

Developing NLP-based ChatBot using DeepLearning:

Chatbots come in various types, each designed to serve different purposes and interact with users in specific ways. Here are some common types:

1. **Rule-Based Chatbots** - rely on predefined rules and patterns to generate responses, making them suitable for simple use cases but limited in their ability to understand natural language beyond the predefined rules.
2. **Retrieval-Based Chatbots** - utilize predefined responses from a dataset or knowledge base, leveraging techniques such as keyword matching, TF-IDF, or word embeddings to analyze user inputs and select appropriate responses based on matching patterns or keywords.
3. **Generative Chatbots** - utilize techniques like sequence-to-sequence models, RNNs, or transformer models to generate original and contextually relevant responses, but their training requires significant data and computational resources.
4. **Hybrid Chatbots** - leverage a combination of rule-based or retrieval-based approaches for specific scenarios while integrating generative models to handle complex or ambiguous user inputs, achieving a balance between response flexibility and accuracy.

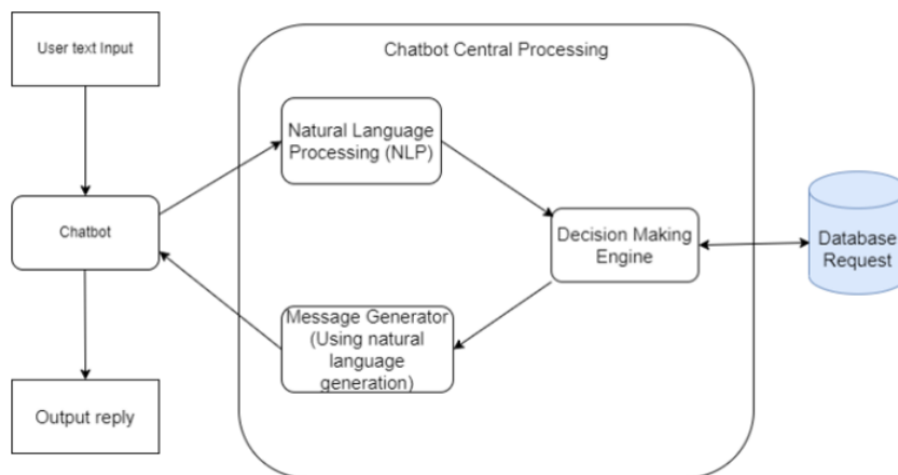
In this , we developed a retrieval-based chatbot using Natural Language Processing (NLP).

So, Natural Language Processing (NLP) is a branch of artificial intelligence that focuses on enabling computers to understand, interpret, and interact with human language in a way that is both meaningful and useful. It involves tasks such as language understanding, language generation, and language translation, allowing machines to process and analyze text or speech data to extract meaning and respond accordingly.

- As we developed a retrieval-based chatbot, we should define responses as “**Intents**” in a JSON file to effectively answer questions, search from a knowledge base, and engage in conversations with users,

a chatbot needs the ability to understand and interpret user input, determining their intentions and meaning. This understanding is crucial for the chatbot to provide accurate and relevant responses.

Chatbot Conceptual Framework



Chatbot Conceptual Framework

- This framework provides a structured approach to designing, developing, and deploying chatbot solutions. It outlines the key components and considerations involved in creating an effective and functional chatbot. The database represents the intents of the chatbot.
-



Code Implementation

Required Packages

To develop the chatbot, we will need to install the following Python packages.

```
nlk==3.7
```

```
numpy==1.23.3
```

```
tensorflow==2.10.0
```

```
joblib==1.2.0
```

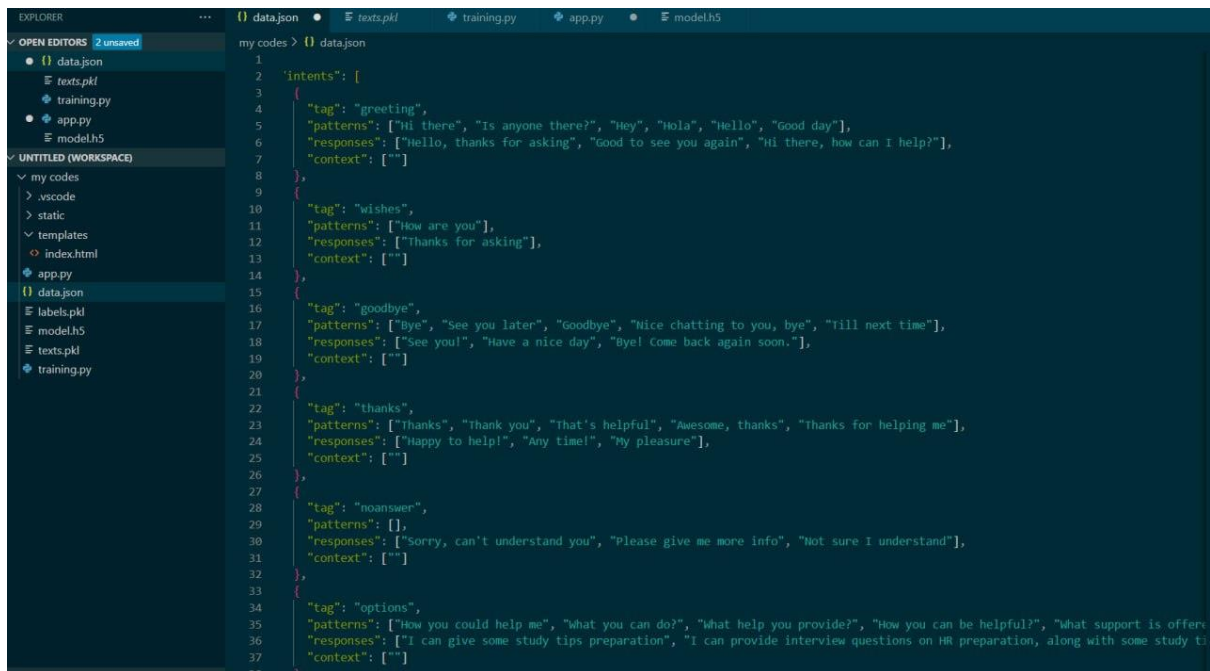
```
keras==2.10.0
```

```
Keras-Preprocessing==1.1.2
```

NLTK package will provide various tools and resources for NLP tasks such as tokenization, stemming, and part-of-speech tagging. **TensorFlow** is a popular deep learning framework used for building and training neural networks, including models for NLP tasks. And, **Keras** is a high-level neural network library that runs on top of TensorFlow. It simplifies the process of building and training deep learning models, including NLP models.

Defining Intents

To define intents, messages, and corresponding responses, we have create a JSON file named "data.json" with the following structure:



```
1
2 'intents': [
3   {
4     "tag": "greeting",
5     "patterns": ["Hi there", "Is anyone there?", "Hey", "Hola", "Hello", "Good day"],
6     "responses": ["Hello, thanks for asking", "Good to see you again", "Hi there, how can I help?"],
7     "context": [""]
8   },
9   {
10    "tag": "wishes",
11    "patterns": ["How are you"],
12    "responses": ["Thanks for asking"],
13    "context": [""]
14  },
15  {
16    "tag": "goodbye",
17    "patterns": ["Bye", "See you later", "Goodbye", "Nice chatting to you, bye", "Till next time"],
18    "responses": ["See you!", "Have a nice day", "Bye! Come back again soon."],
19    "context": [""]
20  },
21  {
22    "tag": "thanks",
23    "patterns": ["Thanks", "Thank you", "That's helpful", "Awesome, thanks", "Thanks for helping me"],
24    "responses": ["Happy to help!", "Any time!", "My pleasure"],
25    "context": [""]
26  },
27  {
28    "tag": "noanswer",
29    "patterns": [],
30    "responses": ["Sorry, can't understand you", "Please give me more info", "Not sure I understand"],
31    "context": [""]
32  },
33  {
34    "tag": "options",
35    "patterns": ["How you could help me", "What you can do?", "What help you provide?", "How you can be helpful?", "What support is offered"],
36    "responses": ["I can give some study tips preparation", "I can provide interview questions on HR preparation, along with some study tips"],
37    "context": [""]
38  }
39 ]
```

```

{
  "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?", "What can you do for me?"],
  "responses": ["I can give some study tips preparation", "I can provide interview questions on HR preparation, along with some study tips", "I can help you with your technical skills"],
  "context": [""]
},
{
  "tag": "study_tips",
  "patterns": ["Can you give me some study tips", "How can I study effectively", "Do you have any advice for studying"],
  "responses": [
    "Sure! Here are some study tips:",
    "1. Create a study schedule and stick to it.",
    "2. Find a quiet and comfortable study space."
  ],
  "context": [""]
},
{
  "tag": "technical_skills",
  "patterns": ["How can I improve my technical skills?", "What are some resources for learning coding?", "Do you have any advice for enhancing my technical skills?"],
  "responses": [
    "Absolutely! Here are some ways to enhance your technical skills:",
    "1. Take online courses on platforms like Coursera or Udemy.",
    "2. Join coding communities or forums for peer support."
  ],
  "context": [""]
},
{
  "tag": "HR_round",
  "patterns": ["What are the questions asked on HR round", "Sample questions on HR round", "HR round preparation"],
  "responses": [
    "1. Tell me about yourself.",
    "2. What are your strengths and weaknesses?",
    "3. Practice active learning techniques like summarizing and teaching the material to someone else."
  ],
  "context": [""]
}
]

```

Creating Chatbot Model

Step 01 - Before proceeding, we have created a Python file as "training.py" then make sure to import all the required packages to the Python file.

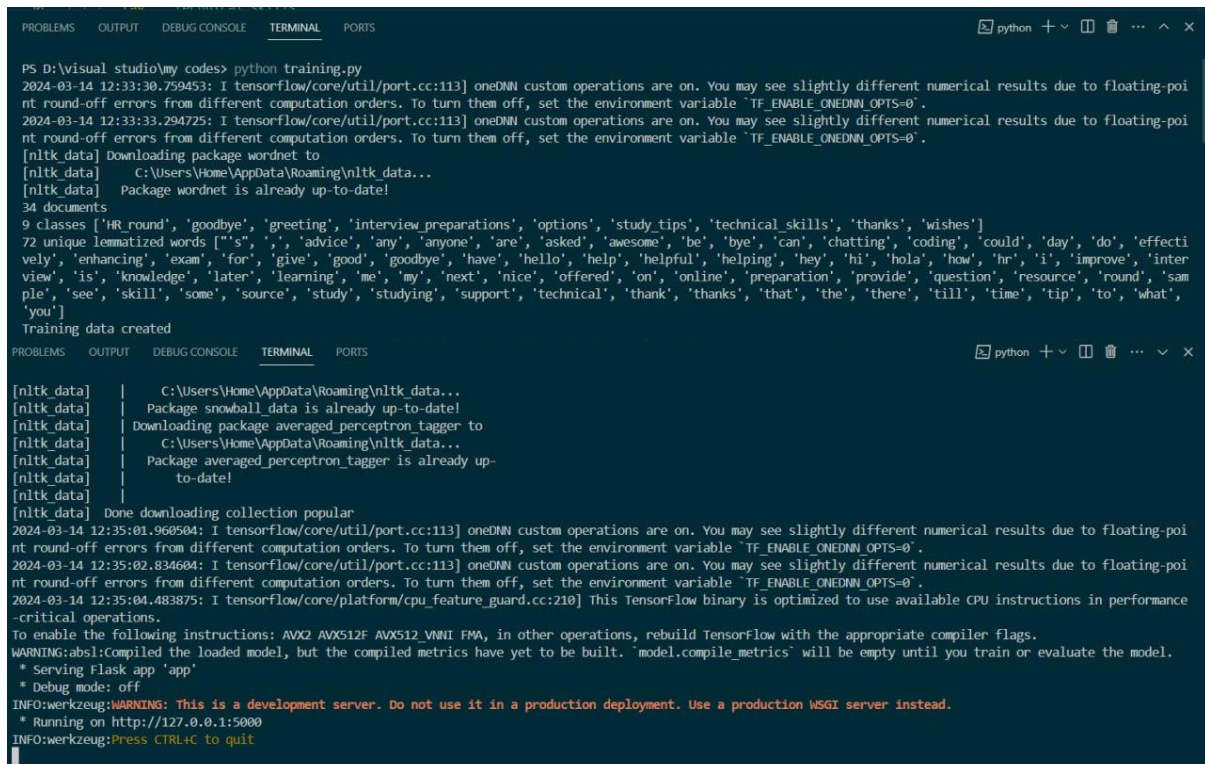
```

data.json  ●  texts.pkl  training.py  app.py  ●  model.h5
my codes > training.py > ...
1  import nltk
2  from nltk.stem import WordNetLemmatizer
3  import json
4  import pickle
5  import numpy as np
6  from keras.models import Sequential
7  from keras.layers import Dense, Activation, Dropout
8  from keras.optimizers import SGD
9  import random
10
11 # Download WordNet if not already downloaded
12 nltk.download('wordnet')
13
14 lemmatizer = WordNetLemmatizer()
15
16 words = []
17 classes = []
18 documents = []
19 ignore_words = ['?', '!']
20 data_file = open('data.json').read()
21 intents = json.loads(data_file)

```

In this step, we import the necessary packages required for building the chatbot. The packages include nltk, WordNetLemmatizer from nltk.stem, json, pickle, numpy, Sequential and various layers from Dense, Activation, Dropout from keras.models, and SGD from keras.optimizers. These

packages are essential for performing NLP tasks and building the neural network model.



```
PS D:\visual studio\my codes> python training.py
2024-03-14 12:33:30.759453: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2024-03-14 12:33:33.294725: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
[nltk_data] Downloading package wordnet to
[nltk_data]   C:\Users\Home\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
34 documents
9 classes ['HR_round', 'goodbye', 'greeting', 'interview_preparations', 'options', 'study_tips', 'technical_skills', 'thanks', 'wishes']
72 unique lemmatized words ['s', ',', 'advice', 'any', 'anyone', 'are', 'asked', 'awesome', 'be', 'bye', 'can', 'chatting', 'coding', 'could', 'day', 'do', 'effectively', 'enhancing', 'exam', 'for', 'give', 'good', 'goodbye', 'have', 'hello', 'help', 'helpful', 'helping', 'hey', 'hi', 'hola', 'how', 'hr', 'i', 'improve', 'inter view', 'is', 'knowledge', 'later', 'learning', 'me', 'my', 'next', 'nice', 'offered', 'on', 'online', 'preparation', 'provide', 'question', 'resource', 'round', 'sample', 'see', 'skill', 'some', 'source', 'study', 'studying', 'support', 'technical', 'thank', 'thanks', 'that', 'the', 'there', 'till', 'time', 'tip', 'to', 'what', 'you']
Training data created

[nltk_data] | C:\Users\Home\AppData\Roaming\nltk_data...
[nltk_data] | Package snowball_data is already up-to-date!
[nltk_data] | Downloading package averaged_perceptron_tagger to
[nltk_data] | C:\Users\Home\AppData\Roaming\nltk_data...
[nltk_data] | Package averaged_perceptron_tagger is already up-
[nltk_data] | to-date!
[nltk_data] | Done downloading collection popular
2024-03-14 12:35:01.960504: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2024-03-14 12:35:02.834604: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2024-03-14 12:35:04.483875: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. 'model.compile_metrics' will be empty until you train or evaluate the model.
* Serving Flask app 'app'
* Debug mode: off
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
```

Step 02 - Loading and Preprocessing the Data

In this step, we load the data from the data.json file, which contains intents, patterns, and responses for the chatbot. We iterate through each intent and its patterns, tokenize the words, and perform lemmatization and lowercasing. We collect all the unique words and intents, and finally, we create the documents by combining patterns and intents.

Saving the Preprocessed Data:



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

Step 03 - Creating the Training Data

In this step, we create the training data by converting the documents into a bag-of-words representation. We iterate through each document, create a bag-of-words array with 1 if a word is present in the pattern, and append the corresponding output row with a '1' for the current intent and '0' for other intents. We shuffle the training data and convert it into a numpy array.

Step 04 - Building the Neural Network Model

```
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'))
```

Here, we define the architecture of the neural network model using the Sequential class from keras.models. The model consists of three layers: the input layer with 128 neurons, a dropout layer to prevent overfitting, a hidden layer with 64 neurons, another dropout layer, and an output layer with a number of neurons equal to the number of intents. The activation functions used are 'relu' for hidden layers and 'softmax' for the output layer.

Step 05 - Compiling and Training the Model

Step 06 - Saving the Trained Model

Load the Trained Model and Create a Flask App

To do this we need to create a Python file as "app.py" (as in my project structure), in this file we are going to load the trained model and create a flask app.

```
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()
```

Here, we use the `load_model` function from Keras to load the pre-trained model from the 'model.h5' file. This file contains the saved weights and architecture of the trained model

In this part of the code, we initialize the WordNetLemmatizer object from the NLTK library. The purpose of using the lemmatizer is to transform words into their base or root forms. This process allows us to simplify words and bring them to a more standardized or meaningful representation. By reducing words to their canonical forms, we can improve the accuracy and efficiency of text-processing tasks performed by the chatbot.

```
def predict_class(sentence, model):  
  
    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]  
  
    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
```

The `predict_class` function enables the model to predict the most likely intent for a given input sentence and provides the associated probability. This allows the chatbot to understand the user's intention and generate appropriate responses based on the predicted intent.

```
data.json • texts.pkl • training.py • app.py • model.h5  
my codes > app.py > predict_class  
51  
52 def getResponse(ints, intents_json):  
53     tag = ints[0]['intent']  
54     list_of_intents = intents_json['intents']  
55     for intent in list_of_intents:  
56         if intent['tag'] == tag:  
57             responses = intent['responses']  
58             return random.choice(responses)  
59     return "Sorry, I couldn't understand that."  
60  
61  
62  
63 def chatbot_response(msg):  
64     ints = predict_class(msg, model)  
65     print("Predicted Intents:", ints) # Add this line to print predicted intents  
66     res = getResponse(ints, intents)  
67     return res  
68
```


Generating Bot Responses

These functions work together to determine the appropriate response from the chatbot based on the user's input. The **getResponse** function matches the predicted intent with the corresponding intents data and randomly selects a response. The **chatbot_response** function orchestrates the intent prediction and response selection process to provide a response to the user's message.

```
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()
```

This code sets up a Flask web application with routes for the home page and receiving user input. It integrates the chatbot functionality by calling the **chatbot_response** function to generate responses based on user messages.

Designing the Chatbot Web Interface

As in the above code snippet we used "index.html" to render the web interface template. So, we need to create a HTML file and its CSS file. To create these files we need to create subdirectories as "templates" (for HTML file) and "static" (for CSS file).

The HTML code creates a chatbot interface with a header, message display area, input field, and send button. It utilizes JavaScript to handle user interactions and communicate with the server to generate bot responses dynamically. The appearance and behavior of the interface can be further customized using CSS.

Chatbot Testing

To test the created chatbot we should run the "app.py" file. Then the flask app will run and will create a server link. After that click the server link to access the chatbot.

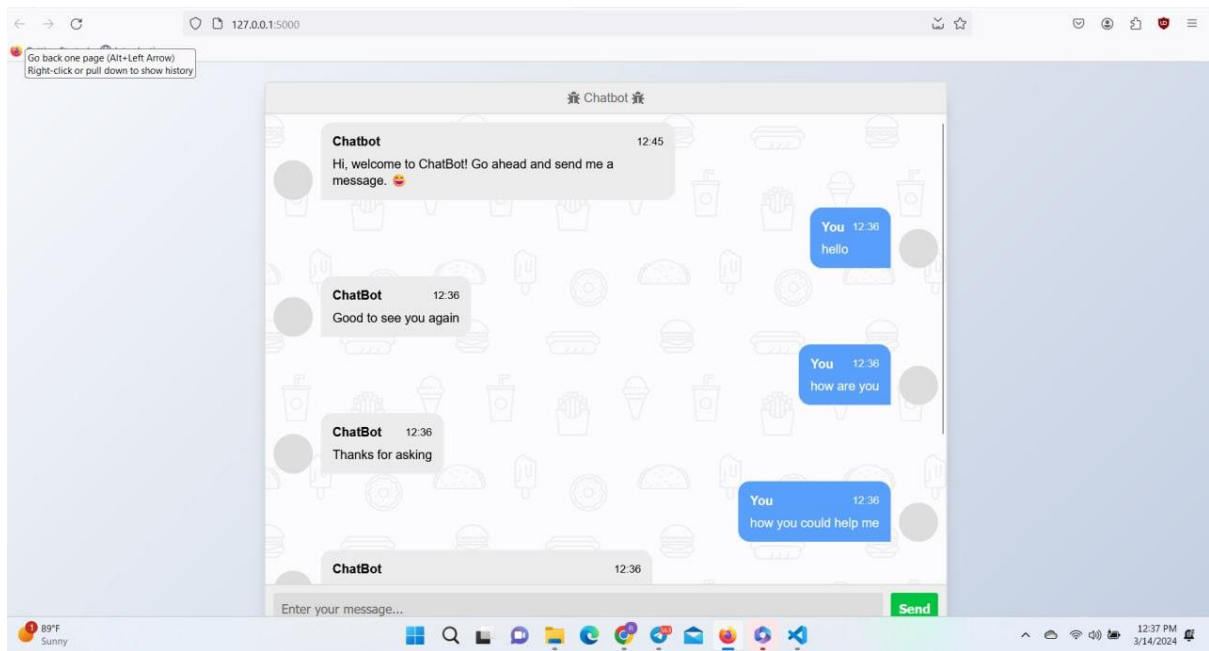
OUTPUT:

When we run the code of training.py the Model is created as shown in the figure:

```
Epoch 186/200
7/7 ----- 0s 3ms/step - accuracy: 1.0000 - loss: 0.0264
Epoch 187/200
7/7 ----- 0s 989us/step - accuracy: 0.9657 - loss: 0.0930
Epoch 188/200
7/7 ----- 0s 2ms/step - accuracy: 0.9926 - loss: 0.0345
Epoch 189/200
7/7 ----- 0s 2ms/step - accuracy: 1.0000 - loss: 0.0057
Epoch 190/200
7/7 ----- 0s 1ms/step - accuracy: 1.0000 - loss: 0.0347
Epoch 191/200
7/7 ----- 0s 2ms/step - accuracy: 1.0000 - loss: 0.0255
Epoch 192/200
7/7 ----- 0s 2ms/step - accuracy: 0.9835 - loss: 0.0542
Epoch 193/200
7/7 ----- 0s 1ms/step - accuracy: 1.0000 - loss: 0.0054
Epoch 194/200
7/7 ----- 0s 3ms/step - accuracy: 0.9885 - loss: 0.0189
Epoch 195/200
7/7 ----- 0s 2ms/step - accuracy: 1.0000 - loss: 0.0326
Epoch 196/200
7/7 ----- 0s 3ms/step - accuracy: 1.0000 - loss: 0.0192
Epoch 197/200
7/7 ----- 0s 3ms/step - accuracy: 1.0000 - loss: 0.0189
Epoch 198/200
7/7 ----- 0s 3ms/step - accuracy: 1.0000 - loss: 0.0036
Epoch 199/200
7/7 ----- 0s 1ms/step - accuracy: 1.0000 - loss: 0.0660
Epoch 200/200
7/7 ----- 0s 2ms/step - accuracy: 1.0000 - loss: 0.0033
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend us
ing instead the native Keras format, e.g. `model.save("my_model.keras")` or `keras.saving.save_model(model, "my_model.keras")`.
Model created
```

After running the code of training.py we get the output as:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
[nltk_data] | C:\Users\Home\AppData\Roaming\nltk_data...
[nltk_data] | Package snowball_data is already up-to-date!
[nltk_data] | Downloading package averaged_perceptron_tagger to
[nltk_data] | C:\Users\Home\AppData\Roaming\nltk_data...
[nltk_data] | Package averaged_perceptron_tagger is already up-
[nltk_data] | to-date!
[nltk_data] | Done downloading collection popular
2024-03-14 12:35:01.960504: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-poi
nt round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2024-03-14 12:35:02.834604: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-poi
nt round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2024-03-14 12:35:04.483875: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance
-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
* Serving Flask app 'app'
* Debug mode: off
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
```



DONE!!!!

The chatbot is successfully running.