AI Chatbot Using ChatGPT

A Project Report

submitted in partial fulfillment of the requirements

of

AIML fundamentals with cloud computing Gen AI

by

HARIHARAN.B

hharandgl1967@gmail.com

492773C9961E389AF604ED1337E25624

Under the Guidance of

Name of Guide (P.Raja, Master Trainer)

ACKNOWLEDGEMENT

We would like to take this opportunity to express our deep sense of gratitude to all individuals who helped us directly or indirectly during this thesis work.Firstly, we would like to thank my supervisor, p.Raja(Master trainer of Edunet) for being a great mentor and the best adviser I could ever have. His advice, encouragement and the critics are a source of innovative ideas, inspiration and causes behind the successful completion of this project. The confidence shown in me by him was the biggest source of inspiration for me. It has been a privilege working with him for the last one year. He always helped me during my project and many other aspects related to the program. His talks and lessons not only help in project work and other activities of the program but also make me a good and responsible professional.

This project explores the development and deployment of an intelligent conversational agent using OpenAI's ChatGPT, a state-of-the-art language model based on the GPT-4 architecture. Chatbots are increasingly used across various industries to improve customer engagement, streamline workflows, and provide real-time assistance. Leveraging natural language processing (NLP) and deep learning, ChatGPT can understand and respond to human input in a conversational style, offering versatile applications in customer service, virtual assistance, education, and more.

The primary objective of this project is to design a chatbot that can engage users with coherent and contextually relevant responses while adapting to diverse conversation flows. By harnessing ChatGPT's capabilities, we address key challenges in chatbot development, including natural language understanding, response generation, and maintaining contextual awareness over extended dialogues. This project also explores integrating ChatGPT with external APIs for task automation and personalizing user interactions by tailoring responses based on user-specific data.

Results demonstrate the potential of ChatGPT-based chatbots to handle dynamic interactions with minimal supervision, offering a scalable and efficient solution for conversational AI applications. Future work includes enhancing the chatbot's memory capabilities for better contextual retention across sessions and exploring domain-specific fine-tuning to increase relevance in specialized fields.

TABLE OF CONTENTS

CHAPTER 1

Introduction

1.1 Problem Statement

In today’s digital landscape, organizations face increasing demand for real-time, responsive communication with users, clients, and employees. Traditional customer service models, which rely heavily on human agents, can be inefficient, costly, and challenging to scale. Users expect fast, accurate, and personalized responses to their queries across multiple platforms and at any time of day. Meeting these expectations with human resources alone often leads to long response times, high operating costs, and inconsistencies in service quality.This project addresses the need for an intelligent, automated chatbot that can engage in meaningful, contextually aware conversations, providing users with instant support and information. By leveraging OpenAI’s ChatGPT model, the proposed chatbot will be capable of understanding user intent, generating coherent and relevant responses, and adapting to different conversation flows without human intervention.

1.2 Motivation

the motivation for creating AI chatbots stems from their ability to enhance operational efficiency, improve user experiences, and drive cost savings. By automating repetitive tasks such as answering FAQs or guiding users through standard processes, chatbots allow human employees to focus on more complex, value-added work. This efficiency boost not only streamlines operations but also supports productivity across diverse sectors. Another key motivator is the positive impact on user experience. Chatbots provide instant responses, are available 24/7, and offer a consistent level of service, meeting modern expectations for quick and reliable support. Many chatbots are also designed to deliver personalized experiences, adapting to individual user preferences, which makes interactions more engaging and meaningful

1.3 Objective

objective of creating an AI chatbot is to develop an automated, efficient, and user-friendly solution that addresses a range of tasks across various industries. By leveraging artificial intelligence, chatbots can provide immediate support, handle large volumes of inquiries simultaneously, and operate continuously, enhancing both user experience and organizational productivity. One primary objective is to automate routine tasks, such as answering common questions, processing transactions, or providing product recommendations.

1.4 scope of the project

AI chatbots have a wide-ranging scope, impacting various industries. They excel in customer service, providing 24/7 support and efficient problem-solving. In e-commerce, they offer personalized recommendations and streamline the shopping experience. Healthcare benefits from AI chatbots for patient engagement, remote monitoring, and appointment scheduling. Education leverages them for personalized learning, tutoring, and administrative tasks. Financial services utilize AI chatbots for customer support, financial advice, and fraud detection. As AI technology advances, the scope of AI chatbots is expanding to include mental health support, language translation, and creative content generatio

CHAPTER 2

Literature Survey

2.1 Literature Review

AI chatbots heavily depends on the quality of data they are trained on, and poor or biased data can lead to incorrect or irrelevant responses. Security and privacy concerns are another challenge, as chatbots often deal with sensitive information, requiring robust safeguards to protect user data. Furthermore, user frustration can arise when chatbots fail to understand queries or provide unsatisfactory answers. In conclusion, while AI chatbots are highly effective in automating tasks and improving user experiences, they are not without limitations, particularly when dealing with more complex or emotional interactions. As technology advances, however, chatbots are expected to become more sophisticated, addressing many of these challenges and further enhancing their value.

2.2 **e**xisting models

There are several existing models and frameworks used for creating AI chatbots, each serving different purposes and complexities .Rule-based models, for example, operate on predefined sets of rules or patterns, matching user input with specific keywords or phrases to provide fixed responses. Pre-trained language models, such as OpenAI’s GPT or Google’s BERT, are large-scale models trained on vast datasets and can be fine-tuned for specific tasks, offering high versatility for building advanced chatbots

2.3 Limitations of the Existing Model

Existing AI chatbot models have several limitations, including limited contextual understanding, emotional intelligence, and personalization. Rule-based and retrieval-based models often struggle with complex or nuanced conversations, while generative models, although more flexible, may still provide inaccurate or irrelevant responses. Many chatbots also depend on predefined data, which can lead to biased or incomplete answers.

CHAPTER 3

Proposed Methodology

3.1 process

1. Define Purpose and Scope

- **Identify goals**: Determine what the chatbot should achieve (e.g., customer support, entertainment, information retrieval, etc.).
- **Define scope**: Decide the level of complexity, such as if the chatbot will handle simple questions or complex conversations, and whether it needs to integrate with other systems or services.

2. Design Conversation Flow

- **Create user stories**: Map out the potential conversations the chatbot will handle.
- **Design flow diagrams**: Visualize how the chatbot should respond to various user inputs using flowcharts or decision trees.
- **Determine intents and entities**: Intents represent the user's goal, while entities are specific details that help fulfill the intent.

3. Choose a Platform and Tools

- **Natural Language Processing (NLP) engine**: Choose tools like GPT, Rasa, or Wit.ai to process and understand text.
- **Development framework**: Select tools like Dialogflow, Microsoft Bot Framework, or Botpress to build, deploy, and manage the chatbot.
- **Hosting**: Decide on hosting options (e.g., AWS, Azure, or local server).

4. Develop the Chatbot

- **Preprocessing**: Tokenize, clean, and normalize the input text.
- **Intent recognition**: Train the NLP model to recognize user intents from text inputs.
- **Entity extraction**: Program the chatbot to identify and extract relevant entities (e.g., dates, names, products).
- **Dialog management**: Implement a system to maintain context and handle multi-turn conversations.

5. Train the Model

- **Prepare training data**: Collect datasets with example sentences for each intent.
- **Train NLP model**: Train the model using your chosen NLP engine, improving accuracy by providing more training data and feedback.

6. Test the Chatbot

- **Unit testing**: Test individual components, like NLP and intent recognition.
- **End-to-end testing**: Run full conversation tests to ensure the chatbot functions as expected in various scenarios.

7. Maintenance and Improvement

- **Analyze conversations**: Review chat logs to identify improvement areas and recurring issues.
- **Update and retrain**: Regularly retrain the model with new data to improve accuracy.
- **Iterate**: Continuously improve the bot's capabilities by adding new intents and refining existing ones.

3.2 Advantages

- **Consistent and Accurate Responses**: AI chatbots provide consistent responses based on predefined data, ensuring that users receive accurate information every time. They eliminate the possibility of human error or variability in answers, ensuring quality control.

- **Scalability**: AI chatbots can easily scale to handle increased interaction volumes without compromising performance. This is particularly useful during high-traffic events, such as product launches, sales, or marketing campaigns.

- **Data Collection and Insights**: AI chatbots can track and analyze user interactions, collecting valuable data on customer behavior, preferences, and common inquiries. This data can be used to improve services, products, and marketing strategies.

- **Easy Integration**: AI chatbots can be easily integrated into various platforms such as websites, mobile apps, social media, and messaging services. This ensures that users can access the chatbot from multiple touchpoints.

3.3 Required specification

1. Natural Language Processing (NLP) Capabilities

- **Intent Recognition**: The chatbot must be able to identify the user's intent, whether it's asking a question, making a request, or expressing an opinion
- **Context Understanding**: The chatbot should understand and maintain context across multiple turns of conversation, remembering past interactions within the same session or longer-term

2. Data Sources and Knowledge Base

- **Training Data**: A large and diverse dataset, including conversations, frequently asked questions, product information, or domain-specific knowledge, is required for training the AI mode

3. Integration Capabilities

- **Platform Integration**: The chatbot must integrate seamlessly with the platforms where it will operate, such as websites, mobile apps, social media (Facebook Messenger, WhatsApp), or messaging platforms (Slack, Microsoft Teams).

4. User Interface (UI) and User Experience (UX)

- **Conversational Flow**: The chatbot should be designed to follow a logical and intuitive flow, making interactions feel natural to the user.
- **Visual Interface**: For chatbots integrated with websites or mobile apps, a clear and easy-to-navigate chat interface is essential, including buttons, quick replies, and rich media options (images, buttons, links).

5. Security and Privacy

- **Data Encryption**: To protect sensitive information, the chatbot should encrypt data both at rest and in transit, ensuring secure communication.

3.4 Hardware require

1. **Development/Training**:
   - **GPUs**: High-performance options like NVIDIA Tesla or GeForce RTX for deep learning.
   - **CPU**: Multi-core processors (16 cores or more) like Intel i9 or AMD Ryzen.
   - **RAM**: At least 32GB–64GB for large-scale training.
   - **Storage**: SSDs (1TB+) for faster data access.
2. **Deployment/Inference**:
   - **CPU**: Powerful multi-core CPUs (Intel i7 or i9).
   - **RAM**: 16GB–32GB for smooth real-time performance.
   - **Storage**: SSD (512GB–1TB) for fast model access.
   - **Network**: High-speed internet (1Gbps or more) for handling traffic.

3.5 software require

- **NLP Libraries**: SpaCy, NLTK, Hugging Face Transformers (for intent recognition, entity extraction).
- **Machine Learning Frameworks**: TensorFlow, PyTorch, Keras (for model training).
- **Backend Frameworks**: Flask, Django, FastAPI (for API and chatbot server).
- **Cloud Platforms**: AWS, Google Cloud, Microsoft Azure (for scalable infrastructure).

- **Databases**: MongoDB, MySQL, PostgreSQL (for storing user data and conversation logs).
- **Frontend Tools**: ReactJS, VueJS, Bootstrap (for UI/UX).
- **Version Control**: Git, GitHub (for collaborative development).
- **Testing Tools**: Postman, unit testing libraries.
- **Analytics**: Google Analytics, Grafana (for monitoring performance).
- **Security**: OAuth, JWT, SSL/TLS encryption (for secure user interaction).

CHAPTER 4

Implementation and Result

Key Findings:

1. Python 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()
```

Chapter 5

Discussion and Conclusion

Limitation

1. Understanding Context

- **Short-Term Memory**: Many AI chatbots, including basic ones like GPT-2, do not retain long-term context. They can only understand and respond based on the current input, not previous interactions (unless designed to store conversation history).

2. Generalization vs. Specialization

- **Limited Domain Knowledge**: A general-purpose chatbot like GPT-2 might not perform well in specialized domains (e.g., medical, legal advice) unless fine-tuned with domain-specific d

3. Handling Ambiguity

- **Misinterpretation**: Chatbots may struggle to handle ambiguous user queries, often leading to misunderstandings or irrelevant answers.

4. Privacy and Security Risks

- **Data Privacy**: Collecting and processing user data can raise concerns about privacy, especially if personal information is involved.
- **Security Threats**: Chatbots can be vulnerable to malicious use, such as data breaches or exploitation of flaws in the system.

Conclusion:

Conclusion of AI Chatbot:

AI chatbots represent a transformative technology that enhances user experiences across various industries by enabling efficient, automated, and interactive communication. They leverage advanced natural language processing (NLP) models and machine learning algorithms to understand user input and generate relevant, human-like responses. The future of AI chatbots holds great potential for integration across multiple domains like healthcare, education, e-commerce, and accessibility, providing personalized, efficient, and responsive experiences. The ongoing development of emotionally intelligent chatbots, multilingual capabilities, and voice interactions will further broaden their application and impact.AI chatbots are shaping the future of digital communication, and their role is poised to grow as technology evolves. With proper implementation, they can significantly improve business operations, enhance user engagement, and create new opportunities for innovation.