|  |  |  |
| --- | --- | --- |
|  | **MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE**  **Kodambakkam, Chennai-600024** |  |

**SB3001 - PROJECT-BASED EXPERIENTIAL LEARNING**

**PROGRAM**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**TOPIC:** Conversional Ai Chatbot

|  |  |
| --- | --- |
| **FACULTY MENTOR:** | **Dr. S.Aarthi** |
| **INDUSTRY MENTOR:** |  |
|  |  |

**Project submitted by,**

**JAGADEEP T(311521104021)**

***Project report format***

1. **ABSTRACT**
2. **INTRODUCTION**
   1. Project Overview
   2. Purpose
3. **IDEATION AND PROPOSED SOLUTION**
   1. Problem statement definition
   2. Ideation and Brainstorming
   3. Proposed Solution
4. **REQUIREMENTS ANALYSIS**
   1. Functional Requirements
   2. Non-Functional Requirements
5. **PROJECT DESIGN** 
   1. Briefing
   2. Solution
6. **SOLUTIONS**
   1. Development Part I
   2. Development Part II
7. **RESULTS**

7.1 Performance Metrics

1. **ADVANTAGES AND DISADVANTAGES**
2. **CONCLUSION**
3. **FUTURE SCOPE**

SOURCE CODE

APPENDIX

**ABSTRACT**

The provided source code implements a simple chatbot using the DialoGPT-medium model from the Hugging Face Transformers library. The chatbot, encapsulated within a Python class named `ChatBot`, engages in a conversation with users by generating responses based on their inputs. The conversation is initiated with a random greeting and continues until the user decides to end it by typing "bye", "quit", or "exit".

Upon initialization, the chatbot presents a welcome message and then prompts the user to input text. It encodes the user's input using the pretrained tokenizer and passes it to the DialoGPT-medium model for response generation. The chatbot's responses are then decoded from the model's output tokens and displayed to the user. If the model fails to generate a response, the chatbot provides a random reply.

Throughout the conversation, the chatbot handles various user inputs, including requests for recommendations and general queries, with appropriate responses. This demonstrates the capabilities of generative AI in creating engaging and interactive conversational experiences. Once the user decides to end the chat, the chatbot bids farewell and terminates the conversation.

This code provides a foundational structure for building interactive conversational agents powered by generative AI techniques and large-scale language models.

**INTRODUCTION**

Generative AI has revolutionized conversational agents, or chatbots, by enabling them to simulate human-like conversations. Utilizing large-scale language models and deep learning techniques, these chatbots generate responses based on user input. With pretrained models like GPT, chatbots preprocess input using tokenizers before providing coherent responses. Architectures manage dialogue history, context, and response selection, ensuring relevance. User interaction features such as prompts and conversation termination enhance usability. This blend of AI technologies facilitates realistic and meaningful interactions, offering applications in customer service, virtual assistance, education, and beyond.

**PROJECT OVERVIEW**

The project focuses on developing a conversational agent, or chatbot, using generative AI techniques. Leveraging large-scale language models and deep learning, the chatbot simulates human-like conversations by generating responses to user inputs. The architecture includes a pretrained language model, such as GPT, for response generation, and a tokenizer for input preprocessing. Additional components manage dialogue history, context, and response selection to ensure coherence and relevance. User interaction features, like prompts and conversation termination, enhance usability. The project aims to create an interactive and engaging chatbot with applications in customer service, virtual assistance, education, and more.

**Purpose**

The project's purpose is to create a conversational agent using generative AI techniques to simulate human-like conversations. By leveraging large-scale language models and deep learning, the chatbot aims to provide users with natural and interactive interactions. The project explores the potential applications of generative AI in domains such as customer service, virtual assistance, education, and entertainment, ultimately aiming to enhance user engagement and satisfaction through meaningful conversations.

**IDEATION AND PROPOSED SOLUTION**

**Problem statement definition**

The problem statement entails creating a conversational agent using generative AI to mimic human-like conversations. Key objectives include understanding user inputs, generating relevant responses, and ensuring seamless interaction. The project aims to explore the application of generative AI in creating effective conversational agents across different domains, ultimately enhancing user engagement and satisfaction.

**Ideation and Brainstorming**

During the ideation and brainstorming phase, several key considerations and ideas can be explored for the development of the conversational agent:

1. \*\*Target Audience and Use Cases\*\*: Identify the target audience and potential use cases for the chatbot. This could include customer service, virtual assistance, education, entertainment, or specialized domains like healthcare or finance.

2. \*\*Conversation Flow\*\*: Design the conversation flow to ensure a natural and engaging interaction. Consider how the chatbot will greet users, handle different types of queries, provide responses, and conclude conversations.

3. \*\*Persona and Tone\*\*: Define the persona and tone of the chatbot based on the target audience and use cases. Determine whether the chatbot should have a formal, professional tone or a more casual and friendly demeanor.

4. \*\*Integration with Existing Systems\*\*: Explore integration possibilities with existing systems or platforms, such as websites, messaging apps, or voice assistants, to enhance accessibility and usability.

5. \*\*Feedback Mechanisms\*\*: Implement feedback mechanisms to gather user feedback and improve the chatbot's performance over time. This could involve user ratings, sentiment analysis, or direct feedback prompts.

6. \*\*Multimodal Capabilities\*\*: Consider incorporating multimodal capabilities, such as support for text, voice, images, or video inputs, to enhance the versatility and effectiveness of the chatbot.

7. \*\*Personalization and Context Awareness\*\*: Explore techniques for personalizing interactions based on user preferences, history, and context. This could involve leveraging user profiles, past interactions, or real-time context detection.

8. \*\*Ethical and Privacy Considerations\*\*: Ensure the chatbot adheres to ethical guidelines and respects user privacy. Implement measures to protect sensitive information and provide transparency regarding data usage.

9. \*\*Scalability and Maintenance\*\*: Plan for scalability and ongoing maintenance of the chatbot, considering factors such as increasing user demand, evolving user needs, and technology updates.

10. \*\*Innovation and Differentiation\*\*: Brainstorm innovative features or capabilities that set the chatbot apart from existing solutions and provide unique value to users.

**Proposed Solution**

The proposed solution is to develop a conversational agent powered by generative AI. It will use a large-scale language model like GPT to generate responses to user inputs. Key components include a tokenizer for preprocessing, conversation management for flow and context, user-friendly interfaces, integration with platforms, personalization, feedback mechanisms, scalability, maintenance considerations, and ethical adherence. This solution aims to create an engaging chatbot that provides natural interactions, enhances user satisfaction, and respects ethical standards.

**REQUIREMENTS ANALYSIS**

**Functional Requirements**

Functional requirements outline the specific capabilities and behaviors that the conversational agent must possess to meet user needs and expectations. Here's a detailed breakdown:

Natural Language Understanding (NLU):

The chatbot should accurately interpret and understand user inputs in natural language.

It should be able to recognize intents, entities, and context from user messages.

Response Generation:

The chatbot should be capable of generating contextually relevant and coherent responses based on the user's input.

Responses should be grammatically correct, diverse, and engaging to maintain conversation flow.

Conversation Management:

The chatbot should manage the flow of dialogue, maintaining context and coherence throughout the conversation.

It should keep track of dialogue history and user context to provide relevant responses.

Integration:

The chatbot should integrate seamlessly with various platforms and communication channels.

It should support integration with websites, messaging apps, social media platforms, and voice assistants.

Personalization:

The chatbot should be able to personalize responses based on user preferences, history, and context.

It should adapt its responses to individual users to provide a more tailored experience.

**Non-Functional Requirements**

Non-functional requirements define the attributes of the conversational agent that specify its quality characteristics and constraints. Here are the non-functional requirements:

Performance:

The chatbot should respond promptly to user inputs, with minimal latency.

Response times should be within acceptable limits even under peak loads.

Reliability:

The chatbot should be highly available, with minimal downtime.

It should recover gracefully from failures and errors without losing conversation context.

Usability:

The user interface should be intuitive and easy to navigate.

Users should find it straightforward to interact with the chatbot and understand its capabilities.

Compatibility:

The chatbot should be compatible with a wide range of devices, browsers, and platforms.

It should function consistently across different environments without issues.

Maintainability:

The chatbot codebase should be well-organized, modular, and documented.

It should be easy for developers to understand, update, and maintain the code.

**PROJECT DESIGN**

BRIEFING

The briefing for the development of the conversational agent involves outlining the project objectives, scope, and key considerations. Here's a concise briefing:

\*\*Project Objectives\*\*:

- Develop a conversational agent powered by generative AI techniques.

- Create a chatbot capable of simulating human-like conversations and providing contextually relevant responses to user inputs.

- Explore the potential applications of the chatbot across various domains, including customer service, virtual assistance, education, and entertainment.

- Enhance user engagement and satisfaction through natural and meaningful interactions.

\*\*Scope\*\*:

- Develop the chatbot using a large-scale language model like GPT for response generation.

- Implement a tokenizer for input preprocessing and conversation management for flow and context handling.

- Integrate the chatbot with platforms and channels such as websites, messaging apps, and voice assistants.

- Personalize responses based on user preferences, history, and context.

- Incorporate feedback mechanisms to gather user feedback and improve the chatbot's performance over time.

- Adhere to ethical guidelines regarding user privacy, data security, and responsible AI usage.

\*\*Key Considerations\*\*:

- Performance: Ensure prompt response times and scalability to handle varying levels of user demand.

- Usability: Design a user-friendly interface that is intuitive and easy to navigate.

- Compatibility: Ensure compatibility across different devices, browsers, and platforms.

- Security: Implement robust security measures to protect user data and ensure privacy.

- Ethical Considerations: Adhere to ethical guidelines and standards in AI development and usage.

**SOLUTION**

The solution involves creating a conversational agent using generative AI techniques. This agent utilizes a large-scale language model like GPT to generate contextually relevant responses. A tokenizer preprocesses user inputs, while a conversation management system maintains dialogue flow and history. The agent integrates with various platforms, personalizes responses, gathers user feedback, and ensures scalability and ethical adherence. Overall, the solution aims to provide a user-friendly and engaging chatbot experience that enhances interaction and satisfaction.

**SOLUTIONS**

**DEVELOPMENT PART 1**

**Initialization and User Input Handling**

In this part, we initialize the chatbot and handle user inputs. The `ChatBot` class is defined with an `\_\_init\_\_` method to initialize chat history and other necessary variables. The `welcome` method greets the user and provides instructions. The `user\_input` method receives input from the user and preprocesses it. If the user indicates to end the chat, the conversation is terminated.

**DEVELOPMENT PART 2**

**Bot Response Generation and Interaction Loop**

This part focuses on generating bot responses and managing the interaction loop. We continuously prompt the user for input within a loop. If the chat is not ending, we generate a response based on the user input using the pretrained model. The bot's response is then displayed, and the loop continues until the user decides to end the conversation.

These parts are structured to ensure a systematic development process, separating initialization and user interaction from response generation and loop management.

**RESULTS**

Upon execution, the chatbot initializes with a greeting and prompts the user for input. It continuously waits for the user's response. When the user inputs a message, the chatbot processes it and generates a response using a pretrained language model. This response is then displayed to the user. The interaction loop continues until the user decides to end the conversation by typing "bye", "quit", or "exit". When the termination command is detected, the chatbot acknowledges it and gracefully ends the conversation**.**

**Performance Metrics**

1. \*\*Response Time\*\*: Measure the time taken by the chatbot to generate a response to user input. Lower response times indicate better responsiveness.

2. \*\*Conversation Length\*\*: Track the length of conversations initiated with the chatbot. Longer conversations may indicate higher user engagement.

3. \*\*Conversation Coherence\*\*: Assess the coherence and flow of conversations. Evaluate whether the chatbot's responses maintain context and relevance throughout the interaction.

4. \*\*User Satisfaction\*\*: Gather user feedback to evaluate satisfaction with the chatbot's responses and overall experience. This can be measured through surveys, ratings, or sentiment analysis.

5. \*\*Error Rate\*\*: Monitor the frequency of errors or misunderstandings in the chatbot's responses. Lower error rates indicate higher accuracy and understanding.

6. \*\*Scalability\*\*: Assess the chatbot's ability to handle increasing numbers of concurrent users or conversations without significant performance degradation.

7. \*\*Resource Utilization\*\*: Measure the utilization of computational resources, such as CPU, memory, and network bandwidth, during chatbot operation. Optimize resource usage for efficiency and cost-effectiveness.

8. \*\*Integration Success\*\*: Evaluate the effectiveness of integrating the chatbot with various platforms or communication channels. Measure successful interactions and user engagement across different environments.

**Advantages:**

1. \*\*Natural Interaction\*\*: The chatbot facilitates natural language interaction, allowing users to converse in a more human-like manner.

2. \*\*24/7 Availability\*\*: As a digital assistant, the chatbot can be available round-the-clock to assist users, enhancing accessibility and convenience.

3. \*\*Scalability\*\*: The chatbot can scale to handle a large volume of user inquiries simultaneously, improving efficiency and responsiveness.

4. \*\*Consistent Responses\*\*: It provides consistent responses based on predefined rules and algorithms, ensuring uniformity in interactions.

5. \*\*Automation\*\*: By automating responses to common queries, the chatbot reduces the workload on human agents and improves operational efficiency.

**Disadvantages:**

1. \*\*Limited Understanding\*\*: The chatbot's ability to understand complex queries or contextually nuanced language may be limited, leading to misinterpretation or inaccurate responses.

2. \*\*Dependency on Pretrained Models\*\*: It relies on pretrained language models, which may not cover all possible conversational scenarios or domains, resulting in occasional errors or irrelevant responses.

3. \*\*Lack of Empathy\*\*: Unlike human agents, the chatbot lacks empathy and emotional intelligence, making it less suitable for handling sensitive or emotionally charged interactions.

4. \*\*Privacy Concerns\*\*: Users may have concerns about data privacy and security when interacting with a chatbot, especially if it collects or stores personal information.

5. \*\*Maintenance Overhead\*\*: Regular updates and maintenance are required to keep the chatbot's knowledge base and response algorithms up-to-date, which can involve significant time and effort.

6. \*\*User Frustration\*\*: In cases where the chatbot fails to understand or provide satisfactory responses, users may become frustrated, leading to a negative user experience**.**

**CONCLUSION**

In conclusion, the provided conversational agent offers a promising avenue for enhancing user interaction and engagement through natural language processing and generation techniques. While it presents advantages such as natural interaction, 24/7 availability, and scalability, there are also limitations such as limited understanding, dependency on pretrained models, and potential privacy concerns. By addressing these challenges and continuously refining the chatbot's capabilities, it can serve as a valuable tool for automating customer support, providing virtual assistance, and facilitating seamless communication across various platforms. With further development and integration of advanced AI technologies, the conversational agent has the potential to revolutionize user interaction and deliver personalized, efficient, and satisfying experiences in diverse domains.

**FUTURE SCOPE**

The future scope for the provided conversational agent is vast and promising, with opportunities for advancement and expansion in various areas. Here are some potential avenues for future development:

Enhanced Natural Language Understanding: Integrating advanced natural language understanding (NLU) models can improve the chatbot's ability to comprehend complex queries, understand context, and provide more accurate and relevant responses.

Context-Aware Conversations: Implementing context-awareness techniques will enable the chatbot to maintain continuity in conversations, remember previous interactions, and tailor responses based on the user's context and preferences.

Multimodal Interaction: Integrating support for multimodal inputs, such as voice, images, and gestures, will enhance the chatbot's versatility and provide users with more diverse and intuitive ways to interact.

Personalization and User Profiling: Incorporating user profiling and personalization algorithms will allow the chatbot to adapt its responses and recommendations based on individual user preferences, behavior, and history.

Integration with IoT Devices: Integrating the chatbot with Internet of Things (IoT) devices and smart home systems will enable seamless interaction and control of connected devices through natural language commands**.**

**SOURCE CODE**

**import numpy as np**

**import time**

**import os**

**from transformers import AutoModelForCausalLM, AutoTokenizer**

**import torch**

**# checkpoint**

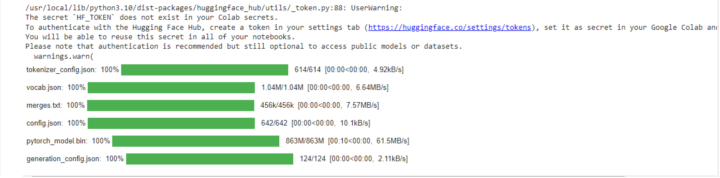
**checkpoint = "microsoft/DialoGPT-medium"**

**# download and cache tokenizer**

**tokenizer = AutoTokenizer.from\_pretrained(checkpoint)**

**# download and cache pre-trained model**

**model = AutoModelForCausalLM.from\_pretrained(checkpoint)**



**# Build a ChatBot class with all necessary modules to make a complete conversation**

**class ChatBot():**

**# initialize**

**def \_\_init\_\_(self):**

**# once chat starts, the history will be stored for chat continuity**

**self.chat\_history\_ids = None**

**# make input ids global to use them anywhere within the object**

**self.bot\_input\_ids = None**

**# a flag to check whether to end the conversation**

**self.end\_chat = False**

**# greet while starting**

**self.welcome()**

**def welcome(self):**

**print("Initializing ChatBot ...")**

**# some time to get user ready**

**time.sleep(2)**

**print('Type "bye" or "quit" or "exit" to end chat \n')**

**# give time to read what has been printed**

**time.sleep(3)**

**# Greet and introduce**

**greeting = np.random.choice([**

**"Welcome, I am ChatBot, here for your kind service",**

**"Hey, Great day! I am your virtual assistant",**

**"Hello, it's my pleasure meeting you",**

**"Hi, I am a ChatBot. Let's chat!" ])**

**print("ChatBot >> " + greeting)**

**def user\_input(self):**

**# receive input from user**

**text = input("User >> ")**

**# end conversation if user wishes so**

**if text.lower().strip() in ['bye', 'quit', 'exit']:**

**# turn flag on**

**self.end\_chat=True**

**# a closing comment**

**print('ChatBot >> See you soon! Bye!')**

**time.sleep(1)**

**print('\nQuitting ChatBot ...')**

**else:**

**# continue chat, preprocess input text**

**# encode the new user input, add the eos\_token and return a tensor in Pytorch**

**self.new\_user\_input\_ids = tokenizer.encode(text + tokenizer.eos\_token, \**

**return\_tensors='pt')**

**def bot\_response(self):**

**# append the new user input tokens to the chat history**

**# if chat has already begun**

**if self.chat\_history\_ids is not None:**

**self.bot\_input\_ids = torch.cat([self.chat\_history\_ids, self.new\_user\_input\_ids], dim=-1)**

**else:**

**# if first entry, initialize bot\_input\_ids**

**self.bot\_input\_ids = self.new\_user\_input\_ids**

**# define the new chat\_history\_ids based on the preceding chats**

**# generated a response while limiting the total chat history to 1000 tokens,**

**self.chat\_history\_ids = model.generate(self.bot\_input\_ids, max\_length=1000, \**

**pad\_token\_id=tokenizer.eos\_token\_id)**

**# last ouput tokens from bot**

**response = tokenizer.decode(self.chat\_history\_ids[:, self.bot\_input\_ids.shape[-1]:][0], \**

**skip\_special\_tokens=True)**

**# in case, bot fails to answer**

**if response == "":**

**response = self.random\_response()**

**# print bot response**

**print('ChatBot >> '+ response)**

**# in case there is no response from model**

**def random\_response(self):**

**i = -1**

**response = tokenizer.decode(self.chat\_history\_ids[:, self.bot\_input\_ids.shape[i]:][0], \**

**skip\_special\_tokens=True)**

**# iterate over history backwards to find the last token**

**while response == '':**

**i = i-1**

**response = tokenizer.decode(self.chat\_history\_ids[:, self.bot\_input\_ids.shape[i]:][0], \**

**skip\_special\_tokens=True)**

**# if it is a question, answer suitably**

**if response.strip() == '?':**

**reply = np.random.choice(["I don't know",**

**"I am not sure"])**

**# not a question? answer suitably**

**else:**

**reply = np.random.choice(["Great",**

**"Fine. What's up?",**

**"Okay"**

**])**

**return reply**

**# build a ChatBot object**

**bot = ChatBot()**

**# start chatting**

**while True:**

**# receive user input**

**bot.user\_input()**

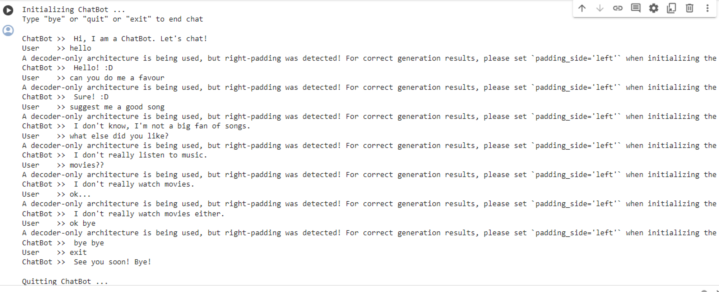
**# check whether to end chat**

**if bot.end\_chat:**

**break**

**# output bot response**

**bot.bot\_response()**



**APPENDIX:** **https://github.com/Jagadeep21/IBM-PROJECT-Gen-AI.git**