**Chatbot**

**TABLE OF CONTENTS**

[**1.** **INTRODUCTION:** 3](#_Toc124350042)

[**2.** **PROBLEM STATEMENT:** 3](#_Toc124350043)

[**3.** **SCOPE:** 4](#_Toc124350044)

[**4.** **OBJECTIVE:** 4](#_Toc124350045)

[**5.** **METHODOLOGY:** 5](#_Toc124350046)

[**6.** **FLOWCHART:** 6](#_Toc124350047)

[**7.** **CODE FILES:** 7](#_Toc124350048)

[**Training.py FILE** 7](#_Toc124350049)

[**App.py FILE** 10](#_Toc124350050)

[**Data.json FILE** 12](#_Toc124350051)

[**Index.html FILE** 14](#_Toc124350052)

[**Style.css File** 17](#_Toc124350053)

[**8.** **CONCLUSION:** 21](#_Toc124350054)

[**9.** **FUTURE WORK:** 21](#_Toc124350055)

# **INTRODUCTION:**

Customer service AI chatbots are automated software applications that use artificial intelligence to simulate conversations with customers. They are designed to provide customer service, answer customer inquiries, provide product information, and complete customer transactions. AI chatbots are used by businesses to provide customer service 24/7, saving businesses money in the process. AI chatbots use natural language processing (NLP) to understand customer inquiries and provide accurate, personalized responses. They are designed to quickly answer customer questions, provide product information, and complete transactions in a timely manner. AI chatbots can also be used to streamline customer service processes, such as processing refunds and handling customer complaints. AI chatbots are becoming increasingly popular with businesses due to their cost-saving capabilities and ability to provide 24/7 customer service.

AI chatbots are also becoming increasingly sophisticated, with the ability to understand customer sentiment and provide personalized responses. AI chatbots are being used by businesses to provide customer service in a variety of industries, from retail to banking. AI chatbots are also being used to provide customer service in different languages, allowing businesses to better serve customers from all over the world. With the advancements in AI technology, AI chatbots are becoming increasingly important for businesses looking to provide customer service in a more efficient and cost-effective manner.

# **PROBLEM STATEMENT:**

The development of AI chatbots has been an ongoing process since the early 2000s, but the technology has yet to reach its full potential. AI chatbots are still limited in their ability to accurately interpret and respond to natural language and do not yet understand the nuances of human communication. Furthermore, AI chatbots cannot yet handle complex conversations in a meaningful way, and their conversation is often limited to predetermined, pre-scripted responses.

Another limitation of AI chatbots is their inability to interpret non-verbal cues, such as facial expressions and body language. This means that AI chatbots cannot effectively understand the emotional states of the user and are unable to react to their emotional needs. Furthermore, AI chatbots lack the human-like empathy and understanding of the user’s needs that would make them more effective conversation partners. As a result, AI chatbots are often unable to engage in meaningful conversations with users, resulting in a less than satisfactory user experience.

Finally, AI chatbots are unable to learn from their conversations with users, meaning their ability to respond appropriately to different scenarios and contexts is limited. Without the ability to learn from user input, AI chatbots cannot develop the natural language processing capabilities needed to understand and respond to complex conversations. Additionally, AI chatbots are unable to build on conversations over time and become more knowledgeable about the user and their needs. This lack of learning ability means that AI chatbots are unable to evolve and become more intelligent, resulting in a stagnant experience for users.

# **SCOPE:**

The scope of this project includes designing, building, and testing the customer service chatbot. This will include creating the chatbot's conversation flow, integrating the chatbot with the store's customer service systems, and training the chatbot with customer service data. The chatbot should be able to understand natural language inputs and respond in a way that is relevant to the conversation. The scope also includes testing the chatbot in real-world customer service scenarios and ensuring that it provides a satisfactory customer service experience. Chatbots can be used for a variety of tasks, including customer service, personal assistant, marketing, sales, and more. They can help businesses automate tasks, engage customers, improve customer service, and provide insights into customer behavior. They can also provide product recommendations, improve customer loyalty, and provide personalized experiences. Additionally, chatbots can be used to facilitate conversations with customers, create automated customer surveys, and provide support for customer service teams.

# **OBJECTIVE:**

The objectives of this project are to create a customer service chatbot that can provide a satisfactory customer service experience. The chatbot should be able to answer basic customer service questions and provide relevant information about the store's products and services. The chatbot should also be able to facilitate customer service inquiries and redirect to human customer service representatives when needed. Finally, the chatbot should be tested and evaluated in real-world customer service scenarios to ensure that it is delivering a satisfactory customer service experience.

1. **Increase customer engagement:**

Chatbots can be used to provide customers with a more interactive and engaging experience by providing quick access to information and responding to customer inquiries in a conversational format.

1. **Automate customer service:**

Chatbots can be used to automate customer service tasks, such as answering FAQs, providing support, and responding to customer inquiries.

1. **Collect customer data:**

Chatbots can be used to collect customer data, such as demographics, purchase history and website interactions, which can be used to better tailor marketing and product offerings.

1. **Increase sales:**

Chatbots can be used to increase sales by providing personalized product recommendations, upselling and cross-selling opportunities and discounts.

1. **Enhance customer experience:**

Chatbots can be used to enhance customer experience by providing a more personalized experience, such as providing tailored product recommendations and providing a more interactive customer service experience.

# **METHODOLOGY:**

1. **Design the User Interface:**

Design the UI of your chatbot according to the use cases identified. Make sure that the design is user-friendly and intuitive enough for a customer to interact with the chatbot.

1. **Plan the Conversation Flow**:

Plan out the conversation flow for each use case. This includes identifying the questions that the customer may ask and the possible responses that the chatbot can provide.

1. **Develop Natural Language Processing:**

Natural language processing is used to process the customer’s query and extract the key information required to provide an appropriate response.

1. **Develop Natural Language Understanding:**

Check what the user wants to say, remove all the ambiguities from the data and user asked question. Do tokenization, stemming and lemmatization

1. **Develop Natural Language Generation:**

Generate the response from the knowledge base and from the understanding. Easy and simplest answer to user from bot

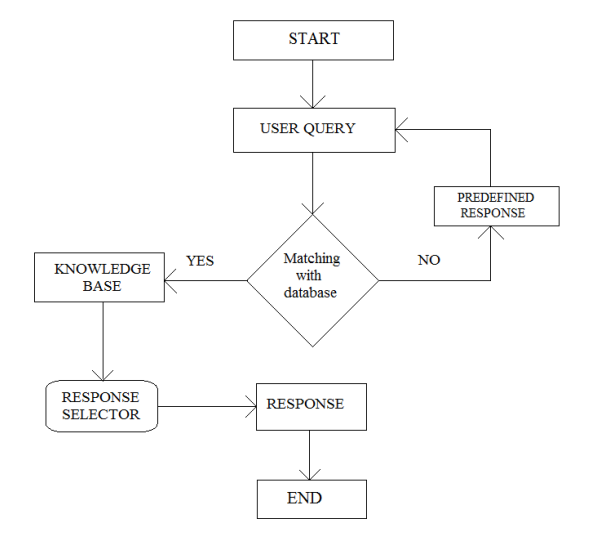
1. **Build the Chatbot:**

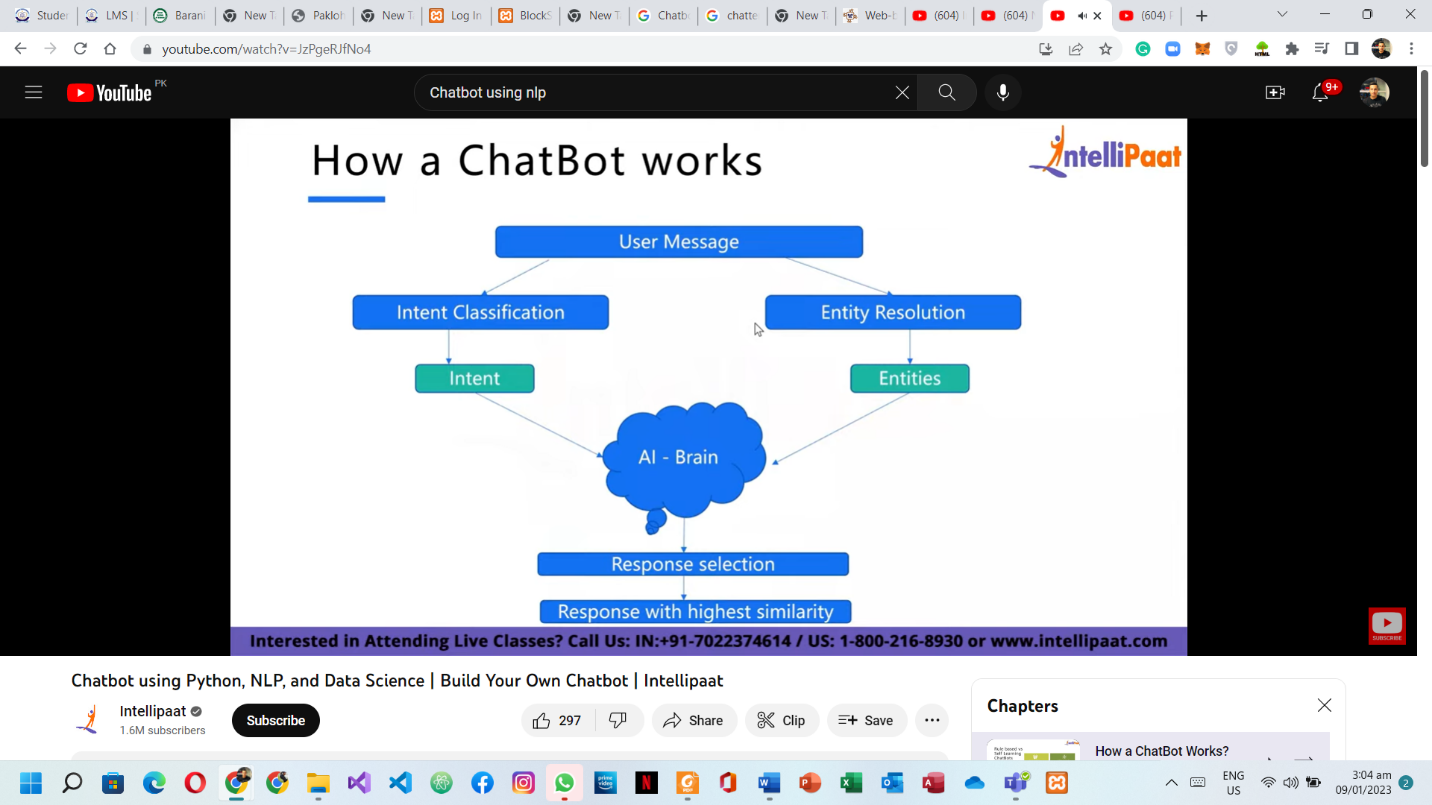
Once the UI and conversation flow are designed and the NLP is developed, the chatbot can be built. This requires coding the chatbot logic and connecting it to the customer service system.

1. **Test and Deploy:**

Finally, the chatbot should be tested and deployed in the customer service system. This step ensures that the chatbot is working as expected and can provide timely and accurate customer service.

# **FLOWCHART:**





# **CODE FILES:**

## **Training.py FILE**

import nltk

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

"""

For Natural Languages processing, nltk Library is used in python

nltk stands for natural language toolkit

As, We need to Tokenize the complete statement from corpus and from user entered string

for that, we need nltk.stem - stemming

and import wordNetLemmatizer

"""

import json

"""

Our data is present in json file

So, we are using json library

"""

import pickle

"""

As, I have mentioned above our data is present in json file.

We need to convert this data into byte stream thats why we are using pickle

"""

import numpy as np

"""

Used to store large data, Majorly used in machine learning

"""

from keras.models import Sequential

from keras.layers import Dense, Activation, Dropout

from keras.optimizers import SGD

"""

For giving brain to our bot, we are using Sequential model from keras

We are using Dense, Activation and dropout layers

Dense - layer which is deply connected with its preceding layers

Activation -

Dropout - Helps to prevent neurons to set synchronous weights

SGD - Gardient descent - Find error and then increase accuracy of our bot

"""

import random

# To generate random responses

words=[]

classes = []

documents = []

ignore\_words = ['?', '!']

"""

Lists:

To store words, classes and document

? and ! are ignored words, if enetred with question

"""

data\_file = open('data.json').read()

intents = json.loads(data\_file)

"""

Simply opening our data file and storing it in data\_file object

Then, with the help of that object, we are converting json strings into python dictionary

"""

for intent in intents['intents']:

    for pattern in intent['patterns']:

        #tokenize each word

        w = nltk.word\_tokenize(pattern)

        words.extend(w)

        # add documents in the corpus

        documents.append((w, intent['tag']))

        # add to our classes list

        if intent['tag'] not in classes:

            classes.append(intent['tag'])

"""

Iterating the complete python dictionary, and then tokenizeing it

Tokenize - Splitting in words

"""

# lemmaztize and lower each word and remove duplicates

words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore\_words]

words = sorted(list(set(words)))

# sort classes

classes = sorted(list(set(classes)))

# documents = combination between patterns and intents

print (len(documents), "documents")

# classes = intents

print (len(classes), "classes", classes)

# words = all words, vocabulary

print (len(words), "unique lemmatized words", words)

pickle.dump(words,open('texts.pkl','wb'))

pickle.dump(classes,open('labels.pkl','wb'))

"""

We are creating and storing all our words in text.pkl and

classes in label.pkl

Using dump - to store python objects in file

"""

# create our training data

training = []

# create an empty list for our output

output\_empty = [0] \* len(classes)

# training set, bag of words for each sentence

for doc in documents:

    # initialize our bag of words

    bag = []

    # list of tokenized words for the pattern

    pattern\_words = doc[0]

    # lemmatize each word - create base word, in attempt to represent related words

    pattern\_words = [lemmatizer.lemmatize(word.lower()) for word in pattern\_words]

    # create our bag of words array with 1, if word match found in current pattern

    for w in words:

        bag.append(1) if w in pattern\_words else bag.append(0)

    # output is a '0' for each tag and '1' for current tag (for each pattern)

    output\_row = list(output\_empty)

    output\_row[classes.index(doc[1])] = 1

    training.append([bag, output\_row])

# shuffle our features and turn into np.array

random.shuffle(training)

training = np.array(training)

# create train and test lists. X - patterns, Y - intents

train\_x = list(training[:,0])

train\_y = list(training[:,1])

print("Training data created")

# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd output layer contains number of neurons

# equal to number of intents to predict output intent with softmax

model = Sequential()

model.add(Dense(128, input\_shape=(len(train\_x[0]),), activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(64, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(len(train\_y[0]), activation='softmax'))

# Compile model. Stochastic gradient descent with Nesterov accelerated gradient gives good results for this model

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)

model.compile(loss='categorical\_crossentropy', optimizer=sgd, metrics=['accuracy'])

#fitting and saving the model

hist = model.fit(np.array(train\_x), np.array(train\_y), epochs=200, batch\_size=5, verbose=1)

model.save('model.h5', hist)

print("model created")

## **App.py FILE**

import nltk

nltk.download('popular')

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

import pickle

import numpy as np

from keras.models import load\_model

model = load\_model('model.h5')

import json

import random

intents = json.loads(open('data.json').read())

words = pickle.load(open('texts.pkl','rb'))

classes = pickle.load(open('labels.pkl','rb'))

def clean\_up\_sentence(sentence):

    # tokenize the pattern - split words into array

    sentence\_words = nltk.word\_tokenize(sentence)

    # stem each word - create short form for word

    sentence\_words = [lemmatizer.lemmatize(word.lower()) for word in sentence\_words]

    return sentence\_words

# return bag of words array: 0 or 1 for each word in the bag that exists in the sentence

def bow(sentence, words, show\_details=True):

    # tokenize the pattern

    sentence\_words = clean\_up\_sentence(sentence)

    # bag of words - matrix of N words, vocabulary matrix

    bag = [0]\*len(words)

    for s in sentence\_words:

        for i,w in enumerate(words):

            if w == s:

                # assign 1 if current word is in the vocabulary position

                bag[i] = 1

                if show\_details:

                    print ("found in bag: %s" % w)

    return(np.array(bag))

def predict\_class(sentence, model):

    # filter out predictions below a threshold

    p = bow(sentence, words,show\_details=False)

    res = model.predict(np.array([p]))[0]

    ERROR\_THRESHOLD = 0.25

    results = [[i,r] for i,r in enumerate(res) if r>ERROR\_THRESHOLD]

    # sort by strength of probability

    results.sort(key=lambda x: x[1], reverse=True)

    return\_list = []

    for r in results:

        return\_list.append({"intent": classes[r[0]], "probability": str(r[1])})

    return return\_list

def getResponse(ints, intents\_json):

    tag = ints[0]['intent']

    list\_of\_intents = intents\_json['intents']

    for i in list\_of\_intents:

        if(i['tag']== tag):

            result = random.choice(i['responses'])

            break

    return result

def chatbot\_response(msg):

    ints = predict\_class(msg, model)

    res = getResponse(ints, intents)

    return res

from flask import Flask, render\_template, request

app = Flask(\_\_name\_\_)

app.static\_folder = 'static'

@app.route("/")

def home():

    return render\_template("index.html")

@app.route("/get")

def get\_bot\_response():

    userText = request.args.get('msg')

    return chatbot\_response(userText)

if \_\_name\_\_ == "\_\_main\_\_":

    app.run()

## **Data.json FILE**

{"intents": [

    {"tag": "greeting",

    "patterns": ["Hi there", "How are you", "Is anyone there?","Hey","Hola", "Hello", "Good day"],

    "responses": ["Hello, thanks for asking", "Good to see you again", "Hi there, how can I help?"],

    "context": [""]

    },

    {"tag": "goodbye",

    "patterns": ["Bye", "See you later", "Goodbye", "Nice chatting to you, bye", "Till next time"],

    "responses": ["See you!", "Have a nice day", "Bye! Come back again soon."],

    "context": [""]

    },

    {"tag": "thanks",

    "patterns": ["Thanks", "Thank you", "That's helpful", "Awesome, thanks", "Thanks for helping me"],

    "responses": ["Happy to help!", "Any time!", "My pleasure"],

    "context": [""]

    },

    {"tag": "website\_working",

    "patterns": ["The website is not working loading", "The website is not loading properly", "The speed of website is very slow" ],

    "responses": ["If the website is not working properly, please check your internet connection first", "Website is not loading properly beacuse of some critical error","The website is slow due to maintenance"],

    "context": [""]

    },

    {"tag": "noanswer",

    "patterns": [],

    "responses": ["Sorry, can't understand you", "Please give me more info", "Not sure I understand"],

    "context": [""]

    },

    {"tag": "options",

    "patterns": ["How you could help me?", "What you can do?", "What help you provide?", "How you can be helpful?", "What support is offered"],

    "responses": ["I can guide you through Adverse drug reaction list, Blood pressure tracking, Hospitals and Pharmacies", "Offering support for Adverse drug reaction, Blood pressure, Hospitals and Pharmacies"],

    "context": [""]

    },

    {"tag": "adverse\_drug",

    "patterns": ["How to check Adverse drug reaction?", "Open adverse drugs module", "Give me a list of drugs causing adverse behavior", "List all drugs suitable for patient with adverse reaction", "Which drugs dont have adverse reaction?" ],

    "responses": ["Navigating to Adverse drug reaction module"],

    "context": [""]

    },

    {"tag": "blood\_pressure",

    "patterns": ["Open blood pressure module", "Task related to blood pressure", "Blood pressure data entry", "I want to log blood pressure results", "Blood pressure data management" ],

    "responses": ["Navigating to Blood Pressure module"],

    "context": [""]

    },

    {"tag": "blood\_pressure\_search",

    "patterns": ["I want to search for blood pressure result history", "Blood pressure for patient", "Load patient blood pressure result", "Show blood pressure results for patient", "Find blood pressure results by ID" ],

    "responses": ["Please provide Patient ID", "Patient ID?"],

    "context": ["search\_blood\_pressure\_by\_patient\_id"]

    },

    {"tag": "search\_blood\_pressure\_by\_patient\_id",

    "patterns": [],

    "responses": ["Loading Blood pressure result for Patient"],

    "context": [""]

    },

    {"tag": "pharmacy\_search",

    "patterns": ["Find me a pharmacy", "Find pharmacy", "List of pharmacies nearby", "Locate pharmacy", "Search pharmacy" ],

    "responses": ["Please provide pharmacy name"],

    "context": ["search\_pharmacy\_by\_name"]

    },

    {"tag": "search\_pharmacy\_by\_name",

    "patterns": [],

    "responses": ["Loading pharmacy details"],

    "context": [""]

    },

    {"tag": "hospital\_search",

    "patterns": ["Lookup for hospital", "Searching for hospital to transfer patient", "I want to search hospital data", "Hospital lookup for patient", "Looking up hospital details" ],

    "responses": ["Please provide hospital name or location"],

    "context": ["search\_hospital\_by\_params"]

    },

    {"tag": "search\_hospital\_by\_params",

    "patterns": [],

    "responses": ["Please provide hospital type"],

    "context": ["search\_hospital\_by\_type"]

    },

    {"tag": "search\_hospital\_by\_type",

    "patterns": [],

    "responses": ["Loading hospital details"],

    "context": [""]

    }

]

}

## **Index.html FILE**

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>Chatbot</title>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <meta http-equiv="X-UA-Compatible" content="ie=edge">

  <link rel="stylesheet" href="{{ url\_for('static', filename='styles/style.css') }}">

  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

</head>

<body>

  <section class="msger">

    <header class="msger-header">

      <div class="msger-header-title">

        Chatbot

      </div>

    </header>

    <main class="msger-chat">

      <div class="msg left-msg">

        <div class="msg-img">

        </div>

        <div class="msg-bubble">

          <div class="msg-info">

            <div class="msg-info-name">

              Chatbot

            </div>

          </div>

          <div class="msg-text">

            Hi, welcome to ChatBot!

          </div>

        </div>

      </div>

    </main>

    <form class="msger-inputarea">

      <input type="text" class="msger-input" id="textInput" placeholder="Please enter your message">

      <button type="submit" class="msger-send-btn">Send</button>

    </form>

  </section>

  <script src='https://use.fontawesome.com/releases/v5.0.13/js/all.js'></script>

  <script>

    const msgerForm = get(".msger-inputarea");

    const msgerInput = get(".msger-input");

    const msgerChat = get(".msger-chat");

    const BOT\_IMG = "";

    const PERSON\_IMG = "";

    const BOT\_NAME = "    ChatBot";

    const PERSON\_NAME = "Muhammad Umar";

    msgerForm.addEventListener("submit", event => {

      event.preventDefault();

      const msgText = msgerInput.value;

      if (!msgText) return;

      appendMessage(PERSON\_NAME, PERSON\_IMG, "right", msgText);

      msgerInput.value = "";

      botResponse(msgText);

    });

    // User message

    function appendMessage(name, img, side, text) {

      const msgHTML = `

                      <div class="msg ${side}-msg">

                        <div class="msg-img" style="background-image: url(${img})"></div>

                        <div class="msg-bubble">

                          <div class="msg-info">

                            <div class="msg-info-name">${name}</div>

                            <div class="msg-info-time">${formatDate(new Date())}</div>

                          </div>

                          <div class="msg-text">${text}</div>

                        </div>

                      </div>

                      `;

      msgerChat.insertAdjacentHTML("beforeend", msgHTML);

      msgerChat.scrollTop += 500;

    }

      // Bot Response

    function botResponse(rawText) {

      $.get("/get", { msg: rawText }).done(function (data) {

        console.log(rawText);

        console.log(data);

        const msgText = data;

        appendMessage(BOT\_NAME, BOT\_IMG, "left", msgText);

      });

    }

    function get(selector, root = document) {

      return root.querySelector(selector);

    }

    function formatDate(date) {

      const h = "0" + date.getHours();

      const m = "0" + date.getMinutes();

      return `${h.slice(-2)}:${m.slice(-2)}`;

    }

  </script>

</body>

</html>

## **Style.css File**

:root {

    --body-bg: linear-gradient(135deg, #f5f7fa 0%, #c3cfe2 100%);

    --msger-bg: #fff;

    --border: 2px solid #ddd;

    --left-msg-bg: #ececec;

    --right-msg-bg: #579ffb;

}

html {

    box-sizing: border-box;

}

  \*,

  \*:before,

  \*:after {

    margin: 0;

    padding: 0;

    box-sizing: inherit;

}

body {

    display: flex;

    justify-content: center;

    align-items: center;

    height: 100vh;

    background-image: var(--body-bg);

    font-family: Helvetica, sans-serif;

}

.msger {

    display: flex;

    flex-flow: column wrap;

    justify-content: space-between;

    width: 100%;

    max-width: 867px;

    margin: 25px 10px;

    height: calc(100% - 50px);

    border: var(--border);

    border-radius: 5px;

    background: var(--msger-bg);

    box-shadow: 0 15px 15px -5px rgba(0, 0, 0, 0.2);

}

.msger-header {

    /\* display: flex; \*/

    font-size: medium;

    justify-content: space-between;

    padding: 10px;

    text-align: center;

    border-bottom: var(--border);

    background: #eee;

    color: #666;

}

.msger-chat {

    flex: 1;

    overflow-y: auto;

    padding: 10px;

}

.msger-chat::-webkit-scrollbar {

    width: 6px;

}

.msger-chat::-webkit-scrollbar-track {

    background: #ddd;

}

.msger-chat::-webkit-scrollbar-thumb {

    background: #bdbdbd;

}

.msg {

    display: flex;

    align-items: flex-end;

    margin-bottom: 10px;

}

.msg-img {

    width: 50px;

    height: 50px;

    margin-right: 10px;

    background: #ddd;

    background-repeat: no-repeat;

    background-position: center;

    background-size: cover;

    border-radius: 50%;

}

.msg-bubble {

    max-width: 450px;

    padding: 15px;

    border-radius: 15px;

    background: var(--left-msg-bg);

}

.msg-info {

    display: flex;

    justify-content: space-between;

    align-items: center;

    margin-bottom: 10px;

}

.msg-info-name {

    margin-right: 10px;

    font-weight: bold;

}

.msg-info-time {

    font-size: 0.85em;

}

.left-msg .msg-bubble {

    border-bottom-left-radius: 0;

}

.right-msg {

    flex-direction: row-reverse;

}

.right-msg .msg-bubble {

    background: var(--right-msg-bg);

    color: #fff;

    border-bottom-right-radius: 0;

}

.right-msg .msg-img {

    margin: 0 0 0 10px;

}

.msger-inputarea {

    display: flex;

    padding: 10px;

    border-top: var(--border);

    background: #eee;

}

.msger-inputarea \* {

    padding: 10px;

    border: none;

    border-radius: 3px;

    font-size: 1em;

}

.msger-input {

    flex: 1;

    background: #ddd;

}

.msger-send-btn {

    margin-left: 10px;

    background: rgb(0, 196, 65);

    color: #fff;

    font-weight: bold;

    cursor: pointer;

    transition: background 0.23s;

}

.msger-send-btn:hover {

    background: rgb(0, 180, 50);

}

.msger-chat {

    background-color: #fcfcfe;

}

# **CONCLUSION:**

In conclusion, the potential of AI chatbot for customer service is huge and could revolutionize the customer service industry. It has the potential to provide customers with a more personalized experience, respond to their queries more quickly, and save businesses money by automating customer service tasks. Despite challenges that still need to be addressed, it is clear that Ai chatbot is a technology that is worth exploring in order to improve customer service

# **FUTURE WORK:**

One possible area of future work regarding AI chatbot for customer service is related to natural language processing. Natural language processing (NLP) is a form of artificial intelligence that allows computers to understand and interpret human language. With advancements in NLP technology, AI chatbots can become more natural and fluid in their conversations with customers. AI chatbots can be programmed to understand the context of a customer’s question and generate an appropriate response. This will allow for more natural conversations and reduce the need for customer service representatives to manually respond to customer inquiries. Another area of future work is related to sentiment analysis. Sentiment analysis is a way of understanding the emotions of a customer, based on the words they use in their conversation with the AI chatbot. This technology can be used to better understand how a customer perceives a product or service, or to identify potential customer service issues before they become a problem. By implementing sentiment analysis into AI chatbot technology, customer service agents can be better equipped to understand the needs of their customers and provide better service. Finally, AI chatbot technology can be further developed to enable more efficient customer service. AI chatbots can be used to automate certain customer service tasks, such as responding to basic inquiries or providing automated follow-ups. By automating these tasks, customer service agents can be freed up to focus on more complex customer service issues. Additionally, AI chatbots can be used to generate personalized customer experiences by gathering data on each customer and providing them with tailored solutions. This will allow customer service agents to provide a more personalized and efficient customer service experience.