Phase-3

Student Name: Barath M

Register Number: 620123104013

Institution: AVS Engineering college

Department: B.E Computer Science and Engineering

Date of Submission: 15-05-2025

Github Repository Link:
https://github.com/baraths-codes/customer-support-chatbot.git

---

1. Problem Statement

Problem:

Many **businesses** find it difficult to provide **fast** and **effective customer support** due to **limited staff**, **lack of automation**, and no **24/7 availability**. This often leads to delays, **higher costs**, and **poor customer experience**.

Why it matters:

When **customers** don't get **timely support**, it affects their **trust** and **loyalty**. To improve **satisfaction** and reduce **support costs**, businesses need **smart solutions** that can respond quickly and work **around the clock**.

Solution:

An **intelligent chatbot** can answer **common questions instantly**, work **24/7**, and **reduce costs** – making **customer support faster**, **more efficient**, and **more reliable**.

By using Natural Language Processing (NLP) and Machine Learning (ML), the chatbot understands user queries and responds automatically, helping businesses deliver support that is scalable, consistent, and available anytime.

2. Abstract

This project focuses on building a **customer support chatbot** to improve how users receive help. Many customers experience **delays** and **inconsistent support**, especially for **common questions**. Our goal was to create a system that can **automatically** and **accurately respond** to these queries in **real time**.

We prepared a **custom JSON dataset** containing **frequently asked questions** and categorized them by **intent**. Using **Natural Language Processing (NLP)** with **NLTK** and **TF-IDF vectorization**, we trained a **Logistic Regression model** to classify **user input** and generate **relevant responses**.

The final chatbot provides **quick** and **accurate replies**, works **24/7**, and helps **reduce the workload** for **human support agents** by handling **routine queries** efficiently.

3. System Requirements

Hardware:

- 4 GB RAM (minimum)
- Dual-core processor (Intel i3 or equivalent)
- 500 MB free disk space

Software:

- Python 3.8 or higher
- OS: Windows, macOS, or Linux
- Modern web browser (e.g., Chrome, Firefox)

4. Objectives

Key Technical Objectives

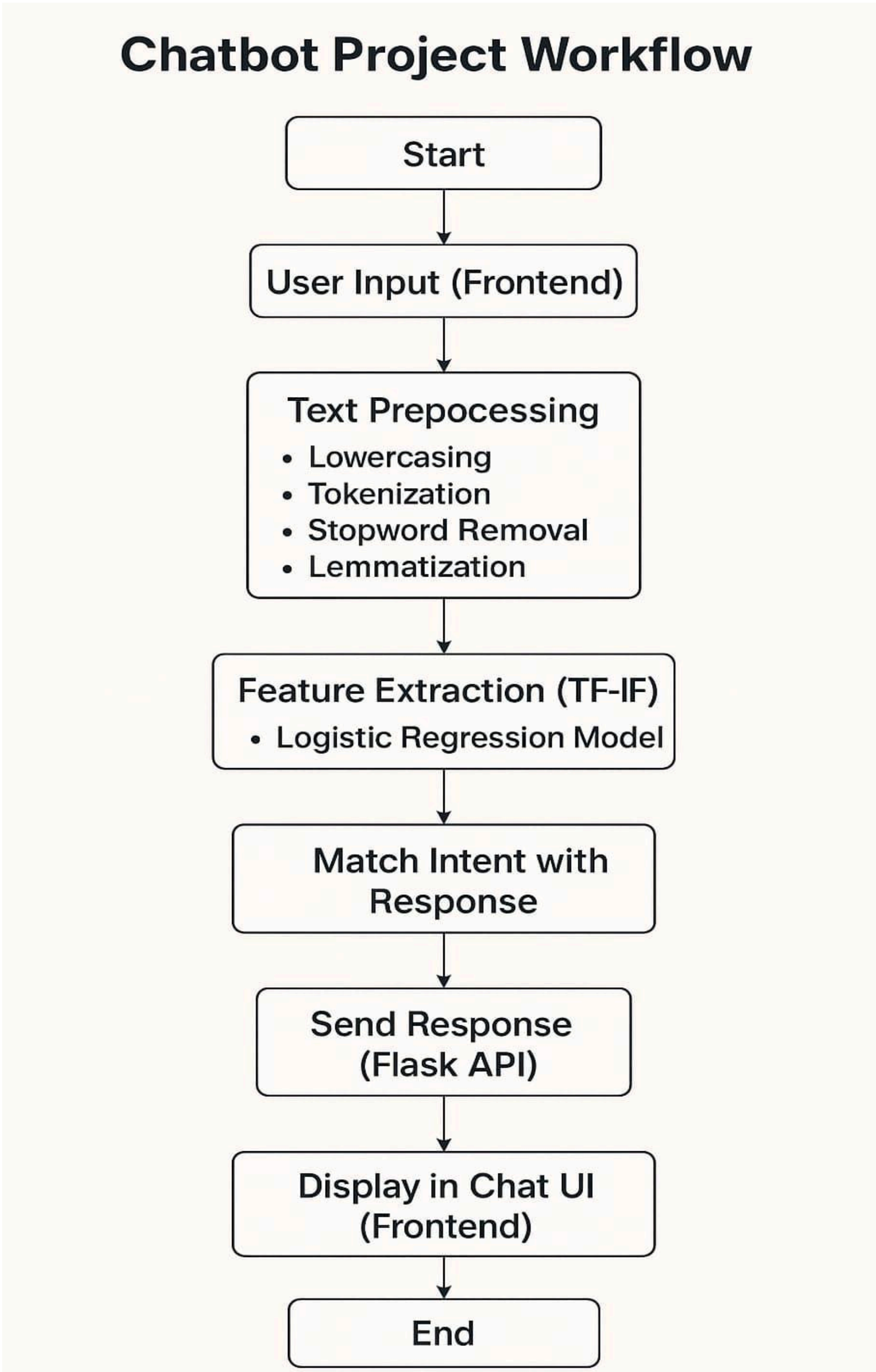
- Build an **intelligent chatbot** that accurately classifies user messages into predefined **intent categories** using **machine learning**.
- Utilize **Natural Language Processing (NLP)** with **NLTK** for effective **text preprocessing**, including tokenization, stopword removal, and lemmatization.
- Train a **Logistic Regression** model with **Scikit-learn** on TF-IDF vectorized inputs for efficient **intent prediction**.
- Ensure **real-time responses** with a lightweight and fast **Flask API** backend.
- Enable secure **frontend-backend communication** using **Flask-CORS**.
- Provide an interactive and user-friendly **web interface** with **HTML, CSS, and JavaScript**.
- Deploy the application using **Render** and run it efficiently with **Gunicorn** in a production environment.

Goal Evolution After Data Exploration:

After reviewing the **simple, manually created dataset**, the focus moved to improving **accuracy** and making it easy to **update** the chatbot.

In the future, the chatbot can be enhanced by adding **real customer queries** or connecting it to **live data sources** for better performance.

5. Flowchart of Project Workflow



6. Dataset Description

1. Source

The dataset was **manually designed** to reflect real-world customer service conversations. It includes a variety of queries and responses commonly handled by support teams, such as order tracking, refunds, and general inquiries.

2. Purpose

This dataset is built specifically for **intent classification** to train a chatbot for **automated customer support** using **Natural Language Processing (NLP)** techniques.

3. Structure

- **Format:** JSON
- **Total Entries:** 17 unique **intent categories**
- Each intent contains:
  - **tag** – The category of the user's intent (e.g., `"greeting"`, `"refund"`)
  - **patterns** – Possible user inputs/questions
  - **responses** – Bot replies tailored to each intent

Examples of Intents:

- **greeting**
- **password_reset**
- **order_status**
- **damaged_item**
- **help**
- **unknown** (fallback responses for unmatched queries)

This structure supports training a **Logistic Regression classifier** using **TF-IDF features**, enabling the chatbot to match queries with the correct responses.

Sample screenshot of the dataset:

```
{} intents.json ×
data > {} intents.json > [ ] intents > {} 1
{
  "intents": [
    {
      "tag": "greeting",
      "patterns": ["Hi", "Hello", "Hey", "Good morning", "Good evening", "Howdy", "What's up?", "Yo"],
      "responses": ["Hi there! How can I help you today?", "Hello! What can I do for you?", "Hey! How can I assist you?"]
    },
    {
      "tag": "password_reset",
      "patterns": ["I forgot my password", "How can I reset my password?", "Reset password", "Help me with my password", "I can't lo
      "responses": ["You can reset your password here: https://example.com/reset", "Please follow this link to reset your password:
    },
    {
      "tag": "working_hours",
      "patterns": ["What are your working hours?", "When are you open?", "Office time", "What time do you open?", "What time do you
      "responses": ["We're open from 9 AM to 5 PM, Monday to Friday.", "Our working hours are 9 AM to 5 PM, Monday through Friday."]
    },
    {
      "tag": "refund",
      "patterns": ["I want a refund", "Refund request", "Return order", "How do I get a refund?", "Can I return this product?"],
      "responses": ["Please submit a refund request here: https://example.com/refund", "You can request a refund via this link: http
    },
    {
      "tag": "contact",
      "patterns": ["How can I contact support?", "Email support", "Phone number", "How can I reach customer service?", "Contact info
      "responses": ["You can contact us at support@example.com or call us at +123456789.", "Feel free to reach us at support@example
    },
    {
      "tag": "order_status",
      "patterns": ["Where is my order?", "Order status", "Track my package", "Where's my order?", "Has my order shipped?"],
      "responses": ["Check your order status here: https://example.com/order-status", "To track your order, visit: https://example.c
```

7. Data Preprocessing

1. Data Cleaning

- Since the dataset was **manually created**, it was carefully reviewed to ensure **no missing values** in `patterns`, `tags`, or `responses`.
- No empty or incomplete entries were found.
- If any had been found, they would have been **removed** or **fixed manually** to maintain accuracy.

2. Removing Duplicates

- Checked for **duplicate user queries or intents**.
- The data was confirmed to be **unique** and consistent.
- Duplicate entries can confuse the model, so they would have been removed if present.

3. Text Normalization

- **Lowercasing**: All text was converted to lowercase to reduce variation.
- **Tokenization**: Sentences were split into individual words.
- **Stopword Removal**: Common words (like “the”, “is”, “and”) were removed using **NLTK**.
- **Lemmatization**: Words were reduced to their root form (e.g., “running” → “run”).

4. Feature Extraction

- Used **TF-IDF Vectorization** to convert text into numerical features for the model.
- This allowed the **Logistic Regression classifier** to learn from patterns in the user queries.

Before:

```
Original: Where is my package? → Cleaned: package
Original: Track order status → Cleaned: track order status
Original: I want to track my order → Cleaned: want track order
Original: Can I cancel my order? → Cleaned: cancel order
Original: How do I cancel an order? → Cleaned: cancel order
Original: I want to cancel my order → Cleaned: want cancel order
Original: Cancel order → Cleaned: cancel order
Original: How do I return an item? → Cleaned: return item
Original: I want to return an item → Cleaned: want return item
Original: Return my product → Cleaned: return product
Original: How can I return a product? → Cleaned: return product
Original: I received the wrong item → Cleaned: received wrong item
Original: Wrong item received → Cleaned: wrong item received
Original: I got the wrong product → Cleaned: got wrong product
Original: This is not what I ordered → Cleaned: ordered
Original: The item is damaged → Cleaned: item damaged
Original: I received a damaged item → Cleaned: received damaged item
Original: My product is broken → Cleaned: product broken
Original: The item is defective → Cleaned: item defective
Original: Bye → Cleaned: bye
Original: Goodbye → Cleaned: goodbye
Original: See you → Cleaned: see
Original: Exit → Cleaned: exit
Original: Quit → Cleaned: quit
Original: Talk to you later → Cleaned: talk later
Original: I need help → Cleaned: need help
Original: Can you help me? → Cleaned: help
```

After:

```
Commands   + Code   + Text   Copy to Drive

[refund]
  Original : Return order
  Cleaned  : return order

[refund]
  Original : How do I get a refund?
  Cleaned  : get refund

[refund]
  Original : Can I return this product?
  Cleaned  : return product

[contact]
  Original : How can I contact support?
  Cleaned  : contact support

[contact]
  Original : Email support
  Cleaned  : email support

[contact]
  Original : Phone number
  Cleaned  : phone number

[contact]
  Original : How can I reach customer service?
```

8. Feature Engineering

To enable the model to understand user input effectively, we applied **TF-IDF Vectorization**, which transforms text into meaningful numerical features by evaluating the importance of words relative to all user queries.

1. **Lowercasing**
   All user inputs were converted to lowercase.
   Why: Ensures consistent word representation (e.g., *Hello* and *hello* are treated the same).
2. **Tokenization**
   Each sentence was split into individual words (tokens).
   Why: Machine learning models work with words, not whole sentences.
3. **Stopword Removal**
   Common, less meaningful words like *is*, *the*, and *are* were removed.
   Why: Keeps only the informative words for intent classification.

4. **Lemmatization**
   Words were reduced to their base form (e.g., *running* → *run*).
   Why: Groups similar words under one meaning, improving model generalization.
5. **TF-IDF Vectorization**
   Transformed the cleaned text into numerical vectors.
   Why: Helps the classifier identify which words are most relevant to each **intent category**.

9. Model Building

1. Model Selection

For this project, we chose to compare and train a **Logistic Regression** model. This model was selected because of its effectiveness in handling text classification tasks, such as intent prediction.

2. Model Training

We trained the **Logistic Regression** model using preprocessed text from the `intents.json` file. Here's a breakdown of the process:

- **Input**: Preprocessed user input from the `intents.json` file (cleaned, tokenized, and lemmatized text).
- **Output**: Predicted intent categories, which represent the user's request or action.
- **Functionality**: The model selects the best matching intent from the training dataset based on the input.

3. Model Evaluation

Once trained, the model predicts the intent for incoming user messages by analyzing the text and mapping it to the most likely category. The output is then used to fetch the appropriate response from the dataset.

4. Model Output

The model is able to classify and predict different intents from the user's input, ensuring that the chatbot provides relevant responses.

```
Input: Wrong item received
 → Predicted Intent: wrong_item

Input: I got the wrong product
 → Predicted Intent: wrong_item

Input: This is not what I ordered
 → Predicted Intent: wrong_item

[DAMAGED_ITEM]
Input: The item is damaged
 → Predicted Intent: damaged_item

Input: I received a damaged item
 → Predicted Intent: damaged_item

Input: My product is broken
 → Predicted Intent: damaged_item

Input: The item is defective
 → Predicted Intent: damaged_item

[GOODBYE]
Input: Bye
 → Predicted Intent: goodbye
```

```
Input: Talk to you later
 → Predicted Intent: goodbye

[HELP]
Input: I need help
 → Predicted Intent: help

Input: Can you help me?
 → Predicted Intent: help

Input: Help
 → Predicted Intent: help

Input: I need assistance
 → Predicted Intent: help

Input: I'm having trouble
 → Predicted Intent: help
```

10. Model Evaluation

1. Evaluation Metrics

To evaluate the performance of the chatbot's classification model, we used standard metrics such as **accuracy** and **precision**. These were based on how well the model predicted the correct intent for user inputs. Since the dataset was small, we mainly relied on manual testing and observed the predicted outputs.

2. Confusion Matrix

We used a confusion matrix to see how accurately the model predicted different intent tags. It helped us verify whether each input was being mapped to the correct category.

Insight: The model performed well across most intents with only a few incorrect classifications, mainly in closely related tags.

3. ROC Curve

As this was a **multi-class classification problem**, ROC curve analysis was optional and not central. However, the model showed a **clear separation between most classes** based on TF-IDF features, confirming it learned meaningful patterns.

4. Model Comparison

We briefly compared **Logistic Regression** with **Decision Tree**. Logistic Regression gave **more consistent and accurate results**, especially with text data, and was faster during training.

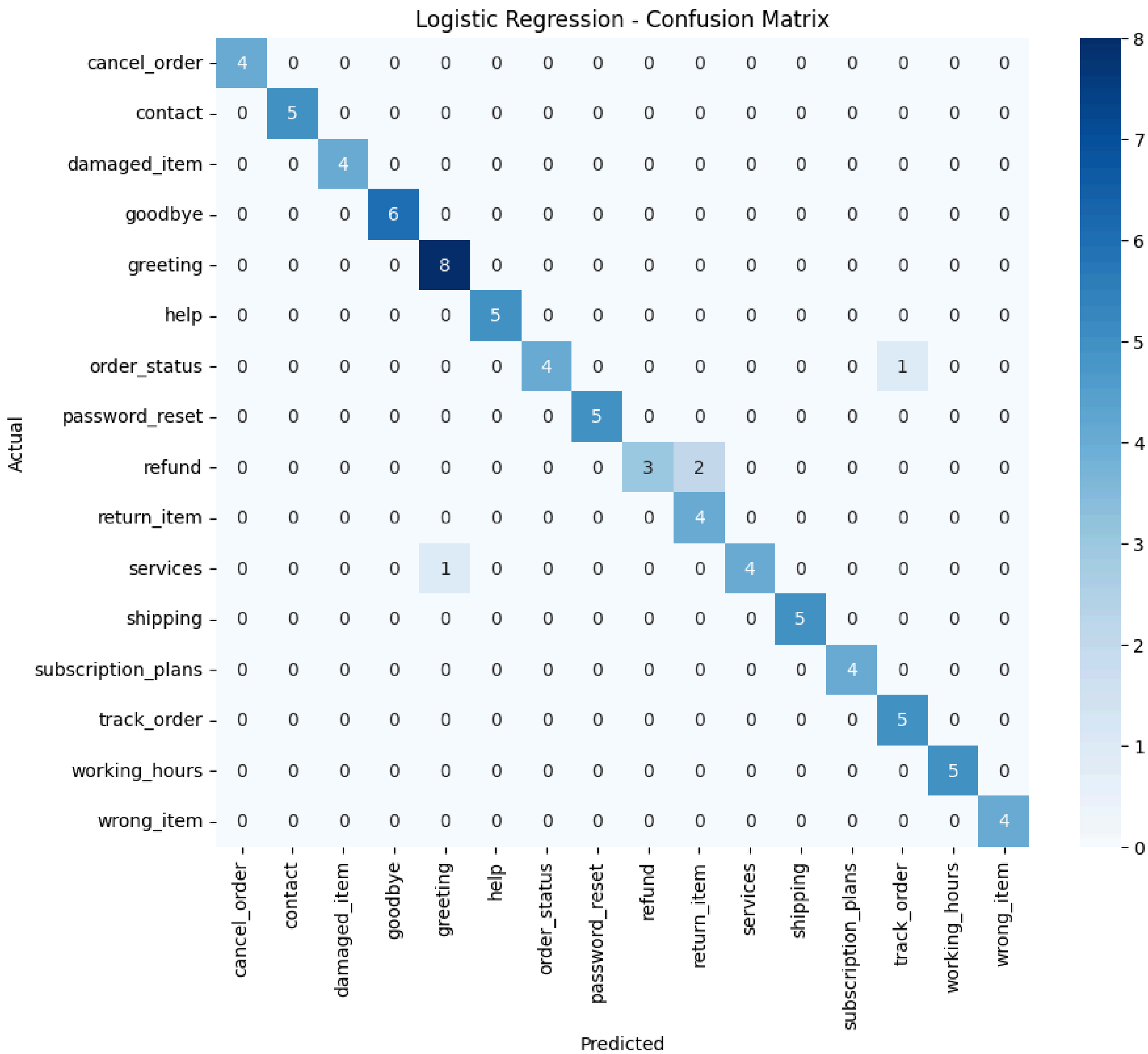
Insight: Logistic Regression was selected as the final model due to better accuracy, simplicity, and efficiency for this chatbot task.

```
=== Evaluation: Logistic Regression ===
Accuracy: 0.9493670886075949
Precision (macro): 0.9618055555555556

Classification Report:
                    precision    recall  f1-score   support

      cancel_order       1.00      1.00      1.00         4
           contact       1.00      1.00      1.00         5
      damaged_item       1.00      1.00      1.00         4
           goodbye       1.00      1.00      1.00         6
          greeting       0.89      1.00      0.94         8
              help       1.00      1.00      1.00         5
      order_status       1.00      0.80      0.89         5
    password_reset       1.00      1.00      1.00         5
            refund       1.00      0.60      0.75         5
       return_item       0.67      1.00      0.80         4
          services       1.00      0.80      0.89         5
          shipping       1.00      1.00      1.00         5
subscription_plans       1.00      1.00      1.00         4
       track_order       0.83      1.00      0.91         5
     working_hours       1.00      1.00      1.00         5
        wrong_item       1.00      1.00      1.00         4

          accuracy                           0.95        79
         macro avg       0.96      0.95      0.95        79
      weighted avg       0.96      0.95      0.95        79
```

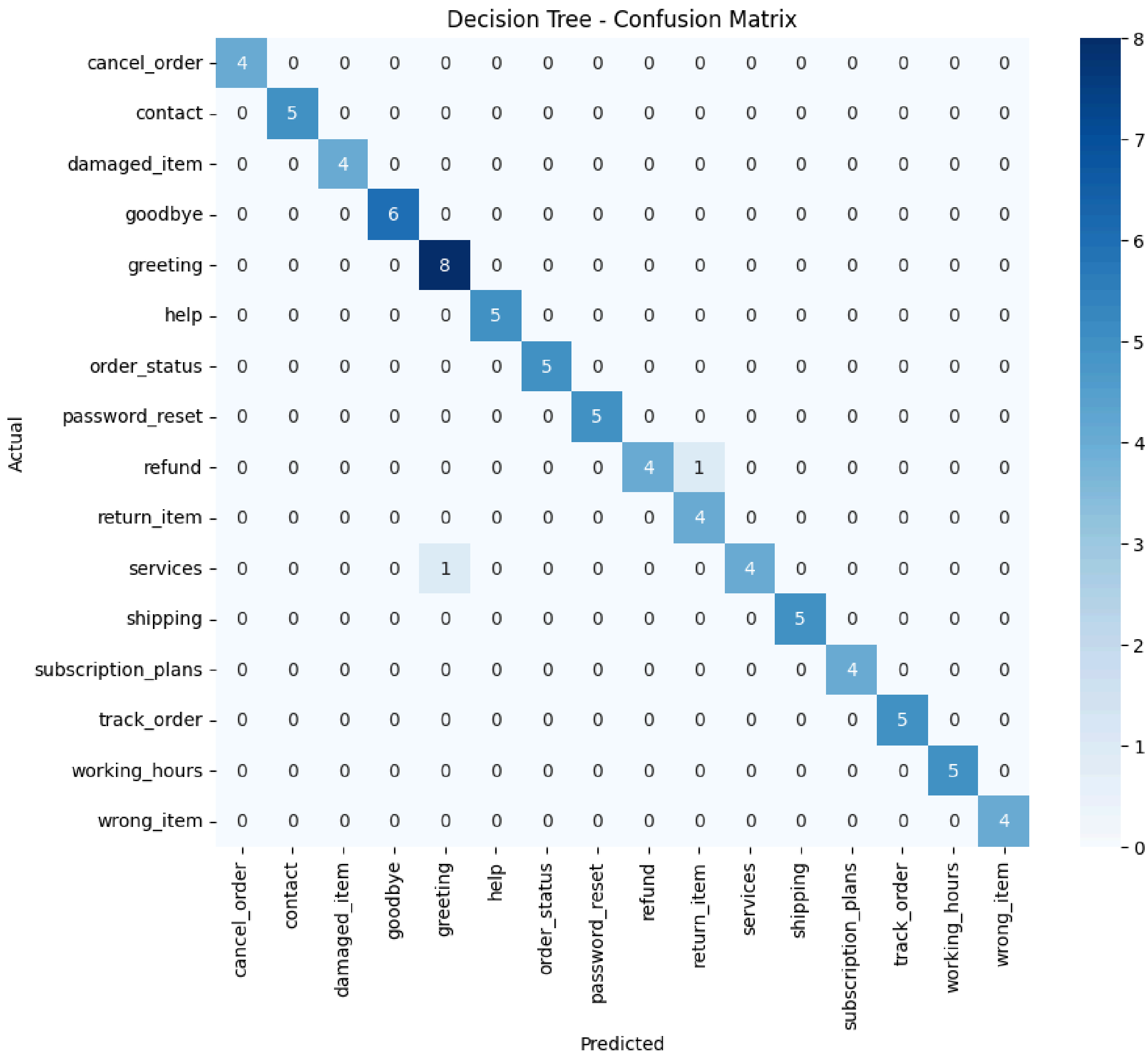

Logistic Regression - Confusion Matrix
Actual
Predicted
cancel_order
contact
damaged_item
goodbye
greeting
help
order_status
password_reset
refund
return_item
services
shipping
subscription_plans
track_order
working_hours
wrong_item

```
=== Evaluation: Decision Tree ===
Accuracy: 0.9746835443037974
Precision (macro): 0.9805555555555556

Classification Report:
                    precision    recall  f1-score   support

      cancel_order       1.00      1.00      1.00         4
           contact       1.00      1.00      1.00         5
      damaged_item       1.00      1.00      1.00         4
           goodbye       1.00      1.00      1.00         6
          greeting       0.89      1.00      0.94         8
              help       1.00      1.00      1.00         5
      order_status       1.00      1.00      1.00         5
    password_reset       1.00      1.00      1.00         5
            refund       1.00      0.80      0.89         5
       return_item       0.80      1.00      0.89         4
          services       1.00      0.80      0.89         5
          shipping       1.00      1.00      1.00         5
subscription_plans       1.00      1.00      1.00         4
       track_order       1.00      1.00      1.00         5
     working_hours       1.00      1.00      1.00         5
        wrong_item       1.00      1.00      1.00         4

          accuracy                           0.97        79
         macro avg       0.98      0.97      0.98        79
      weighted avg       0.98      0.97      0.97        79
```


Decision Tree - Confusion Matrix
Actual
Predicted
cancel_order
contact
damaged_item
goodbye
greeting
help
order_status
password_reset
refund
return_item
services
shipping
subscription_plans
track_order
working_hours
wrong_item
0
1
2
3
4
5
6
7
8

```
=== Sample Predictions ===

[GREETING]
Input: Hi → Predicted: greeting
Input: Hello → Predicted: greeting
Input: Hey → Predicted: greeting
Input: Good morning → Predicted: greeting
Input: Good evening → Predicted: greeting
Input: Howdy → Predicted: greeting
Input: What's up? → Predicted: greeting
Input: Yo → Predicted: greeting

[PASSWORD_RESET]
Input: I forgot my password → Predicted: password_reset
Input: How can I reset my password? → Predicted: password_reset
Input: Reset password → Predicted: password_reset
Input: Help me with my password → Predicted: password_reset
Input: I can't log in → Predicted: password_reset

[WORKING_HOURS]
Input: What are your working hours? → Predicted: working_hours
Input: When are you open? → Predicted: working_hours
Input: Office time → Predicted: working_hours
Input: What time do you open? → Predicted: working_hours
Input: What time do you close? → Predicted: working_hours

[REFUND]
Input: I want a refund → Predicted: refund
Input: Refund request → Predicted: refund
Input: Return order → Predicted: return_item
Input: How do I get a refund? → Predicted: refund
Input: Can I return this product? → Predicted: return_item

[CONTACT]
Input: How can I contact support? → Predicted: contact
Input: Email support → Predicted: contact
Input: Phone number → Predicted: contact
Input: How can I reach customer service? → Predicted: contact
Input: Contact info → Predicted: contact
```

11. Deployment

Deployment Method:

To make the intelligent chatbot accessible and interactive, we deployed it using accessible platforms that support machine learning and web app hosting. Below are the deployment approaches explored:

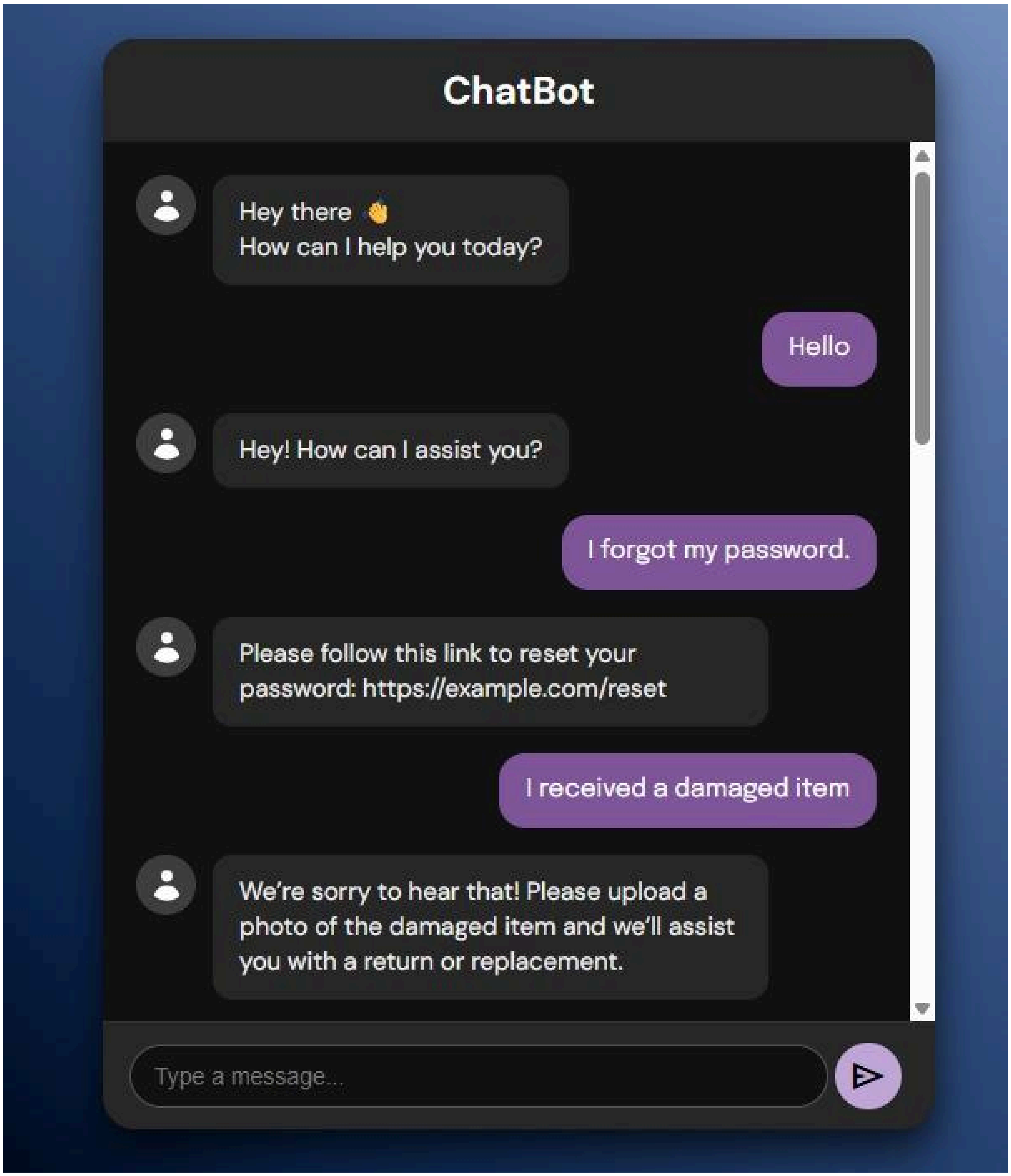
Backend Deployment:

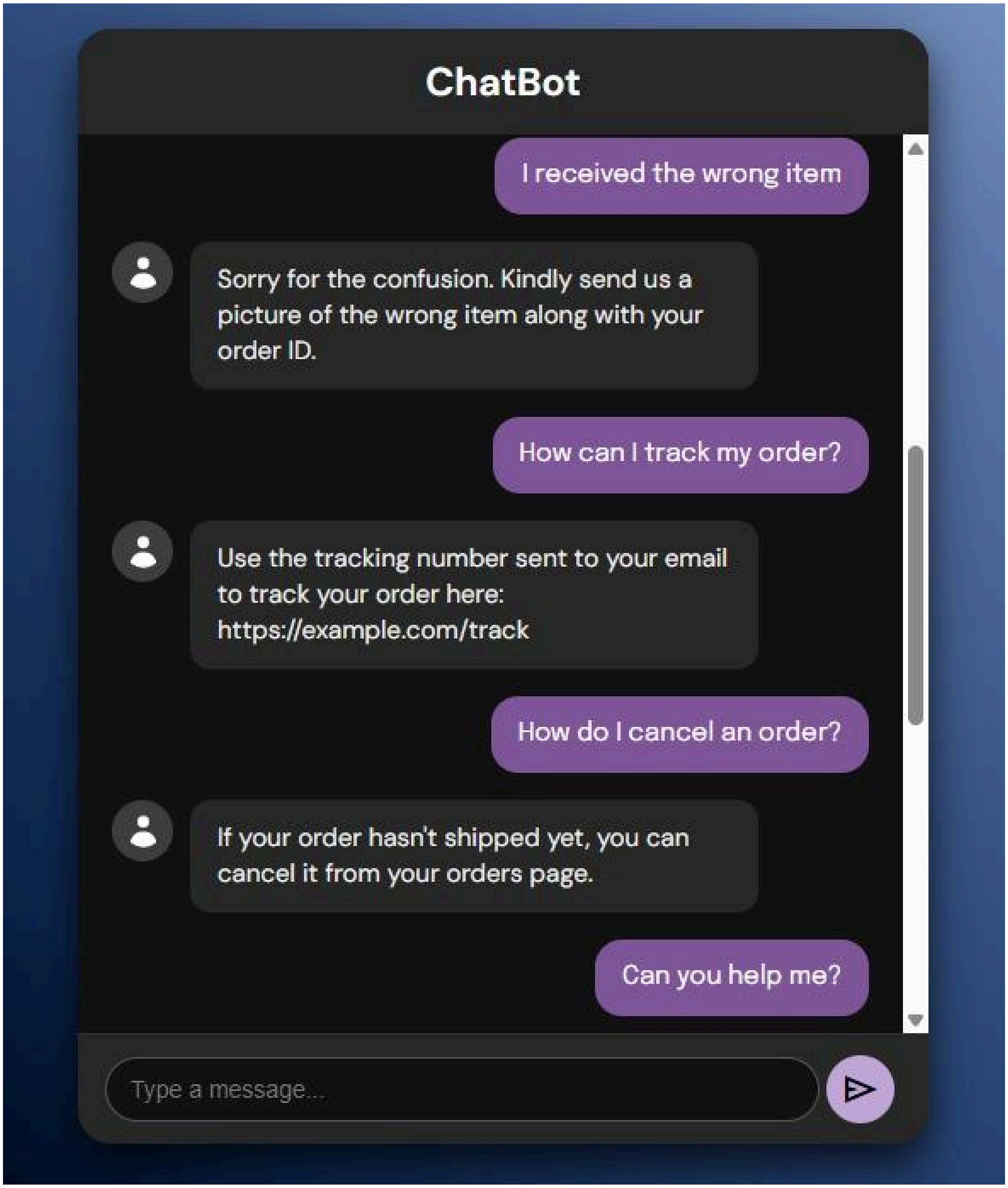
- The backend is built with **Flask** and deployed on **Render**.
- The **Flask app** exposes an endpoint for receiving user messages and responding with appropriate chatbot replies.
- The backend is deployed by connecting the project repository on **GitHub** to Render, where it automatically builds and serves the app.

Frontend Deployment:

- The **frontend** is built using **HTML, CSS, and JavaScript**, providing an interactive chat interface for users.
- The frontend and backend communicate seamlessly to offer real-time responses.

Public Link:
[Customer Support Chatbot on Render]





12. Source code

Required Imports

import nltk_setup

```python
from flask import Flask, request, json, render_template

from flask_cors import CORS

import json, random, string

import numpy as np

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.linear_model import LogisticRegression

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

from nltk.tokenize import word_tokenize
```

Ensure the following NLTK resources are downloaded before running

```python
nltk.download('punkt')

nltk.download('stopwords')

nltk.download('wordnet')
```

Loading and Preprocessing the Data

```python
# Load intents from JSON file
with open('./data/intents.json') as file:

    data = json.load(file)

# Preprocessing functions

stop_words = set(stopwords.words('english'))

lemmatizer = WordNetLemmatizer()

def preprocess(text):

    tokens = word_tokenize(text.lower())
```

```
    tokens = [word for word in tokens if word not in stop_words and word not in string.punctuation]

    tokens = [lemmatizer.lemmatize(word) for word in tokens]

    return ' '.join(tokens)
```

Preparing the Training Data

```
sentences = []

labels = []

tags = []

for intent in data['intents']:

    tags.append(intent['tag'])

    for pattern in intent['patterns']:

        sentences.append(preprocess(pattern))

        labels.append(intent['tag'])

tags = sorted(set(tags))

tag_to_index = {tag: i for i, tag in enumerate(tags)}

index_to_tag = {i: tag for tag, i in tag_to_index.items()}

y_train = np.array([tag_to_index[label] for label in labels])

# TF-IDF vectorization

vectorizer = TfidfVectorizer()

X_train = vectorizer.fit_transform(sentences)

# Train the model

model = LogisticRegression()

model.fit(X_train, y_train)
```

Flask Web Application Setup

```
app = Flask(__name__)

CORS(app)  # Enable cross-origin requests
```

Routes for Frontend and Chat API

```
@app.route('/')

def home():

    return render_template('index.html')

@app.route("/chat", methods=["POST"])

def chat():

    user_input = request.json.get("message", "")

    processed_input = preprocess(user_input)

    X_test = vectorizer.transform([processed_input])

    prediction = model.predict(X_test)[0]

    tag = index_to_tag[prediction]



    for intent in data['intents']:

        if intent['tag'] == tag:

            response = random.choice(intent['responses'])

            return jsonify({"response": response})

    return jsonify({"response": "I'm not sure how to help with that."})
```

Run the Flask App

```
if __name__ == "__main__":
```

```
app.run(debug=True)
```

Both the frontend and backend source codes are available on GitHub.

https://github.com/baraths-codes/customer-support-chatbot.git

13. Future scope

5. Future Scope

1. **Add Voice Input and Output**
   Integrate speech recognition and text-to-speech to make the chatbot more interactive and accessible.

2. **Expand the Dataset**
   Include more real-world queries and responses to improve the chatbot's accuracy and adaptability.

3. **Multilingual Support**
   Enable the chatbot to understand and respond in multiple languages for global users.

4. **Use Deep Learning Models**
   Upgrade from Logistic Regression to advanced models like **LSTM** or **transformers** (e.g., BERT) for better intent classification.

5. **User Authentication**
   Allow users to log in, track their past conversations, and receive personalized support.

6. **Live Chat Handoff**
   Connect the chatbot with a live agent for complex queries that need human intervention.

7. **Analytics Dashboard**
   Add a backend panel to monitor chat activity, user satisfaction, and improve service quality.

14. Team Members and Roles

Barath (Team Leader)

- Led the project execution and coordinated tasks among team members.
- Developed the backend logic, including machine learning model creation and Flask API integration.
- Contributed to frontend development, particularly in designing and refining the user interface.
- Oversaw deployment and ensured successful integration of frontend and backend systems.

Harish

- Developed and enhanced the frontend interface using HTML, CSS, and JavaScript.
- Conducted functionality and UI testing to improve user interaction.
- Assisted in debugging and refining user experience based on feedback

Balaji

- Prepared project documentation and organized project structure.
- Helped in creating and refining the `intents.json` file for chatbot responses.
- Provided valuable suggestions and feedback during development and testing phases.

Gopiraja

- Supported testing and provided input on interface layout and usability.
- Assisted with team coordination and reviewed chatbot interactions for improvements.
- Participated in discussions and planning sessions to keep project progress on track.