

PROJECT TITLE: **CREATE A CHATBOT IN PYTHON**

PROBLEM STATEMENT:

The challenge is to create a chatbot in 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.

DATASET LINK: <https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot>

INTRODUCTION:

The project will explore the fundamentals of chatbot development, including the essential technologies and techniques that power these conversational AI entities. From natural language processing and machine learning to user interface design and ethical considerations, the chatbot development will equip with the knowledge and tools to create intelligent and engaging chatbot solutions.

DESIGN THINKING:

1. FUNCTIONALITY:

The core functionality of the chatbot is to engage in casual conversations with users. It is capable of responding to general questions, greetings, and small talk. It can give some jokes and can even comfort the users in a wider view.

2. USER INTERFACE:

The user interface of the chatbot includes a webpage which allows the users to interact with the bot. It is created to provide user friendly environment which is designed to receive user inputs and provide the response from the pre-trained model.

3. NATURAL LANGUAGE PROCESSING(NLP):

- *Tokenization*: The dataset is tokenized that is break into individual words, phrases for analysis.
- *AutoTokenizer*: It is a part of the Hugging Face Transformers library, which is a popular open-source library for Natural Language Processing (NLP) tasks, including pre-trained models and tokenization.
- *Spacy*: It is implemented to design the chatbot to perform wide range of NLP tasks efficiently and with high accuracy.

4. RESPONSES:

The chatbot can handle the queries that includes chit-chat and greetings. It can plan responses for common errors or misunderstandings. If the chatbot doesn't understand a query, it should respond with helpful suggestions or ask for clarification. For example;

User: "Tell me a joke about elephants."

Chatbot: "I'm sorry, I don't have any jokes about elephants. How about a joke about cats instead?"

5. INTEGRATION:

Integrating the chatbot with a Flask web application involves setting up a communication mechanism between the web interface and the chatbot's logic.

6. TESTING AND IMPROVEMENT:

The chatbot can improve itself by considering the information provided in the same chat page. It ensure that the chatbot maintains a context of the ongoing conversation. This involves keeping track of previous user messages and chat history to understand the current conversation's context.

PHASES OF DEVELOPMENT:

PHASE 1: In this phase, the problem statement of the chatbot is considered and design thinking process of the problem is developed which includes functionality, user interface, NLP, responses, integration and, testing and improvement of the chatbot.




PHASE 2: In this phase, advanced techniques like using pre-trained language models (e.g., GPT-3) is explored to enhance the quality of responses.

PHASE 3: In this phase, the chatbot is built by preparing the environment and implementing basic user interactions.

PHASE 4: The chatbot is built by integrating it into a web app using flask.

PHASE 5: Documenting the process of the chatbot.

LIBRARIES USED:

-  PANDAS – Used for data manipulation and analysis
-  SPACY – Used for understanding and processing user input in a conversational context.
-  TORCH - implementing deep learning-based chatbots

- ✚ TRANSFORMERS - provides pre-trained transformer-based models and tools that enable chatbots to understand and generate human-like text
- ✚ FLASK - provides the necessary infrastructure for creating web-based chatbot applications, serving as a communication interface between the chatbot and users
- ✚ RENDER TEMPLATE - to display content and responses in a user-friendly manner
- ✚ REQUEST - allows your chatbot to interact with external data sources, retrieve information, and perform various actions
- ✚ AUTO TOKENIZER - simplifies the process of tokenizing text data for use with transformer-based models

INTEGRATION OF NLP TECHNIQUES:

The basic text processing in NLP:

- ✓ Sentence Segmentation: This is a fundamental step in many NLP applications because it helps in breaking down the text into smaller units for analysis, processing, and further understanding. Sentence segmentation can be a non-trivial task because sentences in natural language text can have various structures, making it challenging to determine where one sentence ends and the next begins.
- ✓ Lowercasing: Lowercasing involves converting all text to lowercase letters. This is a common pre-processing step to ensure consistency in text data because it treats words like "word," "Word," and "WORD" as the same word. Lowercasing is particularly useful for tasks like text classification, where the case of the letters may not be relevant.
- ✓ Removing Alphanumeric Characters: Removing alphanumeric characters involves eliminating digits (numbers) and special characters (e.g., punctuation, symbols) from text data. This step is often performed to focus on the textual content itself and remove noise. It can be helpful when you want to analyze the words and their relationships rather than numbers or symbols.

```

import string
import re
# importing regular expressions
punc_lower = lambda x: re.sub('[%s]' % re.escape(string.punctuation), ' ', x.lower())
# Lower case conversion
remove_n = lambda x: re.sub("\n", " ", x)
# removing \n and replacing them with empty value
remove_non_ascii = lambda x: re.sub(r'^\x00-\x7f', r' ', x)
# removing non ascii characters
alphanumeric = lambda x: re.sub('\w*\d\w*', ' ', x)
# removing alpha numeric values
df['User'] = df['User'].map(alphanumeric).map(punc_lower).map(remove_n).map(remove_non_ascii)
# using map function and applying the function on query column
df['Chatbot'] = df['Chatbot'].map(alphanumeric).map(punc_lower).map(remove_n).map(remove_non_ascii)
# using map function and applying the function on response column
print(df)
df.to_csv('modified_dataset.csv', index=False)

```

- ✓ Normalization: The goal of normalization is to reduce text variations, making it easier to process, analyze, and compare textual information. Normalization is essential in many NLP tasks to ensure that the text is consistent and uniform, as it can significantly impact the performance of algorithms and models.
- ✓ Tokenization: Tokenization is a fundamental natural language processing (NLP) task that involves breaking down a text or document into smaller units called tokens. These tokens are typically words, phrases, or symbols, and the goal of tokenization is to facilitate the analysis and processing of text data by splitting it into manageable units.

```

[ ] vocabulary = {"<PAD>": 0, "<UNK>": 1} # Initialize with special tokens
with open("modified_dataset.csv", "r") as file:
    for line in file:
        words = line.strip().split() # Split by whitespace for adapting the data format
        for word in words:
            if word not in vocabulary:
                vocabulary[word] = len(vocabulary)

```

- ✓ Stop word Removal: Stop word removal is a common text pre-processing technique in natural language processing (NLP) that involves eliminating words that are considered common and non-informative from a text or document. These "stop words" are often excluded from analysis because they do not contribute significantly to the meaning of the text and may add noise to NLP tasks. The list of stop words typically includes common words such as "the," "and," "is," "in," "to," and other frequently occurring words.

CHATBOT INTERACTION WITH USER AND WEB APPLICATION:

The interaction between a chatbot and a web application, such as a Flask-based web app, involves a series of steps where the chatbot processes user queries and provides responses within the context of the web application. Overview of how this interaction works:

User Interaction:

A user visits the web application and initiates interaction with the chatbot, typically through a chat interface on the web page. The chat interface allows users to type or speak their queries.

User Input Processing:

When the user submits a query or message in the chat interface, the web application captures the input. This input is then passed to the chatbot for processing.

Chatbot Integration:

The web application is integrated with the chatbot's logic and NLP capabilities. This integration allows the web application to send user messages to the chatbot and receive responses.

NLP Processing:

The chatbot employs Natural Language Processing (NLP) techniques to understand the user's input. This may include tokenization, part-of-speech tagging, entity recognition, intent recognition, and sentiment analysis. NLP helps the chatbot interpret the user's query effectively.

Response Generation:

Based on the NLP analysis of the user input, the chatbot generates a response. This response provides text that is relevant to the user's query.

Dynamic Responses:

The chatbot may generate dynamic responses based on the user's query and the current context of the conversation. For example, if the user asks for a weather update, the chatbot can fetch real-time weather data and provide an up-to-date response.

Response Delivery:

The chatbot sends the response back to the web application, which, in turn, displays the response in the chat interface on the web page. The response is presented to the user in a user-friendly format.

Real-Time Interaction:

The conversation between the user and the chatbot can be conducted in real-time. This means that as the user and chatbot exchange messages, responses are displayed immediately within the chat interface, creating a seamless conversation experience.

Multi-Turn Conversations:

Users can continue to interact with the chatbot, asking follow-up questions or providing additional information. The chatbot maintains the conversation context to ensure coherent responses.

User Engagement:

The chatbot aims to engage the user effectively by providing helpful and accurate responses, assisting with tasks, and offering information or guidance.

Error Handling:

The chatbot may implement error-handling mechanisms to address situations where it doesn't understand the user query or encounters problems during the interaction. In such cases, the chatbot can ask for clarification or suggest alternative actions.

Continuous Improvement:

The chatbot and web application can gather feedback from users, track user interactions, and analyze the performance of the chatbot. This data is used to continuously refine the chatbot's responses and enhance its capabilities.

The interaction between the chatbot and the web application is a dynamic and iterative process that aims to provide a positive user experience and deliver valuable assistance or information to users in real-time. The integration of NLP techniques and dynamic response generation is essential for effective user engagement.

INNOVATIVE TECHNIQUES:

Transformer Models:

Transformer-based models like Dialo-GPT have revolutionized chatbot development. These models are capable of understanding context and generating human-like text, making chatbots more conversational and context-aware.

Transfer Learning:

Chatbots can leverage transfer learning by fine-tuning pre-trained models on specific tasks or domains. This approach reduces the amount of data required for training and improves performance.

Conversational Memory:

Chatbots with conversational memory store and recall previous interactions, providing a more personalized and context-aware user experience.

Fine-Tuning:

Depending on your specific chatbot use case, consider fine-tuning the DialogPT model on domain-specific data. This helps the model better understand and generate content related to your particular industry or field.

User Simulations:

Create user simulations or dialogue datasets to train your chatbot. This helps improve the quality of responses and ensures that the model can handle various user inputs effectively.

Flask integration:

Integrating a chatbot with Flask involves using Flask as the backend for your chatbot application.

```
tokenizer = AutoTokenizer.from_pretrained("microsoft/DialogPT-medium")
model = AutoModelForCausalLM.from_pretrained("chatbot_model")

app = Flask(__name__)

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

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

GETTING RESPONSE FROM MODEL:

```
def get_Chat_response(text):
    for step in range(5):
        # 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 = model.generate(bot_input_ids, max_length=1000, pad_token_id=tokenizer.eos_token_id)

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

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

APPROACHES USED DURING DEVELOPMENT:

Machine Learning-Based Chatbots:

Machine learning-based chatbots use techniques like natural language processing (NLP) and machine learning to understand user input and generate responses. They can handle a wider range of user queries compared to rule-based chatbots.

Generative Chatbots:

Generative chatbots use language models like Dialo-GPT is used to generate human-like text responses. These chatbots can engage in natural conversations and provide context-aware responses.

Retrieval-Based Chatbots:

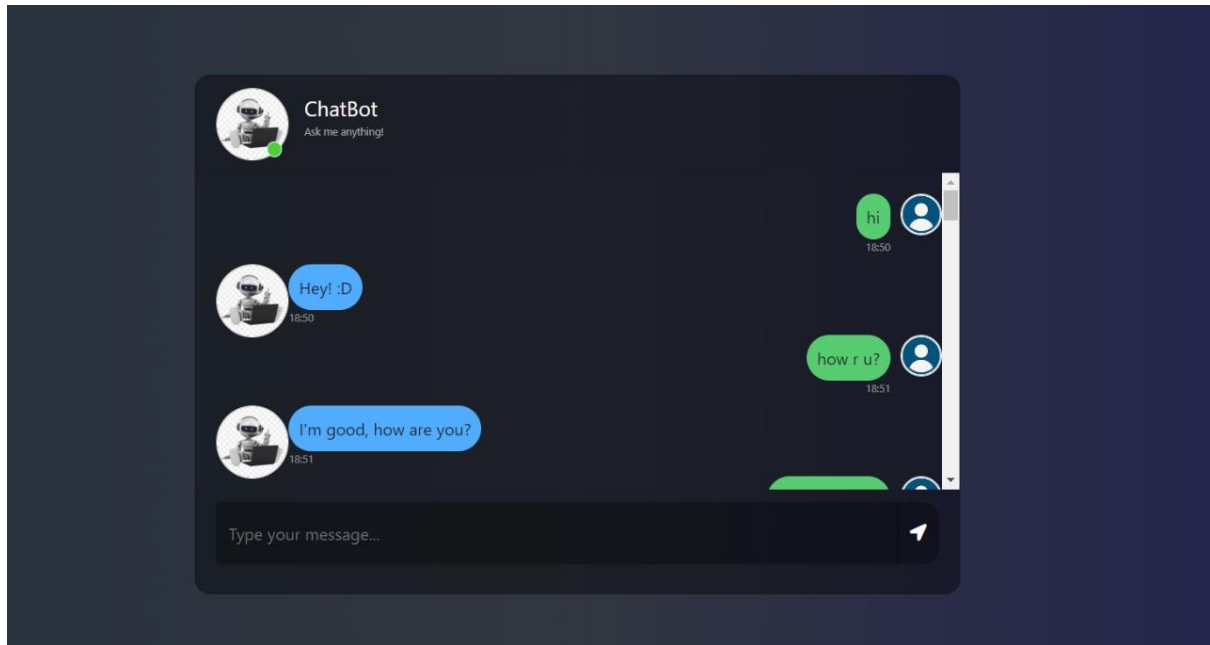
Retrieval-based chatbots select responses from predefined sets based on user input. They match user queries with the closest predefined response and are useful for FAQ or knowledge-based chatbots.

Conversational Agents:

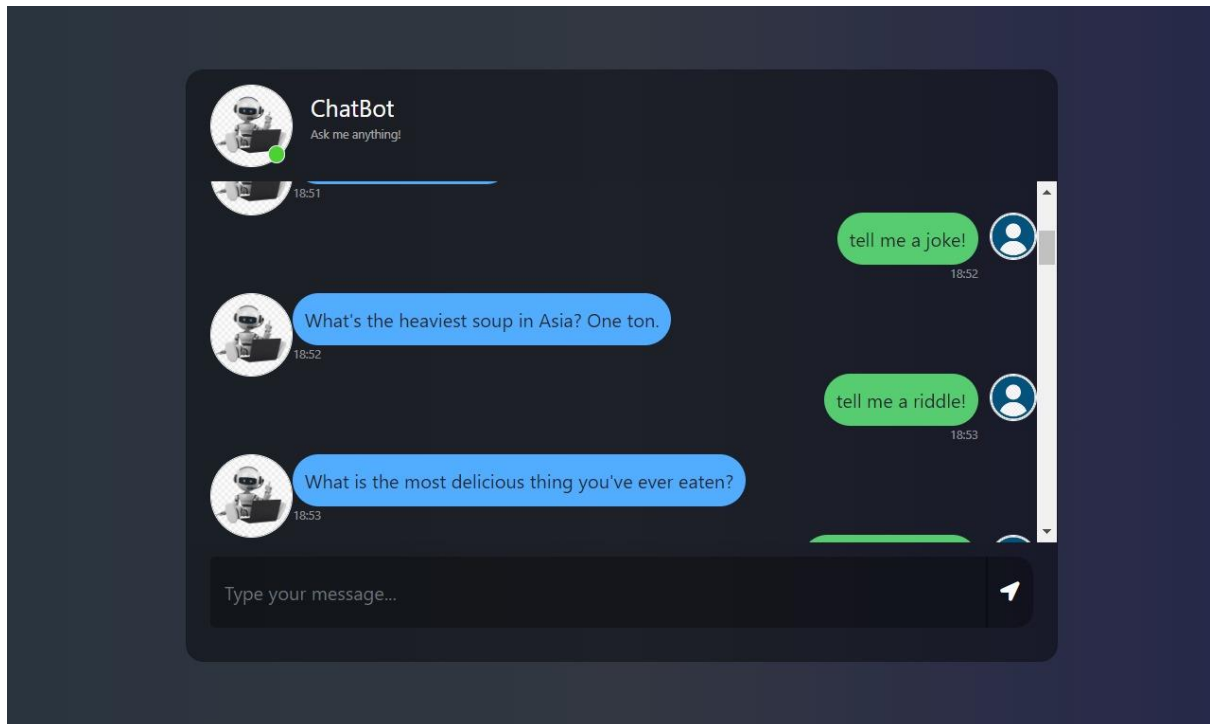
Conversational agents aim to create a human-like conversation experience by simulating human interactions and personalities. It actually behaves like a human while responding to a query raised.

BASIC USER INTERACTION WITH THE CHATBOT:

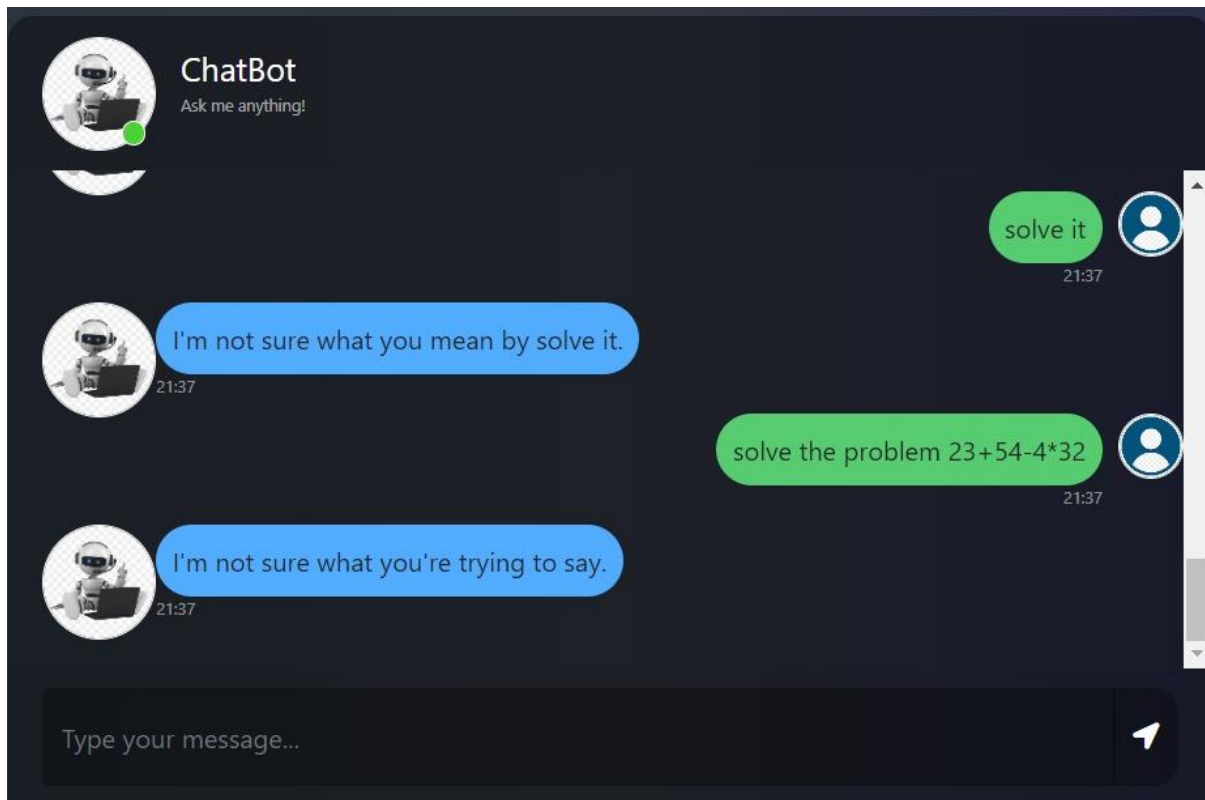
GREETINGS:



ENTERTAINMENT:



ERROR HANDLING:



The developed chatbot contains the trained model “chatbot_model” which acts as the trained dataset and the pre-trained model “Dialo-GPT” developed by Microsoft for easier interaction with the user and the bot.

The chatbot can provide easier user interaction interface, greetings, entertainment (eg: riddles and jokes), can act as human intelligence, error handling and information sharing.

HTML FILE:

A hyper-text markup language (HTML) is developed for the front end integration of the chatbot with the web application. It helps the user to experience active chatbot usage.

CONCLUSION:

The chatbot developed is a basic user interaction made to have conversation with the bot in a web application by the integration of the flask. It includes greetings, entertainment (example: riddles and jokes), acting as human intelligence, error handling and information sharing. The model built has the capability of adapting to the generating texts and provides the user with better experience.

