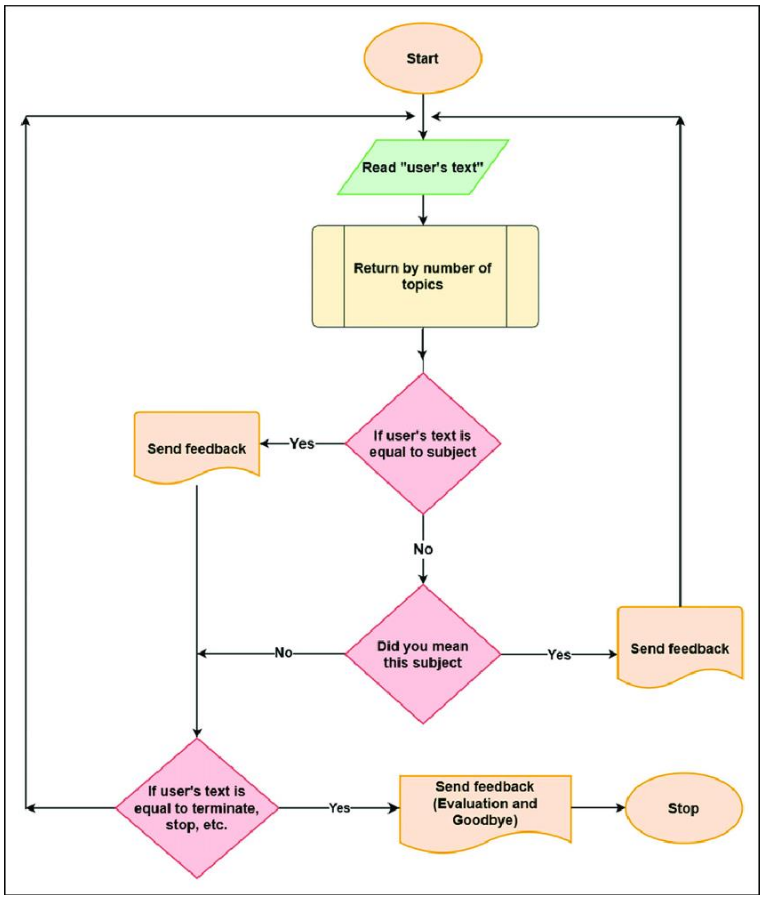
System Architecture

This chatbot's architecture is organized and modular to guarantee user engagement, scalability, and efficiency. Fundamentally, the system combines a number of elements that function as a cohesive whole to provide a conversational experience that is human-like. The Hugging Face Transformers module is used for natural language processing, PyTorch is used for model execution, and Gradio is used for the web-based user interface in this Python-based chatbot.

Gradio Blocks, which offer a versatile and interactive frontend, are used in the construction of the chatbot's interface. In addition to a text input area where users can type their messages, users can choose the chatbot's personality (Neutral, Friendly, Professional, or Sarcastic), which enables the bot to customize its responses. Usability is improved with additional UI buttons that let users send messages, end a conversation, or download chat history.

The BlenderBot model then receives the processed input and uses autoregressive sampling approaches (top-k and top-p) to produce a response. The response is timestamped and added to the global discussion history after being decoded from token IDs back into legible text. The history is reduced to the most recent exchanges in order to preserve system performance and prevent memory overload.

Lastly, a chatbox containing the user's message and the chatbot's response is presented, together with the timestamps for each. In order to ensure dependability, the system also has strong error handling to identify problems like invalid token ranges. These elements work together to create a seamless pipeline that preserves clarity and performance while facilitating dynamic, personality-aware discussions.


Start
Read "user's text"
Return by number of topics
If user's text is equal to subject
Yes
Send feedback
No
Did you mean this subject
No
Yes
Send feedback
If user's text is equal to terminate, stop, etc.
Yes
Send feedback (Evaluation and Goodbye)
Stop

This flowchart shows how a chatbot-based system works. To find pertinent topics, it first reads the user-inputted text and analyzes it. The chatbot instantly provides the relevant feedback if the user's input precisely corresponds to a recognized topic. If it doesn't match exactly, the algorithm checks to see if the user meant a specific subject in an attempt to decipher their intent. Feedback is supplied in accordance with the discovery of a likely match. In the event that no match is discovered, the system determines whether the user is attempting to terminate the discussion by using terms like "terminate" or "stop." If so, the conversation is terminated and a final feedback message is sent. If not, the system repeats the interaction by looping back to read fresh user input.

Modules Description

The chatbot system's architecture is modularized to provide improved user engagement, maintainability, and clarity. Each part is in charge of carrying out a certain task that helps the chatbot run smoothly and generate intelligent responses.

1. **Input-preprocessing:**

A Gradio-based interface receives user input and combines it with recent conversation history and a selected personality prompt. To give the model context, this combined input is organized into a prompt format. Hugging Face's tokenizer tailored to the BlenderBot 400M model is used for tokenization and encoding. To keep the model from becoming overloaded with lengthy input sequences, the procedure incorporates truncation and max-length management.

2. **Model loading:**

Together with its matching tokenizer, Hugging Face's AutoModelForSeq2SeqLM API is used to load the pre-trained BlenderBot 400M Distilled model. To provide reliable model inference, a pad_token_id is manually set if it is not predefined. Responding to the structured input prompt with context-aware responses is the responsibility of this model.

3. **Response Generation:**

After being tokenized, the input is sent into the BlenderBot model, which generates replies using autoregressive decoding and top-k and top-p sampling approaches. The tokenizer is then used to decode the output tokens back into text that can be read by humans. To prevent runtime issues, safety checks are in place to make sure all token IDs are within the model's vocabulary range.

4. **Personality conditioning:**

The chatbot offers personality cues including Neutral, Friendly, Professional, and Sarcastic to make discussions more interesting and flexible. The tone and style of the chatbot's responses are determined by these personality settings. The model's interpretation and response to user inquiries are influenced by the chosen personality, which is appended to the input prompt.

5. **Chat History maintains:**

A collection of tuples comprising user messages, bot responses, and timestamps is kept by the chatbot as a global history of recent encounters. This history provides context retention for cohesive multi-turn talks. To maintain the chatbot's responsiveness and balance performance and relevancy, a cap is placed on the quantity of stored exchanges.

6. **Front-End Interface (Gradio):**

Gradio Blocks, which are used in the development of the user interface, include elements like as checkboxes, chat display panels, dropdowns for selecting a personality, and buttons for sending, finishing, or preserving talks. Real-time engagement with feedback mechanisms is provided by the user-friendly interface. Additionally, it has the ability to download conversation logs as a structured text file, which improves usability and traceability.

Technologies Used

A variety of contemporary, open-source technologies are used in this chatbot project to enable intelligent, context-aware, and real-time conversational engagements. Python, a popular and adaptable programming language perfect for creating AI-driven apps, is used for the development. In order to easily use pre-trained language models, such as the Facebook BlenderBot 400M Distilled model utilized in this project, the chatbot depends on the Hugging Face Transformers library. Using large-scale transformer models, this library makes loading, tokenizing, and text generation easier.

The system makes advantage of PyTorch, a potent machine learning framework renowned for its adaptability and high-performance tensor computations, to carry out the deep learning operations effectively. The transformer model's backend, PyTorch, manages model inference and sampling tasks during chatbot responses.

Gradio, a Python-based framework that enables developers to easily design interactive web interfaces for machine learning models, is used to build the chatbot's frontend and deployment. Gradio offers features like buttons, dropdown menus, chat displays, and text input boxes that let users interact with the chatbot in an intuitive way.

In addition to these fundamental technologies, the project code was written, debugged, and organized using optional tools such as Visual Studio Code (VS Code), which served as the integrated development environment (IDE). Git was used for collaborative development and version control, guaranteeing effective tracking and management of project modifications.

These technologies work together to create a seamless stack that facilitates the creation and implementation of an AI chatbot system that is reliable, responsive, and scalable.

Algorithm Used

The chatbot in this project is powered by the Facebook BlenderBot 400M Distilled model, a transformer-based sequence-to-sequence language model designed for conversational AI tasks.

Model Type: Transformer (Seq2Seq)

Framework: Hugging Face Transformers (PyTorch backend)

Pretrained Model: facebook/blenderbot-400M-distil

Inference Technique: Autoregressive generation with sampling (top-k, top-p nucleus sampling)

Personality Conditioning: Chatbot responses can be customized using personality prompts to modify the tone of replies.

Fallback: Includes a preset Q&A dictionary for deterministic answers to common questions.

Dataset and Required Libraries

No external training dataset was used for this project. A pre-trained transformer-based conversational model, facebook/blenderbot-400M-distill, which is publicly accessible through the Hugging Face Transformers library, was used instead.

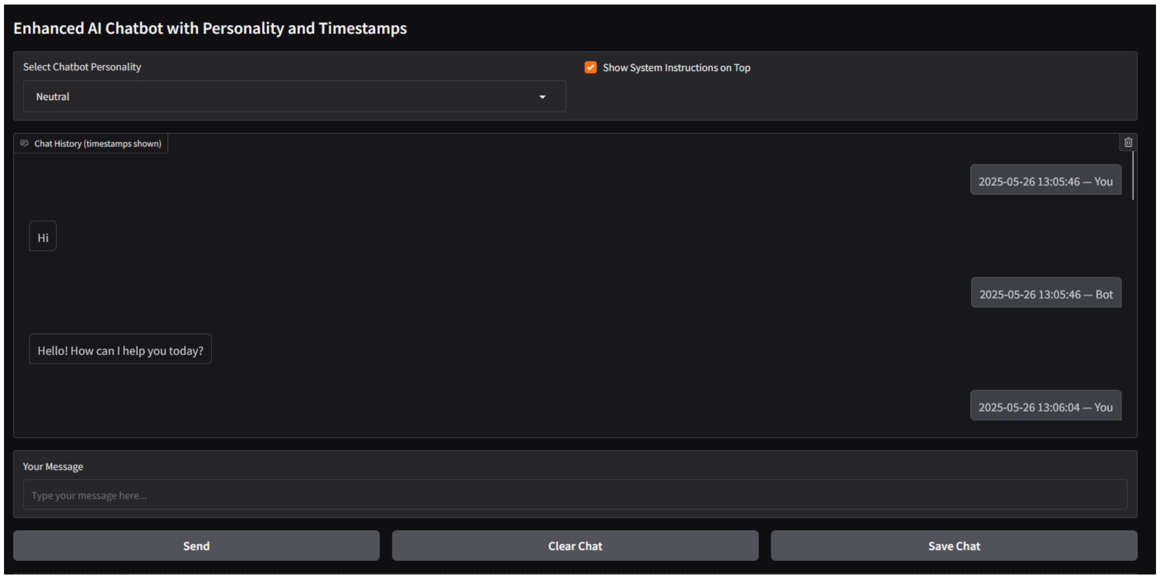
Libraries:

- **transformers** – for using BlenderBot from Hugging Face.
- **torch** – for backend deep learning operations.
- **gradio** – for deploying the chatbot as a simple, interactive web app.
- **os** – for path handling.
- **datetime** – for timestamping chat logs (if saving).
- **logging** – for saving and tracking chat history.

Installation Command:

```
pip install transformers torch gradio
```

Result



Enhanced AI Chatbot with Personality and Timestamps

Select Chatbot Personality: Neutral

☑ Show System Instructions on Top

Chat History (timestamps shown)

2025-05-26 13:05:46 — You

Hi

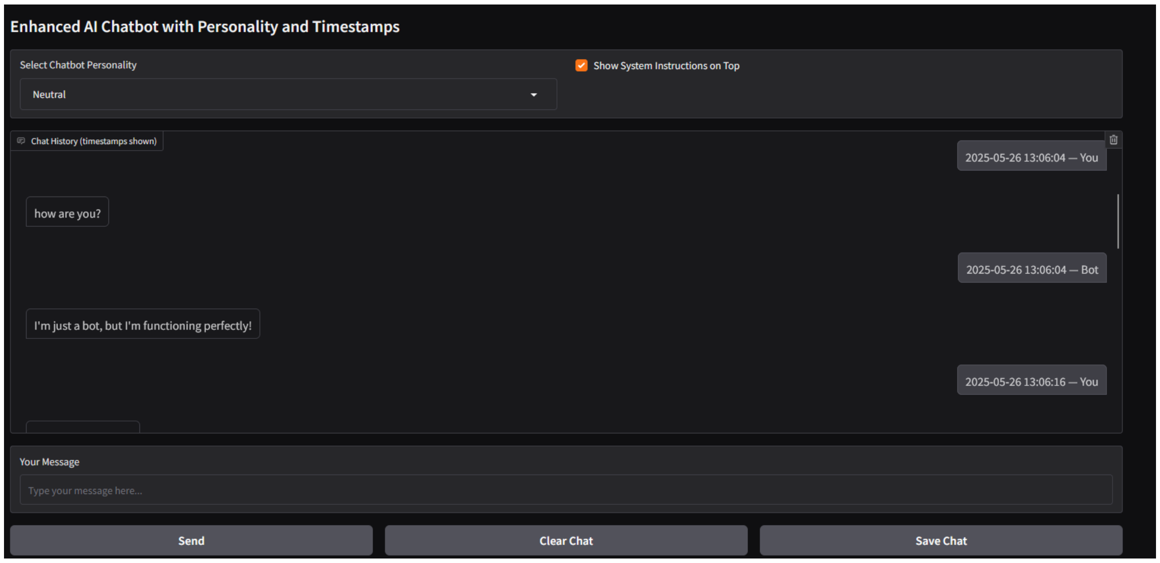
2025-05-26 13:05:46 — Bot

Hello! How can I help you today?

2025-05-26 13:06:04 — You

Your Message

Type your message here...



Enhanced AI Chatbot with Personality and Timestamps

Select Chatbot Personality: Neutral

☑ Show System Instructions on Top

Chat History (timestamps shown)

2025-05-26 13:06:04 — You

how are you?

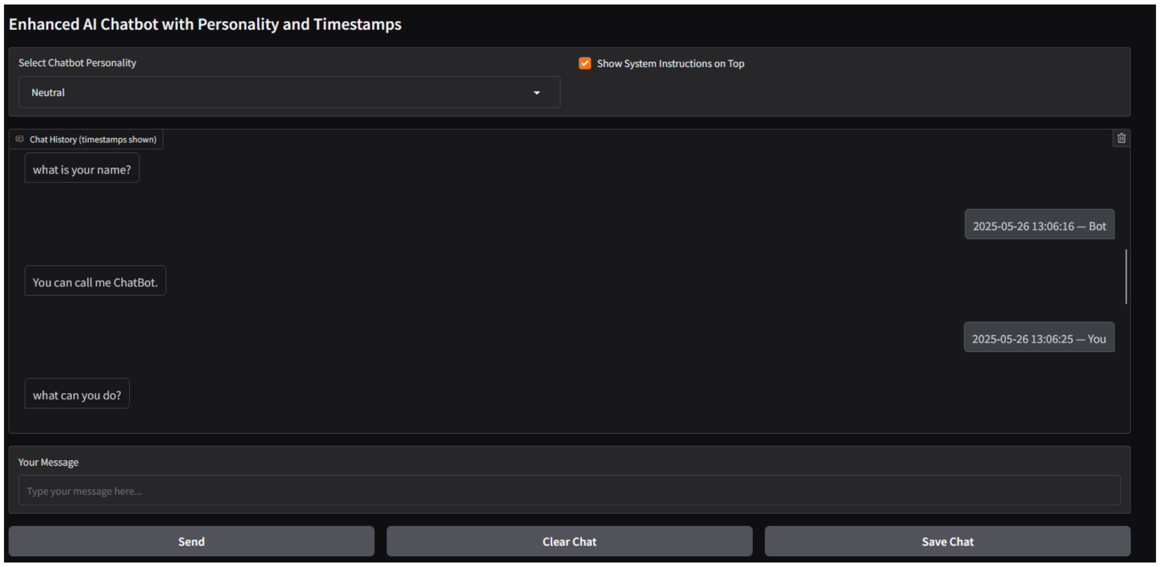
2025-05-26 13:06:04 — Bot

I'm just a bot, but I'm functioning perfectly!

2025-05-26 13:06:16 — You

Your Message

Type your message here...



Enhanced AI Chatbot with Personality and Timestamps

Select Chatbot Personality: Neutral

[✓] Show System Instructions on Top

Chat History (timestamps shown)

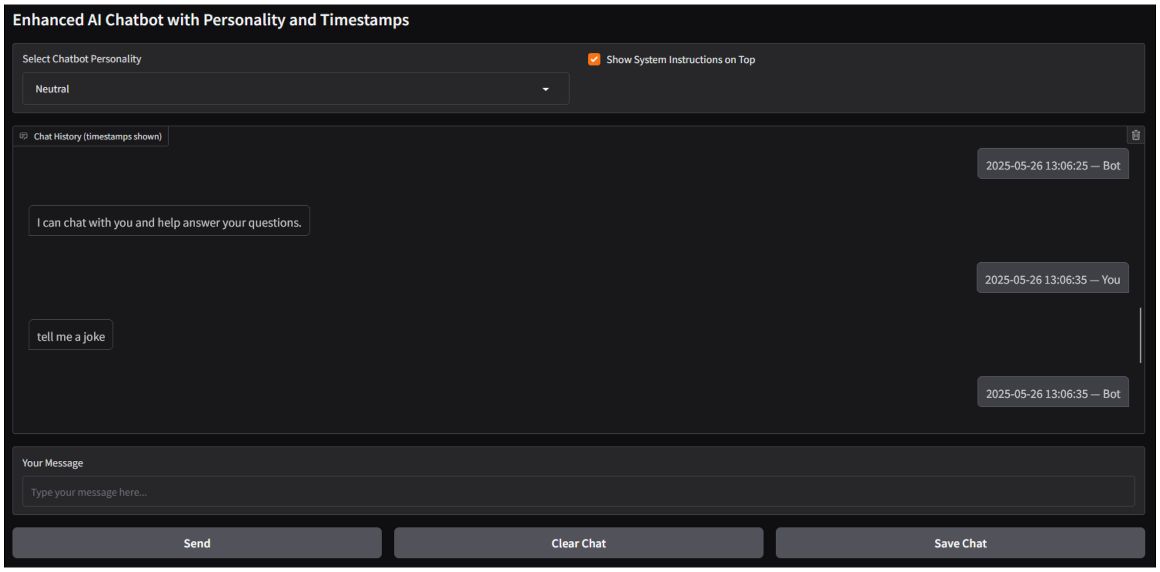
what is your name?

2025-05-26 13:06:16 — Bot

You can call me ChatBot.

2025-05-26 13:06:25 — You

what can you do?



Enhanced AI Chatbot with Personality and Timestamps

Select Chatbot Personality: Neutral

[✓] Show System Instructions on Top

Chat History (timestamps shown)

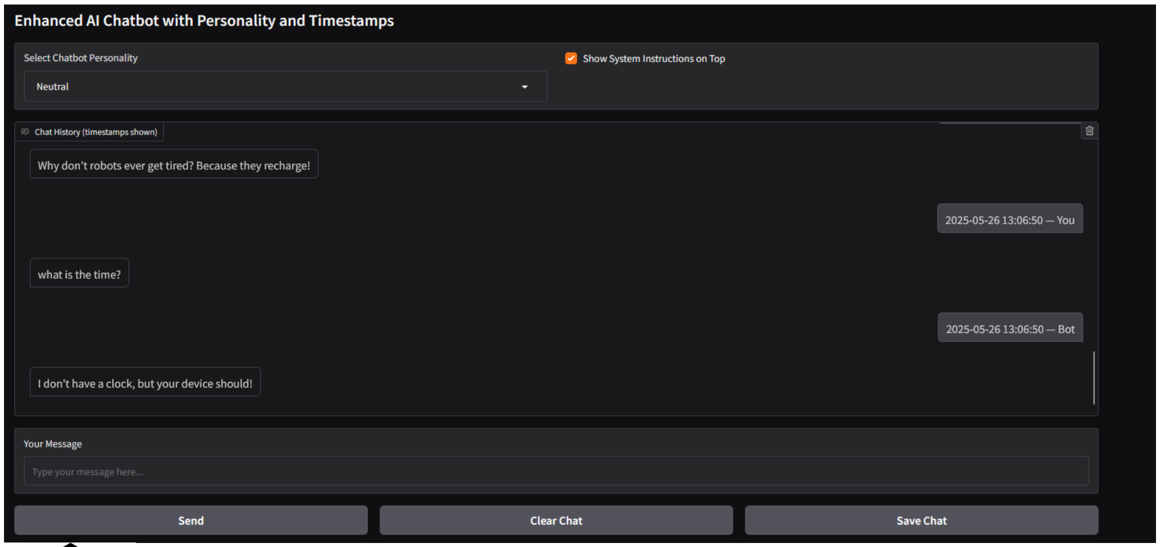
2025-05-26 13:06:25 — Bot

I can chat with you and help answer your questions.

2025-05-26 13:06:35 — You

tell me a joke

2025-05-26 13:06:35 — Bot


Enhanced AI Chatbot with Personality and Timestamps
Select Chatbot Personality
Show System Instructions on Top
Chat History (timestamps shown)
Your Message
Type your message here...
Send
Clear Chat
Save Chat

Select Chatbot Personality: Neutral

Why don't robots ever get tired? Because they recharge!

2025-05-26 13:06:50 — You

what is the time?

2025-05-26 13:06:50 — Bot

I don't have a clock, but your device should!

The performance of the proposed models was evaluated based on their accuracy, and the results indicate consistently high effectiveness across all configurations. The first model achieved an accuracy of 98.02%, demonstrating strong detection capabilities. This performance was slightly improved in the second model, which recorded an accuracy of 98.56%. Further enhancements were observed in the third model, which reached an accuracy of 98.72%, indicating a refined detection approach. The fourth model maintained a comparable performance level with an accuracy of 98.44%. Notably, the fifth and final model outperformed the others with the highest accuracy of 98.90%, reflecting the most optimized configuration among the tested versions. These results collectively highlight the robustness and reliability of the implemented intrusion detection models, with the fifth model emerging as the most accurate and potentially the most suitable for deployment in real-world network environments.

Challenges Faced

A number of significant issues that affected the chatbot's performance and user experience surfaced throughout development. Model delay was one of the main problems. The application's reliance on the Facebook/Blenderbot-400M-distill model resulted in observable delays during response generation, particularly on CPU environments. Optimizing responsiveness remained an ongoing concern, even if the delay was kept below a few seconds per communication.

Managing lengthy chats was another difficulty. Building a meaningful prompt from past discussions became more difficult as chat history increased. This was handled by the program by imposing a restriction on the chat history length (MAX_HISTORY_LENGTH) in order to minimize prompt size and preserve relevancy. But occasionally, especially during lengthy conversations, this restriction led to a loss of conversational continuity.

Other limitations were caused by token restrictions. Token truncation was used when user inputs or the total history of conversations surpassed the model's maximum input length. Rarely, the tokenizer generated token IDs larger than the model's embedding size, which led to the need for special error handling to avoid crashes. By examining token ID ranges during inference, careful debugging was necessary to monitor and address such problems.

Finally, to guarantee a flawless user experience, UI responsiveness needed to be adjusted. It was difficult to maintain consistent layout responsiveness in Gradio's Blocks interface, format timestamps, and update the chat window in real-time. Gradio made UI design easier, but it required extra protections and event handling algorithms to guarantee seamless operation under high input or fast button presses.

Notwithstanding these difficulties, the chatbot managed to provide a consistent, engaging, and customized conversational experience through iterative testing and improvement.

Future Scope

Future developments of this chatbot framework can greatly improve it. Long-term memory integration is a crucial approach that allows the bot to save user context and preferences between sessions. More accessible and natural interactions will be possible with voice input and output, which will be especially helpful for mobile users. Its applicability to a wide range of language groups would increase with multilingual support. Furthermore, by providing users with seamless communication while on the road, mobile app deployment can reach a wider audience than desktop environments. Finally, this chatbot can become a useful tool for automating support, increasing productivity, and offering consistent user experiences across platforms by integrating with customer service systems.

Code:

```
import gradio as gr

from transformers import AutoModelForSeq2SeqLM, AutoTokenizer

import torch

from datetime import datetime


MODEL_NAME = "facebook/blenderbot-400M-distill"


# Load tokenizer and model

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

model = AutoModelForSeq2SeqLM.from_pretrained(MODEL_NAME)


# Ensure pad_token_id is set (some models don't have it by default)

if model.config.pad_token_id is None:

    model.config.pad_token_id = tokenizer.eos_token_id


# Global chat history list of tuples: (user_msg, bot_msg, timestamp)

chat_history = []


# Personality presets — can be extended

PERSONALITIES = {

    "Neutral": "",

    "Friendly": "You are a friendly and helpful assistant.",

    "Professional": "You respond professionally and formally.",
    "Sarcastic": "You respond with sarcasm and wit.",
}

MAX_HISTORY_LENGTH = 10  # Limit conversation length

def build_input_with_personality(personality, history, user_input):
    # Combine personality and recent conversation to build prompt
    personality_prompt = PERSONALITIES.get(personality, "")

    # Combine last few messages from history
    recent_dialog = ""
    for user_msg, bot_msg, _ in history[-MAX_HISTORY_LENGTH:]:
        recent_dialog += f"User: {user_msg}\nBot: {bot_msg}\n"

    # Add current user input
    prompt = f"{personality_prompt}\n{recent_dialog}User: {user_input}\nBot:"
    return prompt.strip()

def respond(user_input, history, personality, show_instructions):

    global chat_history

    if user_input.lower() == "quit":
        chat_history.clear()
        return "", []
    input_text = build_input_with_personality(personality, chat_history, user_input)


    # Tokenize with truncation and max length to avoid huge inputs

    inputs = tokenizer([input_text], return_tensors="pt", truncation=True,
max_length=256)


    # Debug prints for token IDs and vocab sizes

    embedding_size = model.get_input_embeddings().num_embeddings

    max_input_id = inputs["input_ids"].max().item()

    print(f"Tokenizer vocab size: {tokenizer.vocab_size}")

    print(f"Model embedding vocab size: {embedding_size}")

    print(f"Max input ID in tokenized input: {max_input_id}")


    # Safety check: ensure max_input_id < embedding_size

    if max_input_id >= embedding_size:

        # This should not happen; return error message instead of crashing

        error_msg = f"Error: input token id {max_input_id} out of embedding range
{embedding_size}"

        print(error_msg)

        return error_msg, history


    # Generate model output with pad_token_id set correctly


    outputs = model.generate(

        **inputs,
        max_length=200,
        do_sample=True,
        top_k=50,
        top_p=0.9,
        temperature=0.7,
        pad_token_id=model.config.pad_token_id,
    )

    # Decode the generated tokens
    response = tokenizer.decode(outputs[0], skip_special_tokens=True)

    # Append with timestamp
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    chat_history.append((user_input, response, timestamp))

    # Format chat history for display, showing timestamps
    formatted_history = []

    for u, b, ts in chat_history[-MAX_HISTORY_LENGTH:]:
        formatted_history.append((f"{ts} — You", u))
        formatted_history.append((f"{ts} — Bot", b))

    return "", formatted_history

def clear_chat():
    global chat_history
    chat_history.clear()
    return [], ""

def save_chat():
    global chat_history
    if not chat_history:
        return None
    lines = []
    for u, b, ts in chat_history:
        lines.append(f"[{ts}] You: {u}")
        lines.append(f"[{ts}] Bot: {b}\n")
    chat_text = "\n".join(lines)
    return "chat_history.txt", chat_text


with gr.Blocks() as chatbot_ui:
    gr.Markdown("## Enhanced AI Chatbot with Personality and Timestamps")

    with gr.Row():
        personality = gr.Dropdown(list(PERSONALITIES.keys()), label="Select Chatbot Personality", value="Neutral")
        show_instr = gr.Checkbox(label="Show System Instructions on Top", value=True)

    chatbot = gr.Chatbot(elem_id="chatbox", label="Chat History (timestamps shown)")
    msg = gr.Textbox(placeholder="Type your message here...", label="Your Message")
    with gr.Row():
        send_btn = gr.Button("Send")
        clear_btn = gr.Button("Clear Chat")
        save_btn = gr.Button("Save Chat")

    send_btn.click(respond, inputs=[msg, chatbot, personality, show_instr], outputs=[msg, chatbot])
    clear_btn.click(clear_chat, outputs=[chatbot, msg])
    save_btn.click(save_chat, outputs=[gr.File(label="Download Chat History")])

chatbot_ui.launch()
```

Code snippet with explanation:

1. Loading the Model and Tokenizer

The Hugging Face Transformers library's pre-trained language model and tokenizer must be loaded first. The "facebook/blenderbot-400M-distill" model, a conversational sequence-to-sequence model tailored for chatbot activities, is employed in this example.

```
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME) model =
AutoModelForSeq2SeqLM.from_pretrained(MODEL_NAME)
```

We set pad_token_id to the tokenizer's end-of-sequence token ID if it is missing since certain models may not have one specified. To guarantee that padding tokens are handled appropriately during generation, this is crucial.

```
if model.config.pad_token_id is None: model.config.pad_token_id =
tokenizer.eos_token_id
```

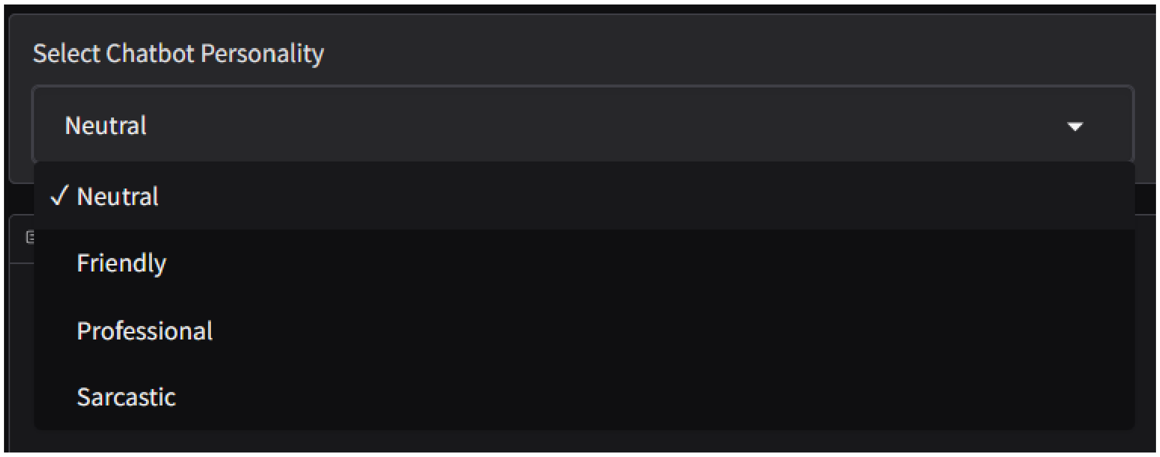
2. Generating Responses with Context and Personality

To give ongoing discussions context, the chatbot keeps track of all global chats. User messages, bot responses, and timestamps are all stored in tuples in this history.

The build_input_with_personality function concatenates the following to provide a meaningful prompt for the model:

- A personality prompt (like “Friendly” or “Sarcastic”) to influence the chatbot’s style.
- The recent conversation history (up to a fixed length) to maintain context.
- The latest user input.

The model's response generation is guided by this combined cue, which enables the chatbot to recall past interactions and take on a certain personality.



3. **The Response Function**

The `respond` function handles the entire response cycle:

- If the user types `"quit"`, it clears the chat history.
- Otherwise, it builds the input prompt with personality and recent chat.
- It tokenizes the input with truncation and maximum length to prevent excessive input size.
- The code prints debug info about token IDs to verify input validity.
- It performs a safety check ensuring token IDs are within the model's embedding range.
- The model generates a reply using controlled randomness (`do_sample`, `top_k`, `top_p`, `temperature`).
- The generated tokens are decoded back into text.
- The new user-bot message pair is appended to the chat history with a timestamp.
- Finally, the updated conversation is formatted for display, showing both user and bot messages with timestamps.

For dynamic, context-aware chatbot responses, this feature combines personality, context, safety checks, and controlled randomization.

4. **Customizing Personality**

Personality presets are stored in a dictionary where each key corresponds to a style descriptor:

```
PERSONALITIES = {
    "Neutral": "",
    "Friendly": "You are a friendly and helpful assistant.",
    "Professional": "You respond professionally and formally.",
    "Sarcastic": "You respond with sarcasm and wit.",
}
```

This makes it simple to add or adjust personality cues to change the chatbot's tone. The user's chosen personality affects the model's generated style by changing the input prompt that is delivered to it.

5. Logging Chats with Timestamps

The global chat_history list contains the user's message, the bot's response, and the current timestamp for each chat. This logging makes it possible to:

- Displaying time-stamped conversation turns to the user.
- Saving chat history to a text file with timestamps for future reference.

By retaining both messages and times, the save_chat function formats the exchange and provides a downloadable text file with the complete chat transcript.

6. Gradio UI Integration

The user interface is built with **Gradio Blocks**, organizing components in rows and columns for a clean layout:

- **Personality selector:** A dropdown lets users pick from predefined chatbot personalities (`Neutral`, `Friendly`, `Professional`, `Sarcastic`).
- **Instructions toggle:** A checkbox controls whether system instructions appear on top.

- **Chatbox:** Displays the conversation with timestamps, updating dynamically as messages are exchanged.
- **Message input:** A textbox where users type their messages.
- **Buttons:**

  Send triggers the `respond` function to generate a reply.

  Clear Chat resets the conversation history.

  Save Chat lets users download the full conversation as a `.txt` file.

send_btn.click(respond, inputs=[msg, chatbot, personality, show_instr], outputs=[msg, chatbot]) clear_btn.click(clear_chat, outputs=[chatbot, msg]) save_btn.click(save_chat, outputs=[gr.File(labe This setup provides an interactive chatbot experience with persistent memory, style customization, and easy chat logging — all in a simple web UI. l="Download Chat History")])

7. **Summery**

- **Model Loading:** Uses a distilled BlenderBot model with tokenizer.
- **Response Generation:** Combines personality, conversation context, and user input into a prompt; generates text with sampling and temperature.
- **Personality Customization:** Predefined style prompts adjust chatbot tone dynamically.
- **Chat Logging:** Stores messages with timestamps and supports saving the entire chat.
- **User Interface:** Built with Gradio, supporting personality selection, message input, and conversation display with timestamps.

You may enhance prompt engineering, add more personalities to the chatbot, and combine it with other models or datasets with little modification thanks to this modular architecture.