**PHASE 5**

**Problem statement:**

The problem statement for the "Chatbot with Python" project revolves around the development of a conversational chatbot using the Python programming language. The primary objective of this project is to create a versatile and intelligent chatbot that can engage in natural language conversations, understand user intent, and provide relevant responses.

**Design Thinking:**

**Empathize:**

Start by understanding the needs and pain points of potential users. Conduct user research through surveys, interviews, and observation to gain insights into what users expect from a chatbot.Identify common use cases and user personas to understand the diverse needs and preferences of your audience.

**Define:**

Define the problem and set clear goals for your chatbot project. What specific issues will the chatbot address, and what objectives will it fulfill?Create user stories and define the scope of the chatbot's capabilities, including the topics it will cover, and the level of interaction it should support.

**Ideate:**

Brainstorm creative ideas for your chatbot's functionality and features. Encourage your team to generate as many ideas as possible.Consider innovative ways to make the chatbot engaging and helpful. Think about incorporating AI-driven features, personalization, and context-aware responses.

**Prototype:**

Create a preliminary design and user interface for your chatbot. This could include wireframes, mockups, or even a simple working prototype to visualize the user experience.Define the chatbot's conversational flow, including how it will respond to different user inputs and intents.

**Test:**

Begin testing your chatbot with real users. Collect feedback on the prototype and iterate based on user input.Pay close attention to user satisfaction, ease of use, and whether the chatbot meets their needs. Make improvements accordingly.

**Implement:**

Develop the chatbot using Python and relevant NLP libraries. Implement the chatbot's core functionality, including natural language understanding, intent recognition, and response generation.Build a user interface, either as a web application or a chat platform integration, to allow users to interact with the chatbot.

**Refine and Iterate:**

Continue testing and gathering user feedback even after the initial implementation. This iterative process ensures that the chatbot evolves to better meet user needs.Make refinements to the chatbot's functionality, fine-tune its responses, and address any issues that arise.

**Deploy and Monitor:**

Deploy the chatbot to a hosting environment or the desired platform, making it accessible to users.Implement monitoring and analytics to gather data on how the chatbot is performing, and use this information to make ongoing improvements.

**Document and Communicate:**

Create comprehensive documentation for developers and users. Explain how the chatbot works, its features, and how users can interact with it.Communicate the availability of the chatbot to your target audience through marketing and promotion efforts.

**Development phase 1:**

**Create a Chatbot with Python and Machine Learning**

To create a chatbot with Python and Machine Learning, you need to install some packages. All the packages you need to install to create a chatbot with Machine Learning using the Python programming language are mentioned below:

tensorflow==2.3.1

nltk==3.5

colorama==0.4.3

numpy==1.18.5

scikit\_learn==0.23.2

Flask==1.1.2

**Data preparation:**

The second step of this task to create a chatbot with Python and Machine Learning is to prepare the data to train our chatbot. I’ll start this step by importing the necessary libraries and packages:

import json

import numpy as np

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Embedding, GlobalAveragePooling1D

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from sklearn.preprocessing import LabelEncoder

**Now I will read the JSON file and process the required files:**

with open('intents.json') as file:

data = json.load(file)

training\_sentences = []

training\_labels = []

labels = []

responses = []

for intent in data['intents']:

for pattern in intent['patterns']:

training\_sentences.append(pattern)

training\_labels.append(intent['tag'])

responses.append(intent['responses'])

if intent['tag'] not in labels:

labels.append(intent['tag'])

num\_classes = len(labels)

**Now we need to use the label encoder method provided by the Scikit-Learn library in Python:**

lbl\_encoder = LabelEncoder()

lbl\_encoder.fit(training\_labels)

training\_labels = lbl\_encoder.transform(training\_labels)

**Tokenization:**

Now we need to vectorize the data using the Tokenization method to create a chatbot with Python and Machine Learning:

vocab\_size = 1000

embedding\_dim = 16

max\_len = 20

oov\_token = "<OOV>"

tokenizer = Tokenizer(num\_words=vocab\_size, oov\_token=oov\_token)

tokenizer.fit\_on\_texts(training\_sentences)

word\_index = tokenizer.word\_index

sequences = tokenizer.texts\_to\_sequences(training\_sentences)

padded\_sequences = pad\_sequences(sequences, truncating='post', maxlen=max\_len)

**Training a Neural Network**

Now the next and most important step in the process of building a chatbot with Python and Machine Learning is to train a neural network. Now, I will train and create a neural network to train our chatbot:

model = Sequential()

model.add(Embedding(vocab\_size, embedding\_dim, input\_length=max\_len))

model.add(GlobalAveragePooling1D())

model.add(Dense(16, activation='relu'))

model.add(Dense(16, activation='relu'))

model.add(Dense(num\_classes, activation='softmax'))

model.compile(loss='sparse\_categorical\_crossentropy',

optimizer='adam', metrics=['accuracy'])

model.summary()

epochs = 500

history = model.fit(padded\_sequences, np.array(training\_labels), epochs=epochs)

**Saving The Neural Network:**

We’ve trained the model, but before we go any further in the process of building a chatbot with Python and Machine Learning, let’s save the model so that we can use this neural network in the future as well:

# to save the trained model

model.save("chat\_model")

import pickle

# to save the fitted tokenizer

with open('tokenizer.pickle', 'wb') as handle:

pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST\_PROTOCOL)

# to save the fitted label encoder

with open('label\_encoder.pickle', 'wb') as ecn\_file:

pickle.dump(lbl\_encoder, ecn\_file, protocol=pickle.HIGHEST\_PROTOCOL)

**Now let’s Build a Chatbot with Python and our Trained Machine Learning Model**

Now I am going to implement a chat function to interact with a real user. When the message from the user will be received, the chatbot will compute the similarity between the sequence of the new text and the training data.Taking into account the trust scores obtained for each category, it categorizes the user’s message according to an intention with the highest trust score:

import json

import numpy as np

from tensorflow import keras

from sklearn.preprocessing import LabelEncoder

import colorama

colorama.init()

from colorama import Fore, Style, Back

import random

import pickle

with open("intents.json") as file:

data = json.load(file)

def chat():

# load trained model

model = keras.models.load\_model('chat\_model')

# load tokenizer object

with open('tokenizer.pickle', 'rb') as handle:

tokenizer = pickle.load(handle)

# load label encoder object

with open('label\_encoder.pickle', 'rb') as enc:

lbl\_encoder = pickle.load(enc)

# parameters

max\_len = 20

while True:

print(Fore.LIGHTBLUE\_EX + "User: " + Style.RESET\_ALL, end="")

inp = input()

if inp.lower() == "quit":

break

result = model.predict(keras.preprocessing.sequence.pad\_sequences(tokenizer.texts\_to\_sequences([inp]),

truncating='post', maxlen=max\_len))

tag = lbl\_encoder.inverse\_transform([np.argmax(result)])

for i in data['intents']:

if i['tag'] == tag:

print(Fore.GREEN + "ChatBot:" + Style.RESET\_ALL , np.random.choice(i['responses']))

# print(Fore.GREEN + "ChatBot:" + Style.RESET\_ALL,random.choice(responses))

print(Fore.YELLOW + "Start messaging with the bot (type quit to stop)!" + Style.RESET\_ALL)

chat()

**Development phase 2:**

**Project Setup:**

**1. Setting up the Development Environment**

Begin by ensuring that your development environment is prepared for the project.

Verify the presence of essential tools and libraries, with a specific focus on Python

and Flask. Install them if necessary to establish a suitable working environment.

**2. Saving the Chatbot Code**

The core of the project resides in a Python file, typically named chatbot.py. This file houses the chatbot&#39;s functionality, where it can interpret and respond to user messages. The chatbot should incorporate a function that receives user input and generates appropriate responses based on its internal logic.

**Flask Integration:**

**3. Installing Flask in Google Colab**

In the Google Colab environment, Flask must be installed to facilitate the creationof the web application. Use a straightforward pip installation command to addFlask to your projects dependencies.

**4. Creating a Flask Web App**

Within a Python file named app.py, commence the creation of a Flask web application. This application will serve as the central hub for handling user requests, communicating with the chatbot&#39;s logic, and providing a platform for the web interface.

**5**. **Handling Routes and Endpoints**

Define the routes and endpoints within the Flask app to orchestrate user interactions. These routes serve various purposes, including rendering the web interface and managing the chatbot interactions, such as receiving user messages and delivering chatbot responses.

**Web Interface**

**6. Creating the Web Interface (HTML)**

An interactive web interface is pivotal for enabling users to communicate with thechatbot. Create an HTML file (commonly named index.html) that serves as theuser interface. This HTML file should incorporate input fields for user messages, achat history display, and elements that create a user-friendly chat experience

**Interaction with the Chatbot**

**7. Sending User Messages**

Through the web interface, users have the capability to input their messages. Thesemessages are then dispatched to the Flask app for processing and communicationwith the chatbot.

**8. Chatbot Response**

Within the Flask app, user messages are processed and delivered to the chatbots logic. The chatbot generates responses based on the user&#39;s input and returns these responses to the Flask app for transmission to the web interface.

**9. Displaying Responses**The web interface plays a critical role in conveying chatbot responses to users. Responses from the chatbot are dynamically displayed, fostering a conversational flow within the interface and creating a seamless user experience.

**Running the Application**

**10. Starting the Flask App**

With your Flask applications code in place, it is time to initiate the app. This step involves running your Flask app within your Google Colab environment or on your local machine, ensuring it is accessible and operational.

**11. Accessing the Web Interface**

To engage with the chatbot, users access the web interface by opening a web browser and navigating to the designated URL, typically http://localhost:5000.This is where they interact with the chatbot and initiate conversations.

**Code for the chatbot by integrating it into a web app using flask:**

# Install Flask

!pip install flask

# Import necessary modules

from flask import Flask, render\_template, request

from chatbot import generate\_response # Assuming you have this function

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

# Route to handle chatbot logic

@app.route('/ask', methods=['POST'])

def ask():

user\_message = request.form['user\_message']

chatbot\_response = generate\_response(user\_message)

return chatbot\_response

# Main route to serve the HTML page

@app.route('/')

def home():

return render\_template('index.html')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000)

**Now, let us create the index.html file for the chat interface:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

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

<title>Chatbot Web Interface</title>

</head>

<body>

<h1>Chatbot</h1>

<div id="chatbox">

<div id="chat"></div>

<input type="text" id="user\_input" placeholder="Type your message..." autocomplete="off">

<button onclick="sendMessage()">Send</button>

</div>

<script>

function sendMessage() {

const userMessage = document.getElementById('user\_input').value;

document.getElementById('chat').innerHTML += '<div><strong>You:</strong> ' + userMessage + '</div>';

document.getElementById('user\_input').value = '';

fetch('/ask', {

method: 'POST',

headers: {

'Content-Type': 'application/x-www-form-urlencoded',

},

body: 'user\_message=' + userMessage

})

.then(response => response.text())

.then(data => {

document.getElementById('chat').innerHTML += '<div><strong>Chatbot:</strong> ' + data + '</div>';

});

}

</script>

</body></html>

**Libraries Used:**

**1. Python:**

Python is the primary programming language for building the chatbot and integrating NLP techniques. It offers a wide range of libraries and tools that make NLP and machine learning accessible.

**2. NLTK (Natural Language Toolkit):**

NLTK is a powerful library for working with human language data. It provides easy-to-use interfaces to perform tasks like tokenization, stemming, tagging, parsing, and more. NLTK is widely used for pre-processing and text analysis in NLP.

**3. spaCy**:

spaCy is a popular library for performing advanced NLP tasks such as part-of-speech tagging, entity recognition, and dependency parsing. It is known for its speed and efficiency, making it a good choice for chatbots that require real-time responses.

**4. Gensim:**

Gensim is a library for topic modeling and document similarity analysis. It is valuable for chatbots that need to understand context and semantics in text, making it useful for more advanced conversational AI.

**5. Scikit-Learn:**

Scikit-Learn is a machine learning library that is often used for training and implementing machine learning models, including those used in chatbots for sentiment analysis, intent recognition, and response generation.

**6. TensorFlow or PyTorch**:

For building deep learning models, such as recurrent neural networks (RNNs) or transformer models, TensorFlow and PyTorch are commonly used libraries. These libraries are essential for more advanced chatbots capable of generating human-like responses.

**Integration of NLP Techniques**:

**1. Text Pre-processing**:

The first step in building an NLP-powered chatbot is to pre-process user input. This includes tasks like tokenization (splitting text into words or sentences), lowercasing, and removing punctuation and stop words.

**2. Intent Recognition:**

Intent recognition is a critical component of chatbots. Machine learning models are trained to classify user input into specific intents, such as asking a question, making a request, or expressing sentiment. Common techniques include training classifiers or using pre-trained models.

**3. Named Entity Recognition (NER):**

NER is used to identify entities like names of people, places, or organizations in user input. It's crucial for understanding user queries and providing context-aware responses.

**4. Sentiment Analysis**:

Sentiment analysis determines the emotional tone of user messages, helping the chatbot understand the user's mood. Machine learning models are trained to classify text as positive, negative, or neutral.

**5. Response Generation:**

Response generation is the core of chatbot interactions. This involves training models like sequence-to-sequence models or transformer-based models to generate coherent and contextually relevant responses.

**6. Context Management:**

NLP chatbots maintain conversation context, allowing them to keep track of the dialogue history and provide context-aware responses. Context is often managed using memory or state management techniques.

**7. Continuous Learning:**

Some chatbots use reinforcement learning techniques to improve their performance over time. They learn from user interactions and adapt to provide better responses with more user engagement.The integration of these libraries and NLP techniques enables the chatbot to understand and respond to user messages effectively. Machine learning models, coupled with NLP, allow the chatbot to analyze, process, and generate human-like responses in natural language, making it a valuable tool for various applications, including customer support, virtual assistants, and information retrieval systems.

**Interaction with Users:**

**User Input**:

The interaction begins when a user sends a message or query to the chatbot. This input can be in the form of text, voice, or other modes of communication, depending on the chatbot's capabilities.

**User Message Processing:**

Message Parsing: The chatbot first parses the user's message to understand its structure. This typically involves tasks like tokenization (splitting the message into words or phrases) and entity recognition (identifying important elements like names or dates).

**Intent Recognition:** The chatbot then determines the user's intent or purpose behind the message. It categorizes the message into specific intents, such as asking a question, making a request, expressing sentiment, or seeking information.

**Context Management:** If the chatbot has context-aware capabilities, it keeps track of the conversation history. This means it remembers the user's previous messages and uses them to provide contextually relevant responses. Context management is crucial for maintaining a coherent conversation.

**Response Generation:** Based on the user's intent and the context of the conversation, the chatbot generates a response. This response can be a simple text reply, a link to a web page, or even an action to perform within a web application.

**Response Delivery**: The chatbot sends the response back to the user, which can be in the form of a text message, voice reply, or another appropriate medium.

**Feedback Processing**: If the user provides feedback or further input, the chatbot may need to reprocess and adjust its response accordingly, creating an iterative conversation.

**Interaction with the Web Application:**

Web Application Interface: The web application may have an interface or API (Application Programming Interface) to communicate with external systems like chatbots. This interface allows the chatbot to send and receive data from the web application.

**User Requests**: If the user's message requires the chatbot to interact with the web application, the chatbot sends a request to the web application's interface. This request typically includes details about what action needs to be performed within the application.

**Web Application Processing**: The web application processes the request from the chatbot, performing the necessary actions as specified. This can include querying a database, retrieving information, performing operations, or triggering events within the application.

**Response from Web Application**: The web application generates a response based on the request from the chatbot. This response can be data, information, or a confirmation of the action taken.

**Response Delivery to User:** The chatbot receives the response from the web application and, if required, incorporates it into the conversation with the user. This allows the chatbot to provide the user with the information or results obtained from the web application.

**Data Exchange**: The chatbot and web application may continue to exchange data and instructions as the conversation evolves. This can involve multiple rounds of interaction and processing.The interaction between a chatbot and a web application is a dynamic and iterative process. The chatbot acts as a conversational interface for the user, allowing them to access and control the functionality of the web application seamlessly. This interaction enables users to perform tasks, retrieve information, and receive assistance in a conversational and user-friendly manner.