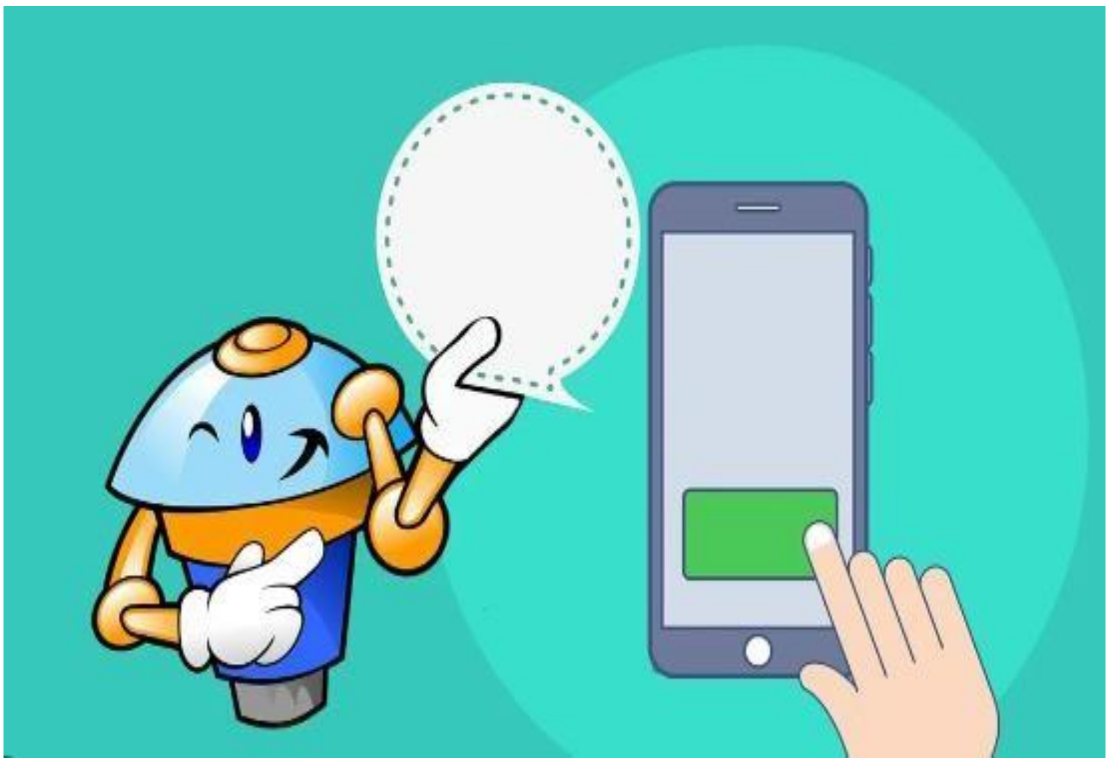
CREATE A CHATBOT IN PYTHON

PHASE-5 SUBMISSION

REGISTER NUMBER : 963521104061
NAME :Thanya S

Project title : Chatbot in python



CREATE A CHATBOT IN PYTHON

Introduction :

Python chatbot development is a dynamic journey that involves the careful integration of key components: feature selection, model training, and model evaluation. These three pillars form the foundation of creating intelligent and responsive chatbots..

Feature selection is the initial step that allows chatbots to comprehend and respond to human language effectively. It involves the identification and extraction of key elements from user input, such as intent, entities, and keywords. Python's powerful natural language processing (NLP) libraries, like NLTK and spaCy, make this process more accessible, enabling the chatbot to grasp user queries.

Model training is the heart of chatbot development, where machine learning and NLP models are employed to process user input and generate contextually relevant responses. Python's rich ecosystem, including scikit-learn, TensorFlow, and PyTorch, offers a wide array of tools for developing these models. The choice of model depends on the chatbot's complexity, from rule-based systems for straightforward tasks to advanced deep learning models, such as recurrent neural networks (RNNs) and transformer architectures.

Model evaluation is the critical phase that gauges the chatbot's performance. Python provides a comprehensive suite of evaluation tools, including metrics like accuracy, precision, recall, and F1 score. These metrics help ensure that the chatbot delivers accurate and contextually relevant responses to users.

Chatbot:

Chatbots are conversational tools that perform routine tasks efficiently. It Communicates with users using interactive text or speech capabilities. People like them because they help them get through those tasks quickly so they can focus their attention on high-level, strategic, and engaging activities

Problem Definition:

The problem is creating a Chatbot in python that uses ask questions throughout the user's journey and provide information that may persuade the user and create a lead. The Chatbot aims to Communicates with users using interactive text or speech capabilities.

Design Thinking:

1. **Functionality**: Define the scope of the chatbot's abilities, including answering common questions, providing guidance, and directing users to appropriate resources.
2. **User Interface**: Determine where the chatbot will be integrated (website, app) and design a user-friendly interface for interactions.
3. **Natural Language Processing (NLP):** Implement NLP techniques to understand and process user input in a conversational manner.
4. **Responses:** Plan responses that the chatbot will offer, such as accurate answers, suggestions, and assistance.
5. **Integration:** Decide how the chatbot will be integrated with the website or app.
6. **Testing and Improvement:** Continuously test and refine the chatbot's performance based on user interactions.

Innovative technices

There are two approaches that can be used to develop a chatbot depending on the algorithms and techniques adopted there are rule-based approach, machine learning approach and dialogue management

Innovation:

1. **Rule based approach:**

These chatbots follow predefined rules and decision trees to respond to user inputs. They are limited to the specific rules and patterns programmed into them and lack the ability to understand natural language process. Rule-based chatbots are relativelysimple to build and are suitable tasks with well-defined interactions.

2. Machine Learning Approach:

These chatbots use machine learning techniques, such as natural language processing (NLP) and neural networks, to understand and generate responses based on patterns in data. Machine learning-based chatbots are more flexible and capable of handling a wider range user inputs but require more extensive development and training.

The choice between these methods depends on the complexity of the chatbot's intended tasks and the desired level of sophistication in its interactions. Many modern chatbots combine both approaches, using rules for specific tasks and machine learning for more general language understanding.

3. Dialog Management:

Dialog management is the process of controlling the flow of conversation in a chatbot. Various methods, such as rule-based systems, state machines, or reinforcement learning, are used to manage the chatbot's interactions with users. It ensures that the chatbot responds appropriately and maintains context during a conversation.

Ensemble Methods:

Creating a chatbot using python is a task that typically relies on sesimological data and machine learning techniques.

1. Stacking Models:

Stack multiple chatbot models in a hierarchical manner, with one model refining the output of another. This can help improve the overall quality of responses

2. Voting Mechanism:

Ensemble voting methods, such as majority voting or weighted voting, can be used to combine predictions from multiple models or components within the chatbot. This is helpful for improving classification or decision-making tasks.

3. Bagging :

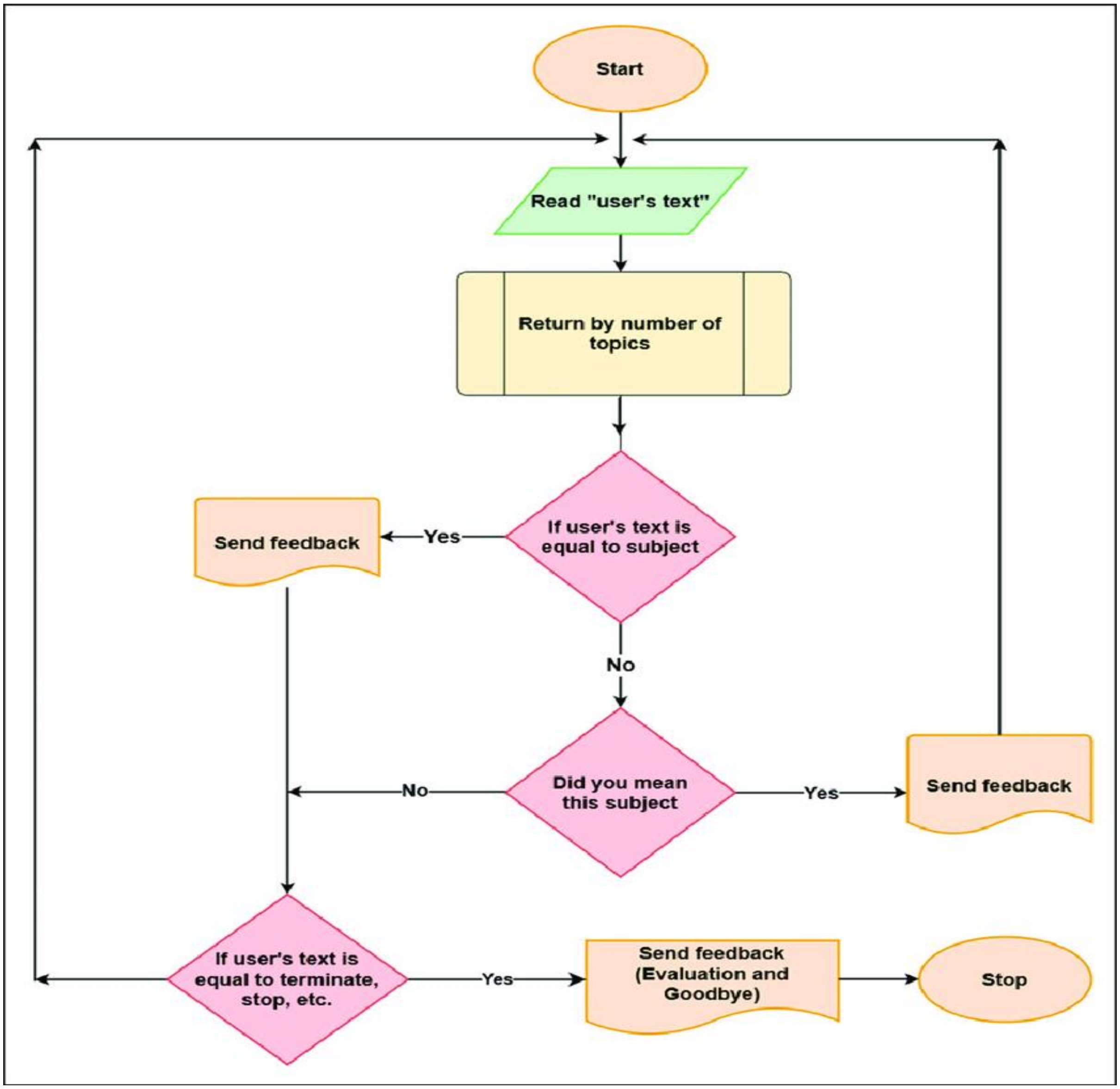
In the Bagging methods can be applied to reduce overfitting and enhance the generalization of chatbot models. Multiple instances of the same model are trained on different subsets of data, and their predictions are aggregated.

4. Boosting :

Boosting algorithms, like AdaBoost, can be used to improve the performance of individual classifiers or models within the chatbot. It assigns weights to data points to emphasize misclassified examples, leading to better accuracy.

5. Random Forest:
   Random Forest is an ensemble method based on decision trees. It can be utilized to improve intent classification or other classification tasks in chatbots by aggregating predictions from multiple decision trees.
6. Model Diversity:
   Ensuring that the chatbot employs diverse models and techniques can enhance its adaptability and robustness, as it can handle a wider range of user inputs and scenarios..

Flowchart:



<u>Libraries</u>:

Here are the key libraries commonly used in chatbot development:

<u>Natural Language Processing (NLP) Libraries:</u>

NLTK (Natural Language Toolkit): It provides tools for tokenization, stemming, lemmatization, part-of-speech tagging, and more.

<u>Example</u>:

```
import nltk
sentence = "This is an example sentence."
tokens = nltk.word_tokenize(sentence)
print(tokens)
```

spaCy: spaCy is a popular NLP library known for its speed and accuracy.

<u>Example</u>:

```
import spacy
nlp = spacy.load("en_core_web_sm")
text = "SpaCy is a fast and efficient natural language processing library."
doc = nlp(text)
for token in doc:
print(token.text, token.pos_)
```

TextBlob: It provides easy-to-use APIs for common NLP tasks like sentiment analysis and part-of-speech tagging.

Example:

```
from textblob import TextBlob
text = "TextBlob is a simple NLP library for Python."
blob = TextBlob(text)
# Sentiment analysis
sentiment = blob.sentiment
print(sentiment)
# Part-of-speech tagging
tags = blob.tags
print(tags)
```

Machine Learning Libraries:

Scikit-learn: It can be used for tasks such as intent classification, sentiment analysis, and entity recognition in chatbots.

Example:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
text_clf = Pipeline([('tfidf', TfidfVectorizer()), ('clf', SVC())])
text_clf.fit(X_train, y_train)
```

TensorFlow and Keras: These libraries are used for building and training deep learning models, which can be applied to natural language understanding tasks.

Example:

```
import tensorflow as tf
from tensorflow import keras
model = keras.Sequential()
model.add(keras.layers.Dense(128, input_shape=(X_train.shape[1],), activation='relu'))
model.add(keras.layers.Dense(num_classes, activation='softmax'))
```

Web Frameworks:

Flask or Django: They provide a way to build the chatbot's user interface and handle HTTP requests and responses.

Example:

```
from flask import Flask, request, jsonify
app = Flask(__name)
@app.route('/chat', methods=['POST'])
def chat():
    user_message = request.json['message']
    # Process the message and generate a response
    return jsonify({"response": chatbot_response})
if __name__ == '__main__':
    app.run()
```

NLP Techniques:

Utilize various NLP techniques to enhance your chatbot's capabilities. Some common techniques include:

- Text Classification: To understand the intent of user messages.
- Named Entity Recognition (NER): To extract entities like dates, locations, or product names.
- Sentiment Analysis: To determine the sentiment of user messages.
- Language Modeling: For generating natural-sounding responses.

Data set:

Here's a brief description of a popular dataset along with its source:

Dataset Name: Iris Dataset

Source:

The Iris dataset is a well-known dataset in machine learning and can be obtained from the UCI Machine Learning Repository: UCI Iris Dataset.

Description:

The Iris dataset is small dataset that contains measurements of four features (sepal length, sepal width, petal length, and petal width) of three different species of iris flowers (setosa, versicolor, and virginica). It consists of 150 instances, with 50 samples from each of the three species. The dataset is commonly used for tasks like classification and clustering in machine learning to demonstrate various algorithms and techniques.

Code:

```
import numpy as np
import pandas as pd
import seaborn as sns
sns.set_palette('husl')
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
data = pd.read_csv('../input/Iris.csv')


IN[2]
data.head()
```

Output:

Out[2]:

	Id	SepalLengthCm	SepalWidthCm	PetalLer
0	1	5.1	3.5	1.4
1	2	4.9	3.0	1.4
2	3	4.7	3.2	1.3
3	4	4.6	3.1	1.5
4	5	5.0	3.6	1.4

Building the chatbot by integrating it into a web app using Flask.

Step 1:

Create a Flask Project

1. Create a new directory for your Flask project.
2. Inside this directory, create a virtual environment:

Python code:

```
python -m venv venv
source venv/bin/activate  # On Windows, use 'venv\Scripts\activate'
```

3. Install Flask:

Python code:

```
pip install Flask
```

Step 2:

Create a Basic Flask Web App

1. Create a file named app.py in your project directory.

2. In app.py, import Flask and create a basic Flask app:

Python code:

```
from flask import Flask, render_template, request
app = Flask(__name)
@app.route('/')
def home():
    return render_template('index.html')
if __name__ == '__main__':
    app.run(debug=True)
```

Step 3:

Create HTML Templates

1. Create a folder named templates in your project directory.

2. Inside the templates folder, create an HTML file named index.html:

HTML code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Chatbot</title>
</head>
<body>
    <h1>Chatbot</h1>
    <div id="chat-box"></div>
```

```html
    <input type="text" id="user-input" placeholder="Type your message">
    <button id="send-button">Send</button>
    <script src="static/chatbot.js"></script>
</body>
</html>
```

Step 4:

Create JavaScript for Chatbot

1. Create a folder named static in your project directory.

2. Inside the static folder, create a JavaScript file named chatbot.js. This file will handle user input and responses from the chatbot.

JavaScript code

```javascript
// chatbot.js
const chatBox = document.getElementById('chat-box');
const userInput = document.getElementById('user-input');
const sendButton = document. getElementById('send-button');
sendButton.addEventListener('click', () => {
    const userMessage = userInput.value;
    chatBox.innerHTML += `<p>User: ${userMessage}</p>`;

    // Send user message to the server for processing
    fetch('/chat', {
        method: 'POST',
        body: JSON.stringify({ userMessage }),
        headers: {
            'Content-Type': 'application/json',
```

```
        },
    })
    .then(response => response.json())
    .then(data => {
        const chatbotMessage = data.chatbotMessage;
        chatBox.innerHTML += `<p>Chatbot: ${chatbotMessage}</p>`;
    });

    userInput.value = '';
});
```

Step 5:

Implement Chatbot Logic in Flask

1. In app.py, add a new route to handle user messages and return chatbot responses.

Python code:

```
import json
# Import your chatbot implementation here
@app.route('/chat', methods=['POST'])
def chat():
    data = json.loads(request.data)
    user_message = data['userMessage']
    # Implement your chatbot logic here
    chatbot_response = get_chatbot_response(user_message)
    return json.dumps({'chatbotMessage': chatbot_response})
```

2. Implement the get_chatbot_response function using your existing chatbot logic. This function should take a user message as input and return the chatbot's response.

Step 6:

Run Your Flask App

1.Run your Flask app using the following command:

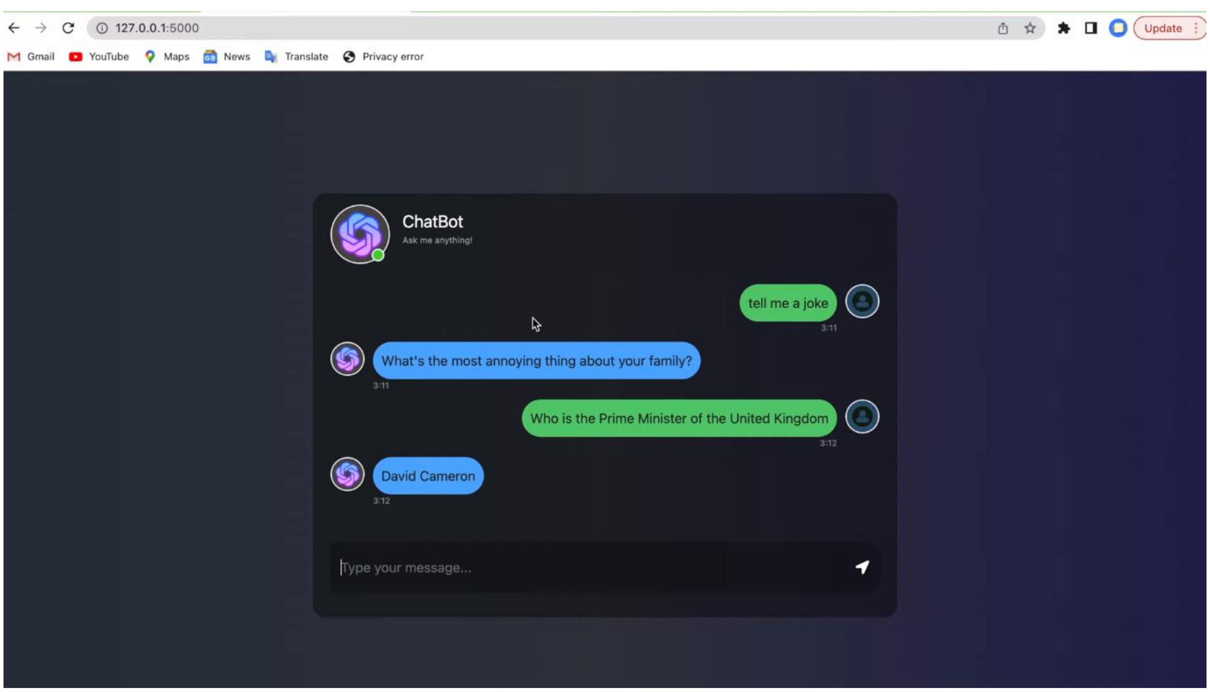
Python Code:

```
python app.py
```

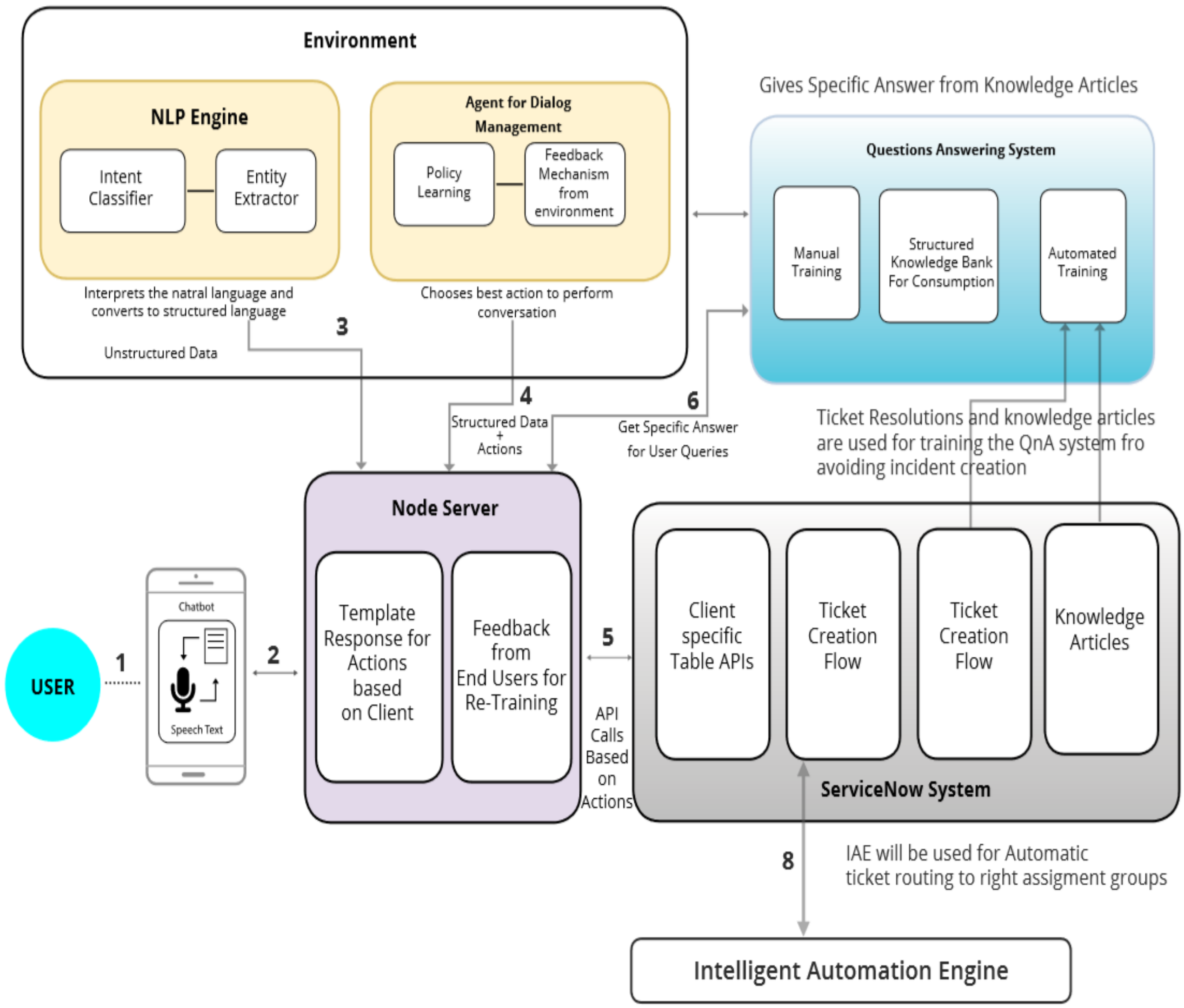
2. Access your chatbot web app by opening a web browser and navigating to http:// 127.0.0.1.5000.

Our chatbot is now integrated into a Flask web app. Users can interact with it through the web interface. Make sure you adapt the chatbot logic and responses to suit your specific use case and chatbot implementation. You can also enhance the web interface to make it more interactive and visually appealing

Output:



Architectural Diagram in Chatbot:



Given data set:

```
hi, how are you doing?	i'm fine. how about yourself?
i'm fine. how about yourself?	i'm pretty good. thanks for asking.
i'm pretty good. thanks for asking.	no problem. so how have you been?
no problem. so how have you been?	i've been great. what about you?
i've been great. what about you?	i've been good. i'm in school right now.
i've been good. i'm in school right now.	what school do you go to?
what school do you go to?	i go to pcc.
i go to pcc.	do you like it there?
do you like it there?	it's okay. it's a really big campus.
it's okay. it's a really big campus.	good luck with school.
good luck with school.	thank you very much.
how's it going?	i'm doing well. how about you?
i'm doing well. how about you?	never better, thanks.
never better, thanks.	so how have you been lately?
so how have you been lately?	i've actually been pretty good. you?
i've actually been pretty good. you?	i'm actually in school right now.
i'm actually in school right now.	which school do you attend?
which school do you attend?	i'm attending pcc right now.
i'm attending pcc right now.	are you enjoying it there?
are you enjoying it there?	it's not bad. there are a lot of people there.
it's not bad. there are a lot of people there.	good luck with that.
good luck with that.	thanks.
how are you doing today?	i'm doing great. what about you?
i'm doing great. what about you?	i'm absolutely lovely, thank you.
i'm absolutely lovely, thank you.	everything's been good with you?
everything's been good with you?	i haven't been better. how about yourself?
i haven't been better. how about yourself?	i started school recently.
i started school recently.	where are you going to school?
where are you going to school?	i'm going to pcc.
i'm going to pcc.	how do you like it so far?
how do you like it so far?	i like it so far. my classes are pretty good right now.
i like it so far. my classes are pretty good right now.	i wish you luck.
```

It consists of two columns: question \t answer \n . Suitable for simple chatbots. Contains 3725 items

Overview of the process:

Building a Python chatbot involves several key steps, including data preparation, feature selection, model training, model evaluation, and deployment. Here's an overview of these steps:

Prepare the Data:

Data Collection: Gather a diverse and representative dataset of text-based conversations, user queries, and responses.

Data Cleaning: Clean the data by removing irrelevant or noisy information, such as special characters, HTML tags, or duplicate entries.
Data Preprocessing: Tokenize the text, convert it to lowercase, and perform other necessary text preprocessing tasks.
Labeling: Annotate or label the data with appropriate intents and entities if your chatbot requires understanding specific user intentions.

Perform Feature Selection:

Feature Engineering: Create relevant features or representations from the text data, such as TF-IDF vectors, word embeddings (e.g., Word2Vec, GloVe), or more advanced representations like BERT embeddings.
Dimensionality Reduction: If the feature space is too large, you can perform dimensionality reduction techniques like Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbor Embedding (t-SNE) to reduce computational complexity and noise.

Train the Model:

Choose a Model Architecture: Select a suitable model architecture for your chatbot, such as a rule-based system, a sequence-to-sequence model, a transformer-based model, or a combination of these.
Train the Model: Train the chosen model using the preprocessed data. You may use machine learning libraries like scikit-learn, deep learning frameworks like TensorFlow or PyTorch, or specialized NLP libraries like spaCy or Rasa NLU.

Evaluate the Model:

Split the Data: Divide your dataset into training, validation, and test sets to assess the model's performance.
Metrics: Evaluate the chatbot's performance using relevant metrics like accuracy, precision, recall, F1-score, or domain-specific metrics like BLEU or ROUGE for text generation tasks.
Fine-Tuning: Refine the model by adjusting hyperparameters or architecture, if necessary, based on the evaluation results.

Deploy the Model:

Integration: Integrate the chatbot model into your desired deployment environment, such as a web application, mobile app, or chat platform (e.g., Slack or Facebook Messenger).
Server Deployment: Deploy the model on a server or cloud service (e.g., AWS, Azure, or Google Cloud) using technologies like Docker containers or serverless computing.
API Creation: Create an API to communicate with the deployed model and handle user requests.
Continuous Monitoring: Continuously monitor the chatbot's performance and gather user feedback for further improvements.
It's important to note that chatbot development is an ongoing process, and you may need to iterate on these steps to enhance the chatbot's performance and capabilities over time.

Additionally, consider the ethical and privacy implications of deploying a chatbot, and ensure that it complies with relevant regulations and guidelines.

Necessary steps to follow:

1.Import Libraries:

Start by importing necessary libraries:

Program:

In [1]:

```
import tensorflow as tf
import numpy as np
 import pandas as pd ha
import matplotlib.pyplot as plt
import seaborn as sns from tensorflow.keras.layers
 import TextVectorization
import re,string from tensorflow.keras.layers
 import LSTM,Dense,Embedding,Dropout,LayerNormalization
```

2.Load the Dataset:

The below Python code is used to load data from a tab-separated values (TSV) file named "dialogs.txt" and store it in a pandas DataFrame.

Program:

In [2]:

```
df=pd.read_csv('/kaggle/input/simple-dialogs-
forchatbot/dialogs.txt',sep='\t',names=['question','answer']) print(f'Dataframe size:
{len(df)}') df.head()
Dataframe size: 3725
```

Out[2]:

	Question	answer
0	hi, how are you doing?	i'm fine. how about yourself?
1	i'm fine. how about yourself?	i'm pretty good. thanks for asking.
	Question	answer
2	i'm pretty good. thanks for asking.	no problem. so how have you been?
3	no problem. so how have you been?	i've been great. what about you?
4	i've been great. what about you?	i've been good. i'm in school right now.

3. Data Preprocessing

Loading and preprocessing of data are crucial steps in the development of a Python chatbot for several important reasons:

Data Quality and Consistency: Loading and preprocessing ensure that the input data, which could be in various formats, is standardized, cleaned, and structured appropriately. This leads to consistent and reliable input for the chatbot, reducing the risk of errors or misunderstandings during conversations.

Understanding User Input: Chatbots rely on Natural Language Processing (NLP) to understand and respond to user input. Preprocessing includes tokenization, which breaks down text into meaningful units like words or phrases. This helps the chatbot understand the user's message and extract relevant information.

Noise Reduction: In real-world scenarios, text data often contains noise in the form of typos, slang, abbreviations, or special characters. Preprocessing can involve tasks like spell-checking and removing special characters to ensure the chatbot can effectively interpret the user's intent.

4.Data Visualization

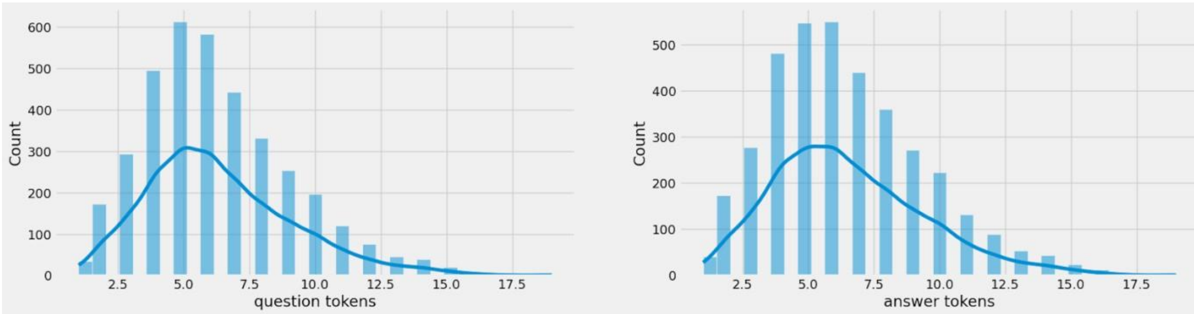
Data virtualization in a Python chatbot enables the bot to seamlessly gather and manipulate data from diverse sources, providing users with a unified and interactive data experience.

The following code is focused on data preprocessing and data visualization for a DataFrame called "df." It appears to be analyzing the distribution of token lengths in the 'question' and 'answer' columns of the DataFrame.

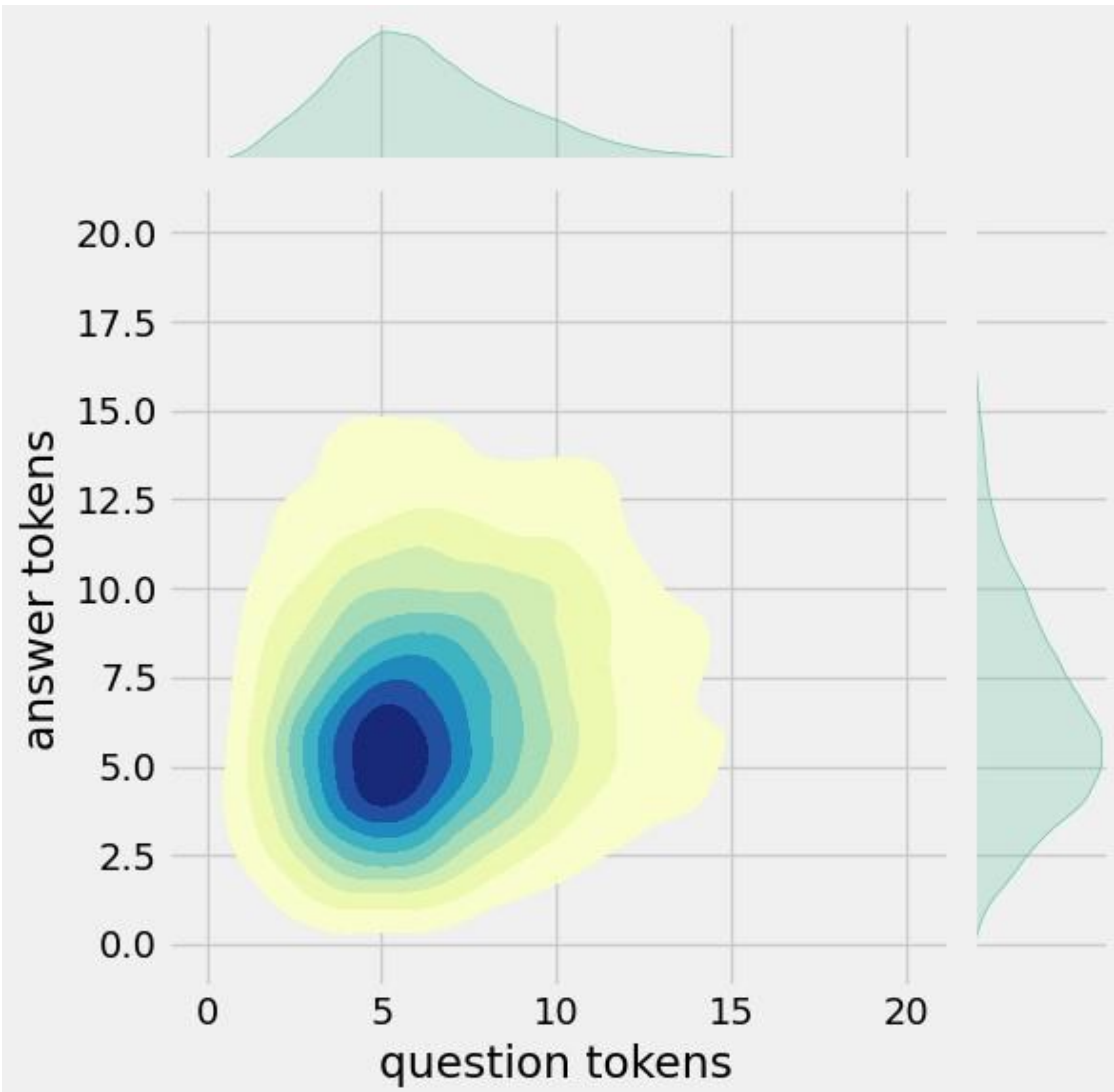
Program:

In [3]:

```
df['question tokens']=df['question'].apply(lambda x:len(x.split())) df['answer tokens']=df['answer'].apply(lambda x:len(x.split()))

plt.style.use('fivethirtyeight') fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))

sns.set_palette('Set2') sns.histplot(x=df['question tokens'],data=df,kde=True,ax=ax[0])

sns.histplot(x=df['answer tokens'],data=df,kde=True,ax=ax[1])

sns.jointplot(x='question tokens',y='answer tokens',data=df,kind='kde',fill=True,cmap='YlGnBu')

plt.show()
```



Count
0
100
200
300
400
500
600
2.5
5.0
7.5
10.0
12.5
15.0
17.5
question tokens
Count
0
100
200
300
400
500
2.5
5.0
7.5
10.0
12.5
15.0
17.5
answer tokens



answer tokens
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
5
10
15
20
question tokens

5.Text Cleaning:

The below code segment is focused on text cleaning and preparing the data for a chatbot training or conversation model. It performs several text cleaning and transformation operations on the given dataset, resulting in encoder and decoder inputs, along with decoder targets.

In [4]:

```
def clean_text(text):
    text=re.sub('-',' ',text.lower())     text=re.sub('[.]',' . ',text)     text=re.sub('[1]',' 1
',text)     text=re.sub('[2]',' 2 ',text)     text=re.sub('[3]',' 3 ',text)     text=re.sub('[4]','
4 ',text)     text=re.sub('[5]',' 5 ',text)     text=re.sub('[6]',' 6 ',text)
text=re.sub('[7]',' 7 ',text)     text=re.sub('[8]',' 8 ',text)     text=re.sub('[9]',' 9 ',text)
text=re.sub('[0]',' 0 ',text)     text=re.sub('[,]',' , ',text)     text=re.sub('[?]',' ? ',text)
text=re.sub('[!]',' ! ',text)     text=re.sub('[$]',' $ ',text)     text=re.sub('[&]',' & ',text)
text=re.sub('[/]',' / ',text)     text=re.sub('[:]',' : ',text)     text=re.sub('[;]',' ; ',text)
text=re.sub('[*]',' * ',text)     text=re.sub('[\ ' ]',' \ ' ',text)     text=re.sub('[\"]',' \"
',text)     text=re.sub('\t',' ',text)     return text

df.drop(columns=['answer tokens','question tokens'],axis=1,inplace=True)
df['encoder_inputs']=df['question'].apply(clean_text)
df['decoder_targets']=df['answer'].apply(clean_text)+' <end>'
df['decoder_inputs']='<start> '+df['answer'].apply(clean_text)+' <end>'

df.head(10)
```

Out[4]:

	question	answer	encoder_inputs	decoder_targets	decoder_inputs
0	hi, how are you doing?	i'm fine. how about yourself?	hi , how are you doing ?	i ' m fine . how about yourself ? <end>	<start> i ' m fine . how about yourself ? <end>
1	i'm fine. how about yourself?	i'm pretty good. thanks for asking.	i ' m fine . how about yourself ?	i ' m pretty good . thanks for asking . <end>	<start> i ' m pretty good . thanks for asking...

2	i'm pretty good. thanks for asking.	no problem. so how have you been?	i ' m pretty good . thanks for asking .	no problem . so how have you been ? <end>	<start> no problem . so how have you been ? ...
	question	answer	encoder_inputs	decoder_targets	decoder_inputs
3	no problem. so how have you been?	i've been great. what about you?	no problem . so how have you been ?	i ' ve been great . what about you ? <end>	<start> i ' ve been great . what about you ? ...
4	i've been great. what about you?	i've been good. i'm in school right now.	i ' ve been great . what about you ?	i ' ve been good . i ' m in school right now ...	<start> i ' ve been good . i ' m in school ri...
5	i've been good. i'm in school right now.	what school do you go to?	i ' ve been good . i ' m in school right now .	what school do you go to ? <end>	<start> what school do you go to ? <end>
6	what school do you go to?	i go to pcc.	what school do you go to ?	i go to pcc . <end>	<start> i go to pcc . <end>
7	i go to pcc.	do you like it there?	i go to pcc .	do you like it there ? <end>	<start> do you like it there ? <end>
8	do you like it there?	it's okay. it's a really big campus.	do you like it there ?	it ' s okay . it ' s a really big campus . <...	<start> it ' s okay . it ' s a really big cam...

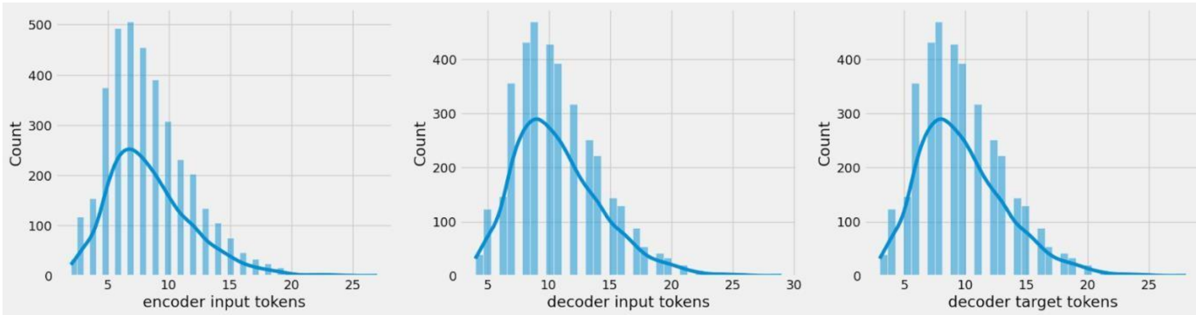
9	it's okay. it's a really big campus.	good luck with school.	it ' s okay . it ' s a really big campus .	good luck with school . <end>	<start> good luck with school . <end>

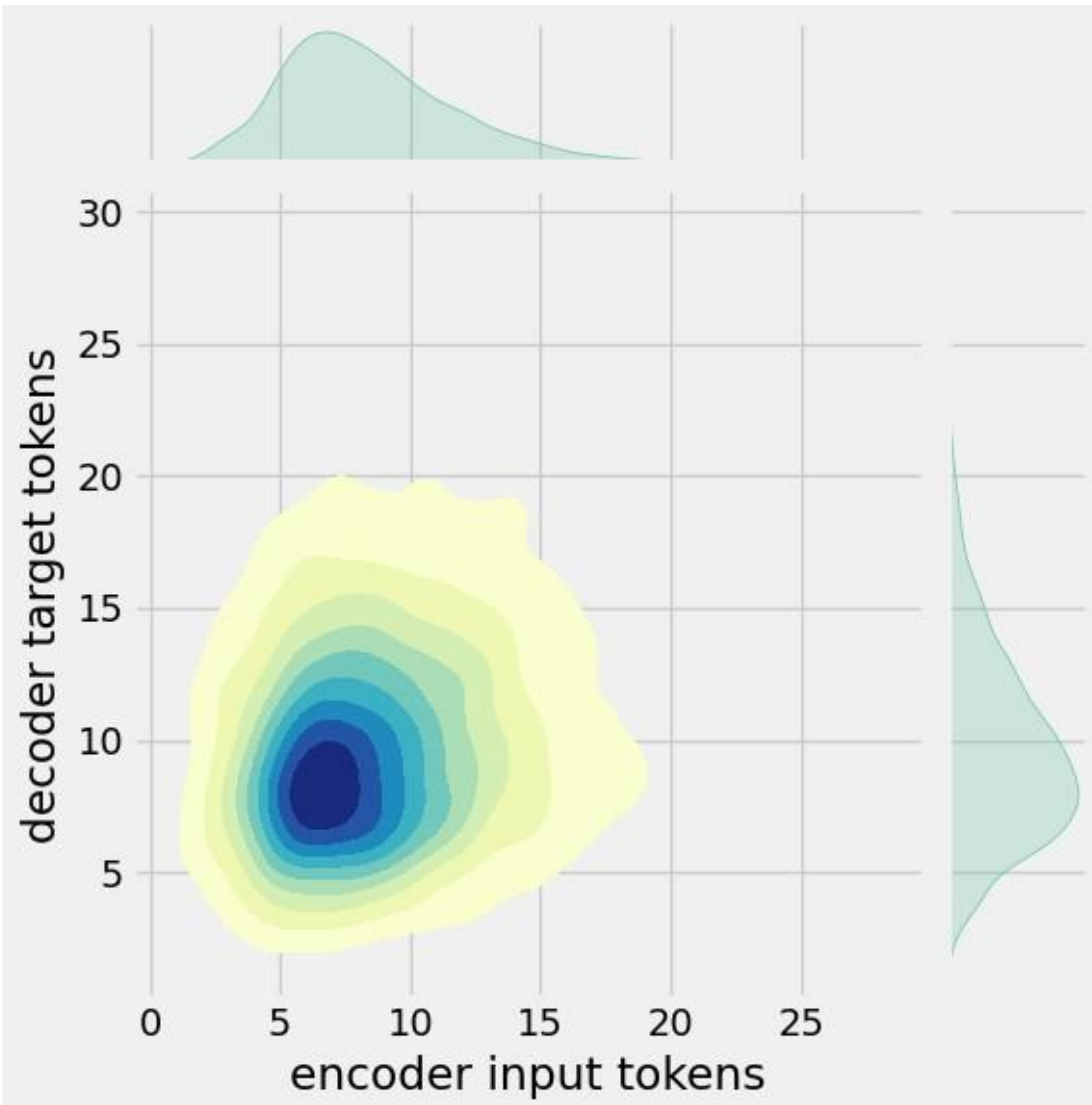
Token Count Analysis and Distribution Visualization

The below code segment extends the data analysis and visualization for the dataset used for training or evaluating a chatbot or conversation model. It calculates the token counts for the encoder inputs, decoder inputs, and decoder targets, and then visualizes the distributions of these token counts.

In [5]:

```
df['encoder input tokens']=df['encoder_inputs'].apply(lambda x:len(x.split())) df['decoder input tokens']=df['decoder_inputs'].apply(lambda x:len(x.split()))
df['decoder target tokens']=df['decoder_targets'].apply(lambda x:len(x.split()))
plt.style.use('fivethirtyeight') fig,ax=plt.subplots(nrows=1,ncols=3,figsize=(20,5)) sns.set_palette('Set2')
sns.histplot(x=df['encoder input tokens'],data=df,kde=True,ax=ax[0]) sns.histplot(x=df['decoder input tokens'],data=df,kde=True,ax=ax[1]) sns.histplot(x=df['decoder target tokens'],data=df,kde=True,ax=ax[2])
sns.jointplot(x='encoder input tokens',y='decoder target tokens',data=df,kind='kde',fill=True,cmap='YlGnBu')
plt.show()
```


Count
encoder input tokens
decoder input tokens
decoder target tokens


decoder target tokens
encoder input tokens

Post-Preprocessing Data Summary and Configuration

The below code segment provides a summary and configuration details after preprocessing the data. It also drops some unnecessary columns and sets parameters for further processing.

Program: **In[6]:**

```
print(f"After preprocessing: {' '.join(df[df['encoder input tokens'].max()==df['encoder input
tokens']]['encoder_inputs'].values.tolist())}") print(f"Max encoder input length: {df['encoder input
tokens'].max()}") print(f"Max decoder input length: {df['decoder input tokens'].max()}") print(f"Max
decoder target length: {df['decoder target tokens'].max()}")

df.drop(columns=['question','answer','encoder input tokens','decoder input tokens','decoder target
tokens'],axis=1,inplace=True) params={
  "vocab_size":2500,
  "max_sequence_length":30,
  "learning_rate":0.008,
  "batch_size":149,
  "lstm_cells":256,
  "embedding_dim":256,     "buffer_size":10000
}
learning_rate=params['learning_rate'] batch_size=params['batch_size']
embedding_dim=params['embedding_dim'] lstm_cells=params['lstm_cells']
vocab_size=params['vocab_size'] buffer_size=params['buffer_size']
max_sequence_length=params['max_sequence_length'] df.head(10)
```

After preprocessing: for example , if your birth date is january 1 2 , 1 9 8 7 , write 0 1 / 1 2 / 8 7 .

Max encoder input length: 27

Max decoder input length: 29

Max decoder target length: 28

Out[6]:

	encoder_inputs	decoder_targets	decoder_inputs
0	hi , how are you doing ?	i ' m fine . how about yourself ? <end>	<start> i ' m fine . how about yourself ? <end>

1	i ' m fine . how about yourself ?	i ' m pretty good . thanks for asking . <end>	<start> i ' m pretty good . thanks for asking...
2	i ' m pretty good . thanks for asking .	no problem . so how have you been ? <end>	<start> no problem . so how have you been ? ...
3	no problem . so how have you been	i ' ve been great . what about you ?	<start> i ' ve been great . what about you

	encoder_inputs	decoder_targets	decoder_inputs
	?	<end>	? ...
4	i ' ve been great . what about you ?	i ' ve been good . i ' m in school right now ...	<start> i ' ve been good . i ' m in school ri...
5	i ' ve been good . i ' m in school right now .	what school do you go to ? <end>	<start> what school do you go to ? <end>
6	what school do you go to ?	i go to pcc . <end>	<start> i go to pcc . <end>
7	i go to pcc .	do you like it there ? <end>	<start> do you like it there ? <end>
8	do you like it there ?	it ' s okay . it ' s a really big campus . <...	<start> it ' s okay . it ' s a really big cam...
9	it ' s okay . it ' s a really big campus .	good luck with school . <end>	<start> good luck with school . <end>

Tokenization

Text Vectorization and Vocabulary Generation

This code segment is responsible for text vectorization and vocabulary generation, which are crucial steps in preparing text data for machine learning or deep learning models. Ln[7]:

```
vectorize_layer=TextVectorization(    max_tokens=vocab_size,
standardize=None,    output_mode='int',
output_sequence_length=max_sequence_length
)
vectorize_layer.adapt(df['encoder_inputs']+' '+df['decoder_targets']+'
<start> <end>')
vocab_size=len(vectorize_layer.get_vocabulary()) print(f'Vocab size:
{len(vectorize_layer.get_vocabulary())}')
print(f'{vectorize_layer.get_vocabulary()[:12]}')
```

Vocab size: 2443

['', '[UNK]', '<end>', '.', '<start>', "'", 'i', '?', 'you', ',', 'the',
'to']

Sequence to ID Conversion and Data Shapes

The below code segment is primarily focused on the conversion of text sequences into numerical IDs using the previously created text vectorization layer. It also provides information about the shapes of the resulting data arrays.

In [8]:

```
def sequences2ids(sequence):
   return vectorize_layer(sequence)

def ids2sequences(ids):
   decode=''     if type(ids)==int:
      ids=[ids]     for id in ids:        decode+=vectorize_layer.get_vocabulary()[id]+' '
return decode

x=sequences2ids(df['encoder_inputs']) yd=sequences2ids(df['decoder_inputs'])
y=sequences2ids(df['decoder_targets'])

print(f'Question sentence: hi , how are you ?') print(f'Question to tokens: {sequences2ids("hi , how are
you ?")[:10]}') print(f'Encoder input shape: {x.shape}') print(f'Decoder input shape: {yd.shape}')
print(f'Decoder target shape: {y.shape}')
```

Question sentence: hi , how are you ?
Question to tokens: [1971   9   45   24   8   7   0   0   0   0]

Encoder input shape: (3725, 30)

Decoder input shape: (3725, 30)

Decoder target shape: (3725, 30)

Sample Encoder and Decoder Inputs and Targets

The below code segment provides a preview of the numerical sequences for encoder inputs, decoder inputs, and decoder targets for a specific data example. In[9]:

```
print(f'Encoder input: {x[0][:12]} ...') print(f'Decoder input: {yd[0][:12]} ...')   # shifted by one
time step of the target as input to decoder is the output of the previous
timestep print(f'Decoder target: {y[0][:12]} ...')
```

Encoder input:
[1971   9   45   24   8 194   7   0   0   0   0   0] ...

Decoder input: [4   6   5   38 646   3   45   41 563   7   2   0] ...

Decoder target: [6   5   38 646   3   45   41 563   7   2   0   0] ...

Data Preprocessing and Batching for Training and Validation

The below code segment focuses on preparing and organizing the data for training and validation of a machine learning model, particularly for sequence-to-sequence tasks such as chatbots or language translation.

In [10]:

```
data=tf.data.Dataset.from_tensor_slices((x,yd,y)) data=data.shuffle(buffer_size)

train_data=data.take(int(.9*len(data))) train_data=train_data.cache()
train_data=train_data.shuffle(buffer_size) train_data=train_data.batch(batch_size)
train_data=train_data.prefetch(tf.data.AUTOTUNE)
train_data_iterator=train_data.as_numpy_iterator()

val_data=data.skip(int(.9*len(data))).take(int(.1*len(data))) val_data=val_data.batch(batch_size)
val_data=val_data.prefetch(tf.data.AUTOTUNE)

_=train_data_iterator.next()
print(f'Number of train batches: {len(train_data)}') print(f'Number of training data:
{len(train_data)*batch_size}') print(f'Number of validation batches: {len(val_data)}')
print(f'Number of validation data: {len(val_data)*batch_size}') print(f'Encoder Input shape (with
batches): {_[0].shape}') print(f'Decoder Input shape (with batches): {_[1].shape}') print(f'Target
Output shape (with batches): {_[2].shape}')
```

Number of train batches: 23

Number of training data: 3427

Number of validation batches: 3

Number of validation data: 447

Encoder Input shape (with batches): (149, 30)

Decoder Input shape (with batches): (149, 30)

Target Output shape (with batches): (149, 30)

6.Build Models

Model building in the context of machine learning and deep learning involves the creation and training of algorithms or neural networks to perform specific tasks. It's a fundamental step in the development of predictive models, classifiers, or any system designed to make intelligent decisions or predictions based on data.

Build Encoder

The below code defines the encoder component of a sequence-to-sequence model, typically used in tasks like chatbots or machine translation.

In [ ]:

In [11]:

```
class Encoder(tf.keras.models.Model):     def __init__(self,units,embedding_dim,vocab_size,*args,**kwargs) ->
None:
      super().__init__(*args,**kwargs)        self.units=units        self.vocab_size=vocab_size
self.embedding_dim=embedding_dim        self.embedding=Embedding(           vocab_size,
embedding_dim,           name='encoder_embedding',           mask_zero=True,
embeddings_initializer=tf.keras.initializers.GlorotNormal()
      )
      self.normalize=LayerNormalization()       self.lstm=LSTM(           units,           dropout=.4,
return_state=True,           return_sequences=True,           name='encoder_lstm',
kernel_initializer=tf.keras.initializers.GlorotNormal()        )         def call(self,encoder_inputs):
self.inputs=encoder_inputs        x=self.embedding(encoder_inputs)        x=self.normalize(x)
x=Dropout(.4)(x)        encoder_outputs,encoder_state_h,encoder_state_c=self.lstm(x)
self.outputs=[encoder_state_h,encoder_state_c]        return encoder_state_h,encoder_state_c

encoder=Encoder(lstm_cells,embedding_dim,vocab_size,name='encoder') encoder.call(_[0])
```

Out[11]:

```
(<tf.Tensor: shape=(149, 256), dtype=float32, numpy=
 array([[ 0.16966951, -0.10419625, -0.12700348, ..., -0.12251794,
          0.10568858,  0.14841646],
        [ 0.08443093,  0.08849293, -0.09065959, ..., -0.00959182,
          0.10152507, -0.12077457],
        [ 0.03628462, -0.02653611, -0.11506603, ..., -0.14669597,
          0.10292757,  0.13625325],
        ...,
        [-0.14210635, -0.12942064, -0.03288083, ...,  0.0568463 ,
         -0.02598592, -0.22455114],
        [ 0.20819993,  0.01196991, -0.09635217, ..., -0.18782297,
          0.10233591,  0.20114912],
        [ 0.1164271 , -0.07769038, -0.06414707, ..., -0.06539135,
         -0.05518465,  0.25142196]], dtype=float32)>,  <tf.Tensor: shape=(149, 256),
dtype=float32, numpy=
 array([[ 0.34589   , -0.30134732, -0.43572   , ..., -0.3102559 ,
          0.34630865,  0.2613009 ],
```

```
[ 0.14154069,  0.17045322, -0.17749965, ..., -0.02712595,
  0.17292541, -0.2922624 ],
[ 0.07106856, -0.0739173 , -0.3641197 , ..., -0.3794833 ,
  0.36470377,  0.23766585],
...,
[-0.2582597 , -0.25323495, -0.06649272, ...,  0.16527973,
 -0.04292646, -0.58768904],
[ 0.43155715,  0.03135502, -0.33463806, ..., -0.47625306,
  0.33486888,  0.35035062],
[ 0.23173636, -0.20141824, -0.22034441, ..., -0.16035017,
 -0.17478186,  0.48899865]], dtype=float32)>)
```

Build Encoder## Build Decoder

The below code provided defines the structure of a Decoder model used in a sequence-to-sequence neural network. This type of architecture is commonly used in tasks like machine translation, text summarization, and chatbot development.

In [12]:

```
class Decoder(tf.keras.models.Model):     def __init__(self,units,embedding_dim,vocab_size,*args,**kwargs) -> None:
        super().__init__(*args,**kwargs)        self.units=units
self.embedding_dim=embedding_dim        self.vocab_size=vocab_size
self.embedding=Embedding(           vocab_size,           embedding_dim,
name='decoder_embedding',           mask_zero=True,
embeddings_initializer=tf.keras.initializers.HeNormal()
        )
        self.normalize=LayerNormalization()        self.lstm=LSTM(           units,
dropout=.4,           return_state=True,           return_sequences=True,
name='decoder_lstm',           kernel_initializer=tf.keras.initializers.HeNormal()
        )        self.fc=Dense(           vocab_size,           activation='softmax',
name='decoder_dense',           kernel_initializer=tf.keras.initializers.HeNormal()        )
def call(self,decoder_inputs,encoder_states):        x=self.embedding(decoder_inputs)
x=self.normalize(x)        x=Dropout(.4)(x)
x,decoder_state_h,decoder_state_c=self.lstm(x,initial_state=encoder_s tates)
        x=self.normalize(x)        x=Dropout(.4)(x)
        return self.fc(x)

decoder=Decoder(lstm_cells,embedding_dim,vocab_size,name='decoder') decoder(_[1][:1],encoder(_[0][:1]))
```

Out[12]:

<tf.Tensor: shape=(1, 30, 2443), dtype=float32, numpy= array([[[3.4059247e-04, 5.7348556e-05, 2.1294907e-05, ..., 7.2067953e-05, 1.5453645e-03, 2.3599296e-04],

[1.4662130e-03, 8.0250365e-06, 5.4062020e-05, ..., 1.9187471e-05, 9.7244098e-05, 7.6433855e-05],

[9.6929165e-05, 2.7441782e-05, 1.3761305e-03, ...,

3.6009602e-05, 1.5537882e-04, 1.8397317e-04], ...,

[1.9002777e-03, 6.9266016e-04, 1.4346189e-04, ..., 1.9552530e-04, 1.7106640e-05, 1.0252406e-04],

[1.9002777e-03, 6.9266016e-04, 1.4346189e-04, ..., 1.9552530e-04, 1.7106640e-05, 1.0252406e-04],

[1.9002777e-03, 6.9266016e-04, 1.4346189e-04, ...,

1.9552530e-04, 1.7106640e-05, 1.0252406e-04]]], dtype=float32)>

Build Training Model

The below code defines a `ChatBotTrainer` class, which is responsible for training and evaluating a chatbot model. This class uses an encoder-decoder architecture and incorporates loss and accuracy functions, training steps, and testing steps ln[13]:

```
class ChatBotTrainer(tf.keras.models.Model):     def
__init__(self,encoder,decoder,*args,**kwargs):
    super().__init__(*args,**kwargs)        self.encoder=encoder
self.decoder=decoder

  def loss_fn(self,y_true,y_pred):        loss=self.loss(y_true,y_pred)
mask=tf.math.logical_not(tf.math.equal(y_true,0))
mask=tf.cast(mask,dtype=loss.dtype)        loss*=mask        return
tf.reduce_mean(loss)
      def accuracy_fn(self,y_true,y_pred):
    pred_values = tf.cast(tf.argmax(y_pred, axis=-1), dtype='int64')        correct = tf.cast(tf.equal(y_true,
pred_values), dtype='float64')        mask = tf.cast(tf.greater(y_true, 0), dtype='float64')        n_correct =
tf.keras.backend.sum(mask * correct)        n_total = tf.keras.backend.sum(mask)        return n_correct /
n_total

  def call(self,inputs):
    encoder_inputs,decoder_inputs=inputs
encoder_states=self.encoder(encoder_inputs)        return
self.decoder(decoder_inputs,encoder_states)     def train_step(self,batch):
    encoder_inputs,decoder_inputs,y=batch        with tf.GradientTape()
as tape:
      encoder_states=self.encoder(encoder_inputs,training=True)
y_pred=self.decoder(decoder_inputs,encoder_states,training=True)          loss=self.loss_fn(y,y_pred)
```

```
        acc=self.accuracy_fn(y,y_pred)

      variables=self.encoder.trainable_variables+self.decoder.trainable_var iables
      grads=tape.gradient(loss,variables)
self.optimizer.apply_gradients(zip(grads,variables))
metrics={'loss':loss,'accuracy':acc}        return metrics
       def test_step(self,batch):
      encoder_inputs,decoder_inputs,y=batch
encoder_states=self.encoder(encoder_inputs,training=True)
y_pred=self.decoder(decoder_inputs,encoder_states,training=True)        loss=self.loss_fn(y,y_pred)
acc=self.accuracy_fn(y,y_pred)        metrics={'loss':loss,'accuracy':acc}
      return metrics
```

In[14]:

```
model=ChatBotTrainer(encoder,decoder,name='chatbot_trainer') model.compile(
   loss=tf.keras.losses.SparseCategoricalCrossentropy(),
optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),     weighted_metrics=['loss','accuracy']
)
model(_[:2])
```

Out[14]:

```
<tf.Tensor: shape=(149, 30, 2443), dtype=float32, numpy= array([[[3.40592262e-04, 5.73484940e-05,
2.12948853e-05, ...,         7.20679745e-05, 1.54536311e-03, 2.35993255e-04],

       [1.46621116e-03, 8.02504110e-06, 5.40619949e-05, ...,         1.91874733e-05, 9.72440175e-05,
7.64339056e-05],

       [9.69291723e-05, 2.74417835e-05, 1.37613132e-03, ...,

        3.60095728e-05, 1.55378671e-04, 1.83973272e-04],        ...,

       [1.90027885e-03, 6.92659756e-04, 1.43461803e-04, ...,         1.95525470e-04, 1.71066222e-05,
1.02524005e-04],

       [1.90027885e-03, 6.92659756e-04, 1.43461803e-04, ...,         1.95525470e-04, 1.71066222e-05,
1.02524005e-04],

       [1.90027885e-03, 6.92659756e-04, 1.43461803e-04, ...,

        1.95525470e-04, 1.71066222e-05, 1.02524005e-04]],


      [[9.24730921e-05, 3.46553512e-04, 2.07866033e-05, ...,         3.65934626e-04, 7.63039337e-04,
5.52638434e-04],

       [8.46863186e-05, 3.65541164e-05, 2.54740953e-05, ...,         7.12379551e-05, 3.62201303e-04,
4.16714087e-04],

       [2.30146630e-04, 3.91469621e-06, 2.72463716e-04, ...,

        9.26126595e-05, 1.03836363e-04, 1.40792166e-04],
```

-

...,

[6.84961735e-04, 9.07644513e 04, 2.86691647e-04, ...,

3.87946144e-04, 6.09236558e-05, 1.12995331e-05],

[6.84961735e-04, 9.07644513e-04, 2.86691647e-04, ..., 3.87946144e-04, 6.09236558e-05, 1.12995331e-05],

[6.84961735e-04, 9.07644513e-04, 2.86691647e-04, ...,

3.87946144e-04, 6.09236558e-05, 1.12995322e-05]],

[[1.19036995e-03, 8.10516722e-05, 2.42324077e-05, ..., 4.99442758e-05, 6.67208573e-04, 9.55566764e-04],

[1.53046989e-04, 9.76863957e-05, 4.96972689e-06, ..., 3.24743196e-05, 2.12563842e-04, 1.18708890e-03],

[9.40205529e-04, 1.80782794e-04, 7.26205144e-06, ...,

1.96355060e-04, 8.16940737e-05, 1.38416886e-03],

...,

[3.52622545e-03, 1.26781175e-03, 1.02695449e-04, ..., 2.35450850e-03, 3.25187625e-06, 9.46984728e-05],

[3.52622545e-03, 1.26781175e-03, 1.02695449e-04, ..., 2.35450850e-03, 3.25187625e-06, 9.46984728e-05],

[3.52622545e-03, 1.26781175e-03, 1.02695449e-04, ...,

2.35450850e-03, 3.25187625e-06, 9.46984728e-05]],

...,

[[9.03617911e-05, 1.57651404e-04, 1.02747028e-04, ..., 2.20922651e-04, 3.61504179e-04, 2.32456136e-03],

[1.55469708e-04, 1.53608169e-04, 1.14945491e-04, ..., 1.88878359e-04, 5.11967926e-04, 5.13108505e-04],

[8.27641197e-05, 2.83437112e-05, 6.29429938e-04, ...,

2.15980137e-04, 3.02832137e-04, 1.77760507e-04],

...,

```
23/23 [==============================] -                                      -                          -
                              -                                 -

      [2.41102395e-03, 1.29279669e-03, 9.11735406e-05, ...,       4.06600971e-04, 7.58682154e-06,
6.05909081e-05],

      [2.41102395e-03, 1.29279669e-03, 9.11735406e-05, ...,       4.06600971e-04, 7.58682154e-06,
6.05909081e-05],

      [2.41102395e-03, 1.29279669e-03, 9.11735406e-05, ...,

       4.06600971e-04, 7.58682154e-06, 6.05909081e-05]],


     [[3.99837241e-04, 2.36026899e-05, 6.89777007e-05, ...,       5.94239136e-05, 4.32556757e-04,
4.60232928e-04],

      [3.88111075e-04, 8.31133584e-05, 1.11861555e-04, ...,       3.03280340e-05, 2.54765386e-04,
2.82170397e-04],

      [2.12516752e-03, 7.19837190e-05, 1.88700986e-04, ...,

       1.86366087e-04, 7.02239413e-05, 2.54370330e-04],

      ...,

      [4.56329063e-03, 2.23812275e-03, 2.37343236e-04, ...,       2.64523784e-04, 4.05454011e-05,
1.55662783e-04],

      [4.56329063e-03, 2.23812275e-03, 2.37343236e-04, ...,       2.64523784e-04, 4.05454011e-05,
1.55662783e-04],

      [4.56329063e-03, 2.23812275e-03, 2.37343236e-04, ...,

       2.64523784e-04, 4.05454011e-05, 1.55662783e-04]],


     [[3.24600202e-04, 9.31067043e-05, 4.60048941e-05, ...,

       6.66230699e-05, 5.76460850e-04, 1.52416309e-04],

      [7.51478728e-05, 7.63997741e-05, 2.09082973e-05, ...,       2.55555002e-04, 2.28998848e-04,
4.37303359e-04],

      [1.03114333e-04, 1.55743372e-04, 9.97955431e-06, ...,

       1.12485175e-03, 4.80950950e-03, 6.83143327e-04],       ...,

      [5.20280097e-03, 3.23211338e-04, 2.47709468e-05, ...,       3.07609705e-04, 6.09844255e-06,
8.61325825e-05],

      [5.20280097e-03, 3.23211338e-04, 2.47709468e-05, ...,       3.07609705e-04, 6.09844255e-06,
8.61325825e-05],
                                                                                                  -


                                                                        -                 -
```

23/23 [==============================] - - -
- -

[5.20280097e-03, 3.23211338e-04, 2.47709468e-05, ..., 3.07609705e-04, 6.09844255e-06, 8.61325825e-05]]], dtype=float32)>

7.Train Model

We have trained a neural network model for 68 epochs. The training process involves monitoring the loss and accuracy on both the training and validation sets.

In [15]:

```
history=model.fit(     train_data,
epochs=100,     validation_data=val_data,
callbacks=[
       tf.keras.callbacks.TensorBoard(log_dir='logs'),
tf.keras.callbacks.ModelCheckpoint('ckpt',verbose=1,save_best_only=Tr ue)
    ]
)
```

Epoch 1/100

23/23 [==============================] - ETA: 0s - loss: 1.6590 - accuracy: 0.2180

Epoch 1: val_loss improved from inf to 1.21875, saving model to ckpt 23/23 [==============================] - 68s 3s/step - loss: 1.6515 -

accuracy: 0.2198 - val_loss: 1.2187 - val_accuracy: 0.3072

Epoch 2/100

23/23 [==============================] - ETA: 0s - loss: 1.2327 - accuracy: 0.3087

Epoch 2: val_loss improved from 1.21875 to 1.10877, saving model to ckpt 23/23 [==============================] - 53s 2s/step - loss: 1.2287 -

accuracy: 0.3092 - val_loss: 1.1088 - val_accuracy: 0.3415

Epoch 3/100

23/23 [==============================] - ETA: 0s - loss: 1.1008 - accuracy: 0.3368

Epoch 3: val_loss did not improve from 1.10877

23/23 [==============================] - 22s 973ms/step - loss: 1.0984 - accuracy: 0.3370 - val_loss: 1.1161 - val_accuracy: 0.3315

Epoch 4/100

23/23 [==============================] - ETA: 0s - loss: 1.0209 - accuracy: 0.3536

-

- -

```
23/23 [==============================] -                                -                    -
                      -                         -

Epoch 4: val_loss improved from 1.10877 to 0.95189, saving model to ckpt 23/23
[==============================] - 53s 2s/step - loss: 1.0186 -
accuracy: 0.3540 - val_loss: 0.9519 - val_accuracy: 0.3718

Epoch 5/100

23/23 [==============================] - ETA: 0s - loss: 0.9622 - accuracy: 0.3673

Epoch 5: val_loss did not improve from 0.95189

                                                          23s 979ms/step loss: 0.9672
accuracy: 0.3670 val_loss: 0.9642 val_accuracy: 0.3666

Epoch 6/100

23/23 [==============================] - ETA: 0s - loss: 0.9159 - accuracy: 0.3801

Epoch 6: val_loss improved from 0.95189 to 0.94015, saving model to ckpt 23/23
[==============================] - 53s 2s/step - loss: 0.9182 -
accuracy: 0.3796 - val_loss: 0.9401 - val_accuracy: 0.3598

Epoch 7/100

23/23 [==============================] - ETA: 0s - loss: 0.8737 - accuracy: 0.3908

Epoch 7: val_loss improved from 0.94015 to 0.83293, saving model to ckpt 23/23
[==============================] - 52s 2s/step - loss: 0.8746 -
accuracy: 0.3900 - val_loss: 0.8329 - val_accuracy: 0.4180

Epoch 8/100

23/23 [==============================] - ETA: 0s - loss: 0.8389 - accuracy: 0.4013

Epoch 8: val_loss improved from 0.83293 to 0.77748, saving model to ckpt 23/23
[==============================] - 53s 2s/step - loss: 0.8395 -
accuracy: 0.4013 - val_loss: 0.7775 - val_accuracy: 0.4305

Epoch 9/100

23/23 [==============================] - ETA: 0s - loss: 0.8148 - accuracy: 0.4094

Epoch 9: val_loss did not improve from 0.77748

23/23 [==============================] - 23s 983ms/step - loss: 0.8187 -
accuracy: 0.4084 - val_loss: 0.8608 - val_accuracy: 0.3830

Epoch 10/100

                                                                                -

                                                        -                 -
```

23/23 [==============================] - - -

- -

23/23 [==============================] - ETA: 0s - loss: 0.7889 - accuracy: 0.4200

Epoch 10: val_loss improved from 0.77748 to 0.73131, saving model to ckpt 23/23 [==============================] - 53s 2s/step - loss: 0.7923 -

accuracy: 0.4188 - val_loss: 0.7313 - val_accuracy: 0.4515

Epoch 11/100

23/23 [==============================] - ETA: 0s - loss: 0.7624 - accuracy: 0.4284

Epoch 11: val_loss did not improve from 0.73131

23/23 [==============================] - 22s 965ms/step - loss: 0.7615 - accuracy: 0.4282 - val_loss: 0.8036 - val_accuracy: 0.4472

Epoch 12/100

23/23 [==============================] - ETA: 0s - loss: 0.7433 - accuracy: 0.4361

Epoch 12: val_loss did not improve from 0.73131

23/23 [==============================] - 23s 984ms/step - loss: 0.7452 - accuracy: 0.4354 - val_loss: 0.7384 - val_accuracy: 0.4623

Epoch 13/100

23/23 [==============================] - ETA: 0s - loss: 0.7246 - accuracy: 0.4493

Epoch 13: val_loss did not improve from 0.73131

23/23 [==============================] - 23s 988ms/step - loss: 0.7281

accuracy: 0.4488 - val_loss: 0.8017 - val_accuracy: 0.4449

Epoch 14/100

23/23 [==============================] - ETA: 0s loss: 0.7080 accuracy: 0.4513

Epoch 14: val_loss did not improve from 0.73131

-

- -

23/23 [==============================] - - -
- -

-
23s 995ms/step loss: 0.7080

accuracy: 0.4509 val_loss: 0.7568 val_accuracy: 0.4259

Epoch 15/100

23/23 [==============================] - ETA: 0s - loss: 0.6853 - accuracy: 0.4620

Epoch 15: val_loss did not improve from 0.73131

23/23 [==============================] - 22s 974ms/step - loss: 0.6826
accuracy: 0.4616 - val_loss: 0.7376 - val_accuracy: 0.4502

Epoch 16/100

23/23 [==============================] - ETA: 0s - loss: 0.6731 - accuracy: 0.4673

Epoch 16: val_loss did not improve from 0.73131

23/23 [==============================] - 23s 983ms/step - loss: 0.6733 -
accuracy: 0.4672 - val_loss: 0.7646 - val_accuracy: 0.4538

Epoch 17/100

23/23 [==============================] - ETA: 0s - loss: 0.6576 - accuracy: 0.4732

Epoch 17: val_loss improved from 0.73131 to 0.66131, saving model to ckpt 23/23
[==============================] - 52s 2s/step - loss: 0.6539 -
accuracy: 0.4738 - val_loss: 0.6613 - val_accuracy: 0.4714

Epoch 18/100

23/23 [==============================] - ETA: 0s - loss: 0.6468 - accuracy: 0.4807

Epoch 18: val_loss improved from 0.66131 to 0.65303, saving model to ckpt 23/23
[==============================] - 53s 2s/step - loss: 0.6458 -
accuracy: 0.4805 - val_loss: 0.6530 - val_accuracy: 0.4993

Epoch 19/100

23/23 [==============================] - ETA: 0s - loss: 0.6353 - accuracy: 0.4881

- -

23/23 [==============================] - - -
- -

-

Epoch 19: val_loss did not improve from 0.65303

23/23 [==============================] - 23s 994ms/step - loss: 0.6357 -
accuracy: 0.4876 - val_loss: 0.7331 - val_accuracy: 0.4677

Epoch 20/100

23/23 [==============================] - ETA: 0s - loss: 0.6194 - accuracy: 0.4968

Epoch 20: val_loss improved from 0.65303 to 0.55054, saving model to ckpt 23/23
[==============================] - 54s 2s/step - loss: 0.6188 - accuracy: 0.4967 - val_loss: 0.5505 -
val_accuracy: 0.5221

Epoch 21/100

23/23 [==============================] - ETA: 0s - loss: 0.6160 - accuracy: 0.4978

Epoch 21: val_loss did not improve from 0.55054

23/23 [==============================] - 23s 987ms/step - loss: 0.6182 - accuracy: 0.4965 -
val_loss: 0.6790 - val_accuracy: 0.4979

Epoch 22/100

23/23 [==============================] - ETA: 0s - loss: 0.6011 - accuracy: 0.5052

Epoch 22: val_loss did not improve from 0.55054

23/23 [==============================] - 23s 996ms/step - loss: 0.6011 -
accuracy: 0.5051 - val_loss: 0.6221 - val_accuracy: 0.5277

Epoch 23/100

23/23 [==============================] - ETA: 0s loss: 0.5950 accuracy: 0.5079

Epoch 23: val_loss did not improve from 0.55054

23s 987ms/step loss: 0.5934

accuracy: 0.5081 val_loss: 0.6142 val_accuracy: 0.5198

-

- -

23/23 [==============================] - - -
- -
-

Epoch 24/100

23/23 [==============================] - ETA: 0s - loss: 0.5810 - accuracy: 0.5160

Epoch 24: val_loss did not improve from 0.55054

23/23 [==============================] - 22s 971ms/step - loss: 0.5803
accuracy: 0.5170 - val_loss: 0.5759 - val_accuracy: 0.5137

Epoch 25/100

23/23 [==============================] - ETA: 0s - loss: 0.5716 - accuracy: 0.5227

Epoch 25: val_loss did not improve from 0.55054

23/23 [==============================] - 23s 986ms/step - loss: 0.5733 -
accuracy: 0.5229 - val_loss: 0.6344 - val_accuracy: 0.5169

Epoch 26/100

23/23 [==============================] - ETA: 0s - loss: 0.5676 - accuracy: 0.5225

Epoch 26: val_loss did not improve from 0.55054

23/23 [==============================] - 22s 963ms/step - loss: 0.5708 -
accuracy: 0.5210 - val_loss: 0.6254 - val_accuracy: 0.4882

Epoch 27/100

23/23 [==============================] - ETA: 0s - loss: 0.5616 - accuracy: 0.5291

Epoch 27: val_loss did not improve from 0.55054

23/23 [==============================] - 23s 988ms/step - loss: 0.5624 -
accuracy: 0.5280 - val_loss: 0.6774 - val_accuracy: 0.5379

Epoch 28/100

23/23 [==============================] - ETA: 0s - loss: 0.5531 - accuracy: 0.5318

-

- -

23/23 [==============================] - - -
- -

-

Epoch 28: val_loss did not improve from 0.55054

23/23 [==============================] - 22s 949ms/step - loss: 0.5543 -
accuracy: 0.5310 - val_loss: 0.7284 - val_accuracy: 0.5302

Epoch 29/100

23/23 [==============================] - ETA: 0s - loss: 0.5398 - accuracy: 0.5389

Epoch 29: val_loss did not improve from 0.55054

23/23 [==============================] - 23s 1s/step - loss: 0.5391 - accuracy: 0.5398 - val_loss: 0.7385 - val_accuracy: 0.5193

Epoch 30/100

23/23 [==============================] - ETA: 0s - loss: 0.5375 - accuracy: 0.5416

Epoch 30: val_loss improved from 0.55054 to 0.50346, saving model to ckpt 23/23 [==============================] - 53s 2s/step - loss: 0.5384 - accuracy: 0.5417 - val_loss: 0.5035 - val_accuracy: 0.5411

Epoch 31/100

23/23 [==============================] - ETA: 0s - loss: 0.5270 - accuracy: 0.5481

Epoch 31: val_loss did not improve from 0.50346

23/23 [==============================] - 22s 958ms/step - loss: 0.5262
accuracy: 0.5477 - val_loss: 0.5805 - val_accuracy: 0.5457

Epoch 32/100

23/23 [==============================] - ETA: 0s loss: 0.5304 accuracy: 0.5447

Epoch 32: val_loss did not improve from 0.50346

22s 963ms/step loss: 0.5329
accuracy: 0.5435 val_loss: 0.5374 val_accuracy: 0.5725

-

- -

23/23 [==============================] - - -

- -

-

Epoch 33/100

23/23 [==============================] - ETA: 0s - loss: 0.5196 - accuracy: 0.5520

Epoch 33: val_loss did not improve from 0.50346

23/23 [==============================] - 23s 975ms/step - loss: 0.5211
accuracy: 0.5518 - val_loss: 0.6217 - val_accuracy: 0.5066

Epoch 34/100

23/23 [==============================] - ETA: 0s - loss: 0.5129 - accuracy: 0.5558

Epoch 34: val_loss did not improve from 0.50346

23/23 [==============================] - 23s 1000ms/step - loss: 0.5129 -
accuracy: 0.5556 - val_loss: 0.6070 - val_accuracy: 0.5653

Epoch 35/100

23/23 [==============================] - ETA: 0s - loss: 0.5059 - accuracy: 0.5620

Epoch 35: val_loss did not improve from 0.50346

23/23 [==============================] - 22s 966ms/step - loss: 0.5081 -
accuracy: 0.5614 - val_loss: 0.6153 - val_accuracy: 0.5452

Epoch 36/100

23/23 [==============================] - ETA: 0s - loss: 0.5037 - accuracy: 0.5619

Epoch 36: val_loss did not improve from 0.50346

23/23 [==============================] - 23s 980ms/step - loss: 0.5063 -
accuracy: 0.5617 - val_loss: 0.5328 - val_accuracy: 0.5873

Epoch 37/100

23/23 [==============================] - ETA: 0s - loss: 0.4977 - accuracy: 0.5682

-

- -

23/23 [==============================] - - -
- -

-

Epoch 37: val_loss did not improve from 0.50346

23/23 [==============================] - 22s 969ms/step - loss: 0.4980 -
accuracy: 0.5682 - val_loss: 0.5976 - val_accuracy: 0.5693

Epoch 38/100

23/23 [==============================] - ETA: 0s - loss: 0.4939 - accuracy: 0.5704

Epoch 38: val_loss did not improve from 0.50346

23/23 [==============================] - 23s 993ms/step - loss: 0.4953 - accuracy: 0.5687 -
val_loss: 0.5937 - val_accuracy: 0.5236

Epoch 39/100

23/23 [==============================] - ETA: 0s - loss: 0.4860 - accuracy: 0.5758

Epoch 39: val_loss did not improve from 0.50346

23/23 [==============================] - 23s 986ms/step - loss: 0.4868 - accuracy: 0.5746 -
val_loss: 0.6155 - val_accuracy: 0.5457

Epoch 40/100

23/23 [==============================] - ETA: 0s - loss: 0.4809 - accuracy: 0.5778

Epoch 40: val_loss did not improve from 0.50346

23/23 [==============================] - 23s 1s/step - loss: 0.4821 accuracy: 0.5760 - val_loss:
0.5046 - val_accuracy: 0.5662

Epoch 41/100

23/23 [==============================] - ETA: 0s loss: 0.4781 accuracy: 0.5817

Epoch 41: val_loss did not improve from 0.50346

23s 990ms/step loss: 0.4782

-

- -

23/23 [==============================] - - -
- -

-
accuracy: 0.5821 val_loss: 0.5256 val_accuracy: 0.5907

Epoch 42/100

23/23 [==============================] - ETA: 0s - loss: 0.4713 - accuracy: 0.5836

Epoch 42: val_loss did not improve from 0.50346

23/23 [==============================] - 23s 982ms/step - loss: 0.4729
accuracy: 0.5824 - val_loss: 0.6387 - val_accuracy: 0.5456

Epoch 43/100

23/23 [==============================] - ETA: 0s - loss: 0.4641 - accuracy: 0.5904

Epoch 43: val_loss did not improve from 0.50346

23/23 [==============================] - 23s 1s/step - loss: 0.4627 -
accuracy: 0.5908 - val_loss: 0.5668 - val_accuracy: 0.5741

Epoch 44/100

23/23 [==============================] - ETA: 0s - loss: 0.4608 - accuracy: 0.5921

Epoch 44: val_loss improved from 0.50346 to 0.49920, saving model to ckpt 23/23
[==============================] - 53s 2s/step - loss: 0.4618 -
accuracy: 0.5920 - val_loss: 0.4992 - val_accuracy: 0.5768

Epoch 45/100

23/23 [==============================] - ETA: 0s - loss: 0.4592 - accuracy: 0.5902

Epoch 45: val_loss did not improve from 0.49920

23/23 [==============================] - 22s 970ms/step - loss: 0.4599 -
accuracy: 0.5887 - val_loss: 0.5423 - val_accuracy: 0.5854

Epoch 46/100

-

- -

23/23 [==============================] - - -
- -

-
23/23 [==============================] - ETA: 0s - loss: 0.4535 - accuracy: 0.5978

Epoch 46: val_loss improved from 0.49920 to 0.48429, saving model to ckpt 23/23
[==============================] - 53s 2s/step - loss: 0.4552 -
accuracy: 0.5966 - val_loss: 0.4843 - val_accuracy: 0.6049

Epoch 47/100

23/23 [==============================] - ETA: 0s - loss: 0.4528 - accuracy: 0.5987

Epoch 47: val_loss improved from 0.48429 to 0.47868, saving model to ckpt 23/23
[==============================] - 54s 2s/step - loss: 0.4537 - accuracy: 0.5990 - val_loss: 0.4787 - val_accuracy: 0.5906

Epoch 48/100

23/23 [==============================] - ETA: 0s - loss: 0.4441 - accuracy: 0.6016

Epoch 48: val_loss did not improve from 0.47868

23/23 [==============================] - 23s 982ms/step - loss: 0.4439 - accuracy: 0.6025 - val_loss: 0.5746 - val_accuracy: 0.5542

Epoch 49/100

23/23 [==============================] - ETA: 0s - loss: 0.4436 - accuracy: 0.6041

Epoch 49: val_loss did not improve from 0.47868

23/23 [==============================] - 22s 951ms/step - loss: 0.4432
accuracy: 0.6045 - val_loss: 0.5058 - val_accuracy: 0.5753

Epoch 50/100

23/23 [==============================] - ETA: 0s loss: 0.4435 accuracy: 0.6033

Epoch 50: val_loss did not improve from 0.47868

22s 949ms/step loss: 0.4441
-

- -

23/23 [==============================] - - -
- -

-
accuracy: 0.6043 val_loss: 0.6037 val_accuracy: 0.5473

Epoch 51/100

23/23 [==============================] - ETA: 0s - loss: 0.4382 - accuracy: 0.6069

Epoch 51: val_loss did not improve from 0.47868

23/23 [==============================] - 22s 957ms/step - loss: 0.4383
accuracy: 0.6067 - val_loss: 0.5206 - val_accuracy: 0.6154

Epoch 52/100

23/23 [==============================] - ETA: 0s - loss: 0.4293 - accuracy: 0.6125

Epoch 52: val_loss did not improve from 0.47868

23/23 [==============================] - 23s 971ms/step - loss: 0.4284 -
accuracy: 0.6123 - val_loss: 0.4997 - val_accuracy: 0.5840

Epoch 53/100

23/23 [==============================] - ETA: 0s - loss: 0.4309 - accuracy: 0.6109

Epoch 53: val_loss improved from 0.47868 to 0.42987, saving model to ckpt 23/23
[==============================] - 52s 2s/step - loss: 0.4317 -
accuracy: 0.6094 - val_loss: 0.4299 - val_accuracy: 0.6062

Epoch 54/100

23/23 [==============================] - ETA: 0s - loss: 0.4292 - accuracy: 0.6120

Epoch 54: val_loss did not improve from 0.42987

23/23 [==============================] - 22s 980ms/step - loss: 0.4309 -
accuracy: 0.6115 - val_loss: 0.6996 - val_accuracy: 0.5592

Epoch 55/100

-

- -

23/23 [==============================] - - -
- -

-

23/23 [==============================] - ETA: 0s - loss: 0.4225 - accuracy: 0.6115

Epoch 55: val_loss did not improve from 0.42987

23/23 [==============================] - 22s 976ms/step - loss: 0.4224 -
accuracy: 0.6102 - val_loss: 0.5500 - val_accuracy: 0.5769

Epoch 56/100

23/23 [==============================] - ETA: 0s - loss: 0.4220 - accuracy: 0.6180

Epoch 56: val_loss did not improve from 0.42987

23/23 [==============================] - 23s 995ms/step - loss: 0.4236 - accuracy: 0.6169 - val_loss: 0.5689 - val_accuracy: 0.5817

Epoch 57/100

23/23 [==============================] - ETA: 0s - loss: 0.4173 - accuracy: 0.6210

Epoch 57: val_loss did not improve from 0.42987

23/23 [==============================] - 22s 976ms/step - loss: 0.4161 - accuracy: 0.6217 - val_loss: 0.4614 - val_accuracy: 0.6048

Epoch 58/100

23/23 [==============================] - ETA: 0s - loss: 0.4183 - accuracy: 0.6198

Epoch 58: val_loss did not improve from 0.42987

23/23 [==============================] - 23s 1s/step - loss: 0.4183 accuracy: 0.6201 - val_loss: 0.4372 - val_accuracy: 0.6067

Epoch 59/100

23/23 [==============================] - ETA: 0s loss: 0.4120 accuracy: 0.6251

Epoch 59: val_loss did not improve from 0.42987

-

- -

23/23 [==============================] - - -
- -

-
23s 994ms/step loss: 0.4136
accuracy: 0.6237 val_loss: 0.6183 val_accuracy: 0.5948

Epoch 60/100

23/23 [==============================] - ETA: 0s - loss: 0.4090 - accuracy: 0.6239

Epoch 60: val_loss did not improve from 0.42987

23/23 [==============================] - 23s 980ms/step - loss: 0.4101
accuracy: 0.6225 - val_loss: 0.5042 - val_accuracy: 0.6161

Epoch 61/100

23/23 [==============================] - ETA: 0s - loss: 0.4051 - accuracy: 0.6314

Epoch 61: val_loss did not improve from 0.42987

23/23 [==============================] - 23s 1s/step - loss: 0.4077 -
accuracy: 0.6296 - val_loss: 0.5100 - val_accuracy: 0.6128

Epoch 62/100

23/23 [==============================] - ETA: 0s - loss: 0.4016 - accuracy: 0.6326

Epoch 62: val_loss did not improve from 0.42987

23/23 [==============================] - 24s 1s/step - loss: 0.4029 -
accuracy: 0.6322 - val_loss: 0.5295 - val_accuracy: 0.6005

Epoch 63/100

23/23 [==============================] - ETA: 0s - loss: 0.4049 - accuracy: 0.6323

Epoch 63: val_loss did not improve from 0.42987

23/23 [==============================] - 23s 981ms/step - loss: 0.4069 -
accuracy: 0.6316 - val_loss: 0.5103 - val_accuracy: 0.6088

Epoch 64/100

23/23 [==============================] - ETA: 0s - loss: 0.3951 - accuracy: 0.6335

Epoch 64: val_loss did not improve from 0.42987

23/23 [==============================] - - -

- -

-

23/23 [==============================] - 22s 981ms/step - loss: 0.3943 -

accuracy: 0.6341 - val_loss: 0.5366 - val_accuracy: 0.5869

Epoch 65/100

23/23 [==============================] - ETA: 0s - loss: 0.3967 - accuracy: 0.6344

Epoch 65: val_loss improved from 0.42987 to 0.40702, saving model to ckpt 23/23 [==============================] - 53s 2s/step - loss: 0.3972 - accuracy: 0.6352 - val_loss: 0.4070 - val_accuracy: 0.6452

Epoch 66/100

23/23 [==============================] - ETA: 0s - loss: 0.3942 - accuracy: 0.6351

Epoch 66: val_loss did not improve from 0.40702

23/23 [==============================] - 22s 961ms/step - loss: 0.3954 - accuracy: 0.6337 - val_loss: 0.4963 - val_accuracy: 0.6039

Epoch 67/100

23/23 [==============================] - ETA: 0s - loss: 0.3884 - accuracy: 0.6409

Epoch 67: val_loss did not improve from 0.40702

23/23 [==============================] - 22s 951ms/step - loss: 0.3879 -

accuracy: 0.6424 - val_loss: 0.4651 - val_accuracy: 0.6276

Epoch 68/100

23/23 [==============================] - ETA: 0s - loss: 0.3876 - accuracy: 0.6398

Epoch 68: val_loss improved from 0.40702 to 0.38016, saving model to ckpt 52s 2s/step - loss: 0.3870 - accuracy: 0.6388 val_loss: 0.3802 val_accuracy: 0.6614

Epoch 69/100

23/23 [==============================] - ETA: 0s - loss: 0.3897 - accuracy: 0.6394

Epoch 69: val_loss did not improve from 0.38016

23/23 [==============================] - 22s 961ms/step - loss: 0.3895

accuracy: 0.6395 - val_loss: 0.4046 - val_accuracy: 0.6587

- -

23/23 [==============================] -

-

Epoch 70/100

23/23 [==============================] - ETA: 0s - loss: 0.3855 - accuracy: 0.6433

Epoch 70: val_loss did not improve from 0.38016

23/23 [==============================] - 22s 967ms/step - loss: 0.3870 - accuracy: 0.6432 - val_loss: 0.4162 - val_accuracy: 0.6475

Epoch 71/100

23/23 [==============================] - ETA: 0s - loss: 0.3828 - accuracy: 0.6422

Epoch 71: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 986ms/step - loss: 0.3828 - accuracy: 0.6423 - val_loss: 0.4099 - val_accuracy: 0.6612

Epoch 72/100

23/23 [==============================] - ETA: 0s - loss: 0.3825 - accuracy: 0.6460

Epoch 72: val_loss did not improve from 0.38016

23/23 [==============================] - 24s 1s/step - loss: 0.3831 - accuracy: 0.6449 - val_loss: 0.5160 - val_accuracy: 0.6117

Epoch 73/100

23/23 [==============================] - ETA: 0s - loss: 0.3795 - accuracy: 0.6451

Epoch 73: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 1s/step - loss: 0.3797 - accuracy: 0.6448 - val_loss: 0.4963 - val_accuracy: 0.6231

Epoch 74/100

23/23 [==============================] - ETA: 0s - loss: 0.3769 - accuracy: 0.6479

Epoch 74: val_loss did not improve from 0.38016

23/23 [==============================] - 22s 975ms/step - loss: 0.3783 - accuracy: 0.6459 - val_loss: 0.4888 - val_accuracy: 0.6084

Epoch 75/100

23/23 [==============================] - - -

- -

-

23/23 [==============================] - ETA: 0s - loss: 0.3719 - accuracy: 0.6541

Epoch 75: val_loss did not improve from 0.38016

23/23 [==============================] - 22s 971ms/step - loss: 0.3724 - accuracy: 0.6538 - val_loss: 0.5175 - val_accuracy: 0.6032

Epoch 76/100

23/23 [==============================] - ETA: 0s - loss: 0.3697 - accuracy: 0.6555

Epoch 76: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 1s/step - loss: 0.3687 accuracy: 0.6548 - val_loss: 0.4598 - val_accuracy: 0.6059

Epoch 77/100

23/23 [==============================] - ETA: 0s loss: 0.3702 accuracy: 0.6552

Epoch 77: val_loss did not improve from 0.38016

22s 954ms/step loss: 0.3713

accuracy: 0.6540 val_loss: 0.5650 val_accuracy: 0.5824

Epoch 78/100

23/23 [==============================] - ETA: 0s - loss: 0.3685 - accuracy: 0.6548

Epoch 78: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 982ms/step - loss: 0.3675

accuracy: 0.6557 - val_loss: 0.4115 - val_accuracy: 0.6292

Epoch 79/100

23/23 [==============================] - ETA: 0s - loss: 0.3659 - accuracy: 0.6584

Epoch 79: val_loss did not improve from 0.38016

23/23 [==============================] - 22s 970ms/step - loss: 0.3662 -

accuracy: 0.6577 - val_loss: 0.3868 - val_accuracy: 0.6516

Epoch 80/100

23/23 [==============================] - ETA: 0s - loss: 0.3626 - accuracy: 0.6628

- -

23/23 [==============================] - - -

- -

-

Epoch 80: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 994ms/step - loss: 0.3627 - accuracy: 0.6638 - val_loss: 0.4733 - val_accuracy: 0.6388

Epoch 81/100

23/23 [==============================] - ETA: 0s - loss: 0.3623 - accuracy: 0.6578

Epoch 81: val_loss did not improve from 0.38016

23/23 [==============================] - 22s 970ms/step - loss: 0.3621 - accuracy: 0.6577 - val_loss: 0.5189 - val_accuracy: 0.5979

Epoch 82/100

23/23 [==============================] - ETA: 0s - loss: 0.3603 - accuracy: 0.6612

Epoch 82: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 982ms/step - loss: 0.3600 - accuracy: 0.6614 - val_loss: 0.4210 - val_accuracy: 0.6280

Epoch 83/100

23/23 [==============================] - ETA: 0s - loss: 0.3608 - accuracy: 0.6604

Epoch 83: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 1s/step - loss: 0.3627 - accuracy: 0.6592 - val_loss: 0.5621 - val_accuracy: 0.6082

Epoch 84/100

23/23 [==============================] - ETA: 0s - loss: 0.3605 - accuracy: 0.6640

Epoch 84: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 998ms/step - loss: 0.3628 - accuracy: 0.6634 - val_loss: 0.4241 - val_accuracy: 0.6462

Epoch 85/100

23/23 [==============================] - ETA: 0s - loss: 0.3498 - accuracy: 0.6713

Epoch 85: val_loss did not improve from 0.38016

- -

23/23 [==============================] - - -
- -

-

23/23 [==============================] - 23s 976ms/step - loss: 0.3484 - accuracy: 0.6713 - val_loss: 0.4425 - val_accuracy: 0.6489

Epoch 86/100

23/23 [==============================] - ETA: 0s loss: 0.3537 accuracy: 0.6663

Epoch 86: val_loss did not improve from 0.38016

- -

23/23 [==============================] -

- -

-

23s 1s/step - loss: 0.3543 -

accuracy: 0.6656 val_loss: 0.4006 val_accuracy: 0.6716

Epoch 87/100

23/23 [==============================] - ETA: 0s - loss: 0.3503 - accuracy: 0.6698

Epoch 87: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 987ms/step - loss: 0.3493
accuracy: 0.6697 - val_loss: 0.4375 - val_accuracy: 0.6527

Epoch 88/100

23/23 [==============================] - ETA: 0s - loss: 0.3497 - accuracy: 0.6714

Epoch 88: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 986ms/step - loss: 0.3495 -
accuracy: 0.6710 - val_loss: 0.5339 - val_accuracy: 0.6160

Epoch 89/100

23/23 [==============================] - ETA: 0s - loss: 0.3500 - accuracy: 0.6671

Epoch 89: val_loss did not improve from 0.38016

23/23 [==============================] - 22s 970ms/step - loss: 0.3501 -
accuracy: 0.6666 - val_loss: 0.4148 - val_accuracy: 0.6438

Epoch 90/100

23/23 [==============================] - ETA: 0s - loss: 0.3494 - accuracy: 0.6661

Epoch 90: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 995ms/step - loss: 0.3529 -
accuracy: 0.6647 - val_loss: 0.4992 - val_accuracy: 0.6324

Epoch 91/100

23/23 [==============================] - ETA: 0s - loss: 0.3479 - accuracy: 0.6718

Epoch 91: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 986ms/step - loss: 0.3482 -
accuracy: 0.6715 - val_loss: 0.6037 - val_accuracy: 0.6195

- -

23/23 [==============================] - - -
- -
-

Epoch 92/100

23/23 [==============================] - ETA: 0s - loss: 0.3436 - accuracy: 0.6767

Epoch 92: val_loss did not improve from 0.38016

23/23 [==============================] - 22s 964ms/step - loss: 0.3452 - accuracy: 0.6764 - val_loss: 0.4368 - val_accuracy: 0.6462

Epoch 93/100

23/23 [==============================] - ETA: 0s - loss: 0.3377 - accuracy: 0.6793

Epoch 93: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 984ms/step - loss: 0.3372 - accuracy: 0.6795 - val_loss: 0.5267 - val_accuracy: 0.6275

Epoch 94/100

23/23 [==============================] - ETA: 0s - loss: 0.3433 - accuracy: 0.6743

Epoch 94: val_loss did not improve from 0.38016

23/23 [==============================] - 22s 964ms/step - loss: 0.3453 -
accuracy: 0.6736 - val_loss: 0.4532 - val_accuracy: 0.6314

Epoch 95/100

23/23 [==============================] - ETA: 0s loss: 0.3409 accuracy: 0.6780

Epoch 95: val_loss did not improve from 0.38016

23s 987ms/step loss: 0.3407
accuracy: 0.6775 val_loss: 0.4901 val_accuracy: 0.6680

Epoch 96/100

23/23 [==============================] - ETA: 0s - loss: 0.3378 - accuracy: 0.6791

Epoch 96: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 991ms/step - loss: 0.3388
accuracy: 0.6793 - val_loss: 0.5620 - val_accuracy: 0.6063

Epoch 97/100

23/23 [==============================] - ETA: 0s - loss: 0.3389 - accuracy: 0.6763

23/23 [==============================] - - -
- -

-

Epoch 97: val_loss improved from 0.38016 to 0.33265, saving model to ckpt 23/23 [==============================] - 53s 2s/step - loss: 0.3402 -

accuracy: 0.6765 - val_loss: 0.3327 - val_accuracy: 0.6854

Epoch 98/100

23/23 [==============================] - ETA: 0s - loss: 0.3408 - accuracy: 0.6768

Epoch 98: val_loss did not improve from 0.33265

23/23 [==============================] - 22s 974ms/step - loss: 0.3407 -

accuracy: 0.6766 - val_loss: 0.4046 - val_accuracy: 0.6695

Epoch 99/100

23/23 [==============================] - ETA: 0s - loss: 0.3388 - accuracy: 0.6795

Epoch 99: val_loss did not improve from 0.33265

23/23 [==============================] - 23s 985ms/step - loss: 0.3394 -

accuracy: 0.6791 - val_loss: 0.4475 - val_accuracy: 0.6622

Epoch 100/100

23/23 [==============================] - ETA: 0s - loss: 0.3358 - accuracy: 0.6787

Epoch 100: val_loss did not improve from 0.33265

23/23 [==============================] - 22s 968ms/step - loss: 0.3385 - accuracy: 0.6773 - val_loss: 0.3742 - val_accuracy: 0.6796

8.Visualize Metrics

Visualization of Training Metrics for Loss and Accuracy:

The following code is used to create a visual representation of training metrics for a machine learning model. Specifically, it visualizes two important metrics, namely "Loss" and "Accuracy," over the course of training, typically for a neural network.

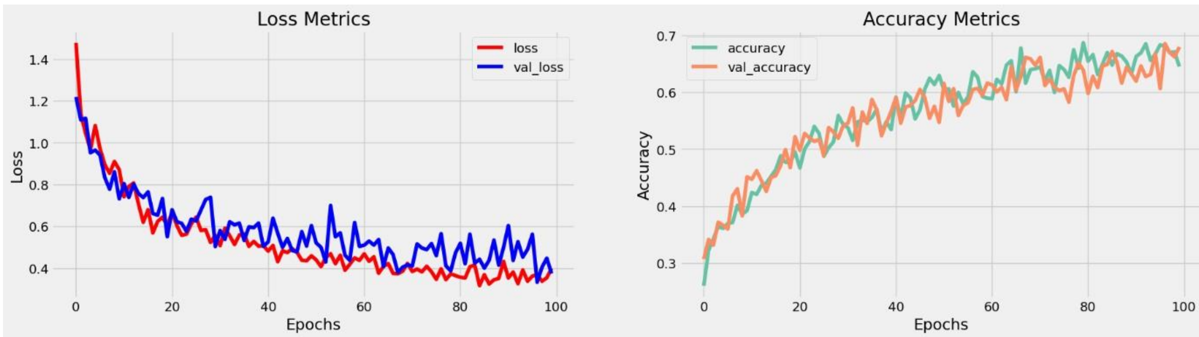
In [16]:

```
fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))
ax[0].plot(history.history['loss'],label='loss',c='red')
```

23/23 [==============================] - - -
- -

-

```
ax[0].plot(history.history['val_loss'],label='val_loss',c = 'blue') ax[0].set_xlabel('Epochs')
ax[1].set_xlabel('Epochs') ax[0].set_ylabel('Loss') ax[1].set_ylabel('Accuracy') ax[0].set_title('Loss
Metrics') ax[1].set_title('Accuracy Metrics')
ax[1].plot(history.history['accuracy'],label='accuracy')
ax[1].plot(history.history['val_accuracy'],label='val_accuracy') ax[0].legend() ax[1].legend()
plt.show()
```



9.Save Model

The below code relates to saving and examining the layers of a machine learning model,presumably a neural network.

In [17]:

```
model.load_weights('ckpt') model.save('models',save_format='tf')
```

In [18]:

```
for idx,i in enumerate(model.layers):
   print('Encoder layers:' if idx==0 else 'Decoder layers: ')     for j in i.layers:
      print(j)     print('---------------------') Encoder layers:
<keras.layers.core.embedding.Embedding object at 0x782084b9d190>
<keras.layers.normalization.layer_normalization.LayerNormalization object at 0x7820e56f1b90>

<keras.layers.rnn.lstm.LSTM object at 0x7820841bd650> ---------------------
Decoder layers:

<keras.layers.core.embedding.Embedding object at 0x78207c258590>
<keras.layers.normalization.layer_normalization.LayerNormalization object at 0x78207c78bd10>

<keras.layers.rnn.lstm.LSTM object at 0x78207c258a10>
```

23/23 [==============================] - - -
- -

-

<keras.layers.core.dense.Dense object at 0x78207c2636d0>

10.Create Inference Model

The code given below defines a custom chatbot model using TensorFlow/Keras, and it includes visualization of the model architecture using `tf.keras.utils.plot_` `ln[19]:`

```
class ChatBot(tf.keras.models.Model):     def
__init__(self,base_encoder,base_decoder,*args,**kwargs):
      super().__init__(*args,**kwargs)
self.encoder,self.decoder=self.build_inference_model(base_encoder,bas e_decoder)

   def build_inference_model(self,base_encoder,base_decoder):
      encoder_inputs=tf.keras.Input(shape=(None,))        x=base_encoder.layers[0](encoder_inputs)
x=base_encoder.layers[1](x)
      x,encoder_state_h,encoder_state_c=base_encoder.layers[2](x)
encoder=tf.keras.models.Model(inputs=encoder_inputs,outputs=[encoder_
state_h,encoder_state_c],name='chatbot_encoder')

      decoder_input_state_h=tf.keras.Input(shape=(lstm_cells,))
```

23/23 [==============================] - - -
- -

-

```
 decoder_input_state_c=tf.keras.Input(shape=(lstm_cells,))
decoder_inputs=tf.keras.Input(shape=(None,))        x=base_decoder.layers[0](decoder_inputs)
x=base_encoder.layers[1](x)        x,decoder_state_h,decoder_state_c=base_decoder.layers[2](x,initial_st
ate=[decoder_input_state_h,decoder_input_state_c])        decoder_outputs=base_decoder.layers[-1](x)
decoder=tf.keras.models.Model(
inputs=[decoder_inputs,[decoder_input_state_h,decoder_input_state
_c]],          outputs=[decoder_outputs,[decoder_state_h,decoder_state_c]],name= 'chatbot_decoder'        )
        return encoder,decoder

    def summary(self):
        self.encoder.summary()        self.decoder.summary()

    def softmax(self,z):
        return np.exp(z)/sum(np.exp(z))

    def sample(self,conditional_probability,temperature=0.5):
        conditional_probability =
np.asarray(conditional_probability).astype("float64")        conditional_probability =
np.log(conditional_probability) / temperature
        reweighted_conditional_probability = self.softmax(conditional_probability)        probas =
np.random.multinomial(1, reweighted_conditional_probability,
1)
        return np.argmax(probas)

    def preprocess(self,text):        text=clean_text(text)
seq=np.zeros((1,max_sequence_length),dtype=np.int32)        for i,word in
enumerate(text.split()):
            seq[:,i]=sequences2ids(word).numpy()[0]        return seq
         def postprocess(self,text):
        text=re.sub(' - ','-',text.lower())        text=re.sub(' [.] ','. ',text)
text=re.sub(' [1] ','1',text)        text=re.sub(' [2] ','2',text)
text=re.sub(' [3] ','3',text)        text=re.sub(' [4] ','4',text)
text=re.sub(' [5] ','5',text)        text=re.sub(' [6] ','6',text)
text=re.sub(' [7] ','7',text)        text=re.sub(' [8] ','8',text)
text=re.sub(' [9] ','9',text)        text=re.sub(' [0] ','0',text)
text=re.sub(' [,] ',', ',text)        text=re.sub(' [?] ','? ',text)
text=re.sub(' [!] ','! ',text)        text=re.sub(' [$] ','$ ',text)
text=re.sub(' [&] ','& ',text)        text=re.sub(' [/] ','/ ',text)
text=re.sub(' [:] ',': ',text)        text=re.sub(' [;] ','; ',text)
text=re.sub(' [*] ','* ',text)        text=re.sub(' [\' ] ','\' ',text)
text=re.sub(' [\"] ','\"',text)        return text

    def call(self,text,config=None):        input_seq=self.preprocess(text)
states=self.encoder(input_seq,training=False)        target_seq=np.zeros((1,1))
target_seq[:,:]=sequences2ids(['<start>']).numpy()[0][0]        stop_condition=False
decoded=[]        while not stop_condition:
```

23/23 [==============================] - - -
- -

-

decoder_outputs,new_states=self.decoder([target_seq,states],train ing=False)
# index=tf.argmax(decoder_outputs[:,-1,:],axis=-1).numpy().item() index=self.sample(decoder_outputs[0,0,:]).item()
word=ids2sequences([index]) if word=='<end> ' or len(decoded)>=max_sequence_length:
stop_condition=True else:
decoded.append(index) target_seq=np.zeros((1,1))
target_seq[:,:]=index states=new_states return
self.postprocess(ids2sequences(decoded))

chatbot=ChatBot(model.encoder,model.decoder,name='chatbot')
chatbot.summary() Model: "chatbot_encoder"

____

Layer (type) Output Shape Param #

=== input_1
(InputLayer) [(None, None)] 0
encoder_embedding (Embeddin (None, None, 256) 625408 g)
layer_normalization (LayerN (None, None, 256) 512 ormalization)
encoder_lstm (LSTM) [(None, None, 256), 525312 (None,
256),

(None, 256)]

===

Total params: 1,151,232

Trainable params: 1,151,232

Non-trainable params: 0

____

Model: "chatbot_decoder"

____

Layer (type) Output Shape Param # Connected to

==
======================== input_4 (InputLayer)
[(None,
None)] 0 []

decoder_embedding (Embedding) (None,
None, 256) 625408 ['input_4[0][0]']

23/23 [==============================] - - -

- -

-

layer_normalization (LayerNorm (None,
None, 256) 512 ['decoder_embedding[0][0]']

alization)

input_2 (InputLayer) [(None,
256)] 0 []

input_3 (InputLayer) [(None,
256)] 0 []

decoder_lstm (LSTM) [(None, None, 256),
525312 ['layer_normalization[1][0]',

(None,
256), 'input_2[0][0]',

(None,
256)] 'input_3[0][0]']

l

decoder_dense (Dense) (None, None,
2443) 627851 ['decoder_lstm[0][0]']

==
========================

Total params: 1,779,083 Trainable
params: 1,779,083

Non-trainable params: 0

____
____

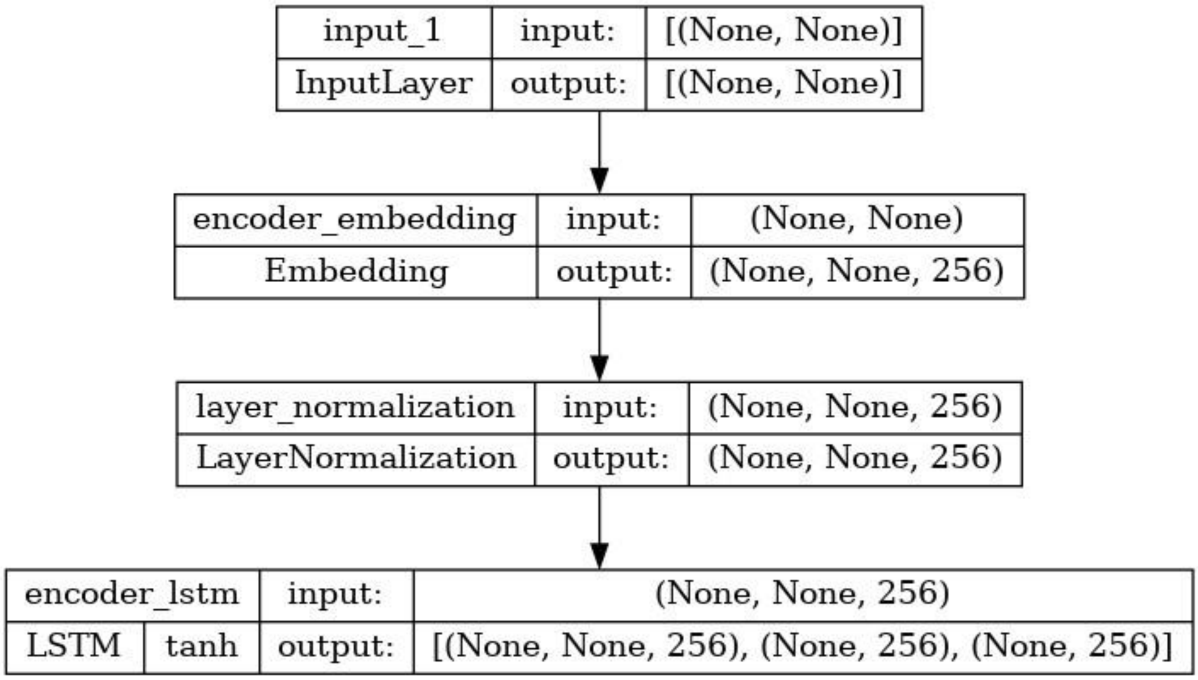
In [20]:

```
tf.keras.utils.plot_model(chatbot.encoder,to_file='encoder.png',show_shapes=T
rue,show_layer_activations=True)
```

Out[20]:

23/23 [==============================] - - -
- -

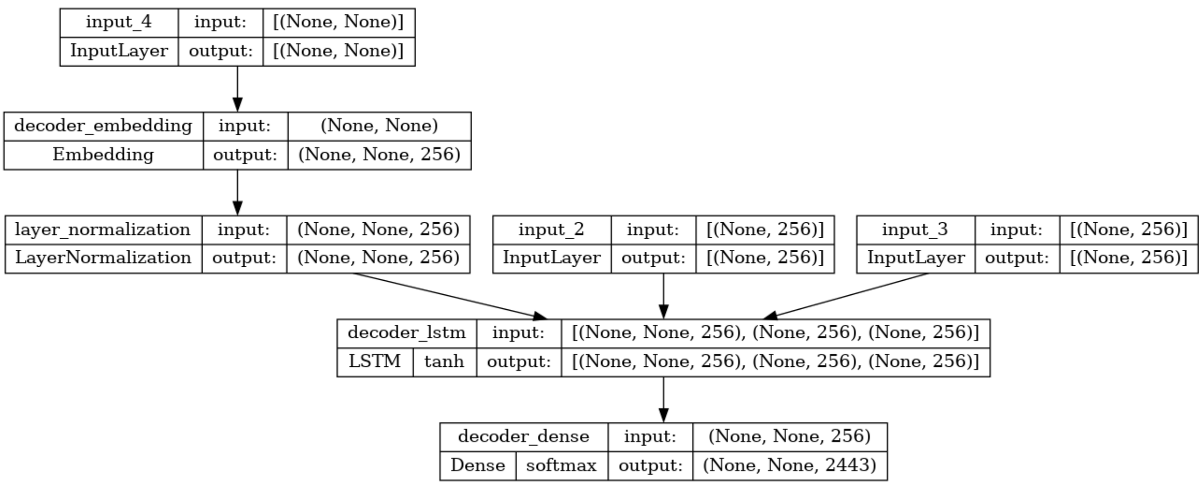
-



In [21]:

```
tf.keras.utils.plot_model(chatbot.decoder,to_file='decoder.png',show_shapes=T
rue,show_layer_activations=True)
```

Out[21]:



11.Time to Chat

"Time to chat" means the moment when the bot is ready to engage in a conversation with the user.

In [22]:

```
def print_conversation(texts):     for text in
texts:
```

23/23 [==============================] - - - - -

-

print(f'You: {text}') print(f'Bot:
{chatbot(text)}')
print('========================')

In [23]: print_conversation([

'hi',
'do yo know me?',
'what is your name?',
'you are bot?',
'hi, how are you doing?',
"i'm pretty good. thanks for asking.",
"Don't ever be in a hurry",
'''I'm gonna put some dirt in your eye ''',
'''You're trash ''',
'''I've read all your research on nano-technology ''',
'''You want forgiveness? Get religion''',
'''While you're using the bathroom, i'll order some food.''',
'''Wow! that's terrible.''',
'''We'll be here forever.''',
'''I need something that's reliable.''',
'''A speeding car ran a red light, killing the girl.''',
'''Tomorrow we'll have rice and fish for lunch.''',
'''I like this restaurant because they give you free bread.'''])

You: hi

Bot: i have to go to the bathroom.
======================== You: do you
know me?

Bot: yes, it's too close to the other.
======================== You: what is your
name?

Bot: i have to walk the house.
======================== You: you
are bot?

Bot: no, i have. all my life. ========================

You: hi, how are you doing?

Bot: i'm going to be a teacher.

========================

You: i'm pretty good. thanks for asking.

Bot: no problem. i'll have to give you the english assignments from my mind.

23/23 [==============================] - - -
- -

-

========================

You: Don't ever be in a hurry Bot: it's not a
great. ========================

You: I'm gonna put some dirt in your eye Bot: that's a good
idea.

========================

You: You're trash

Bot: the tv news is reporting a bank robbery.

========================

You: I've read all your research on nano-technology Bot: it's the weather.
i've gone around the world.

========================

You: You want forgiveness? Get religion Bot: no, i'll be my.

========================

You: While you're using the bathroom, i'll order some food. Bot: don't order for me.
i've been a cheater. ========================

You: Wow! that's terrible.

Bot: never park your car under the house. ========================

You: We'll be here forever.

Bot: we'll be there in half an hour.

========================

You: I need something that's reliable.

Bot: you need a car with low mileage.

========================

You: A speeding car ran a red light, killing the girl.

Bot: what happened?

========================

You: Tomorrow we'll have rice and fish for lunch.

Bot: i'll make a sandwich.

========================

23/23 [==============================] - - -
- -

-

You: I like this restaurant because they give you free bread. Bot: well, i think that's a good idea.

Feature engineering:

Feature engineering is a crucial step in building a Python chatbot, as it involves transforming and selecting relevant data from the raw input to create meaningful features for the model to learn from. Here are the key feature engineering steps for a Python chatbot:

1. Text Preprocessing:

Tokenization:Break the input text into individual words or tokens.

Lowercasing: Convert all text to lowercase to ensure consistency.

Stop Word Removal: Remove common words like "the," "is," and "and" that don't carry much information.

Punctuation Removal: Eliminate punctuation marks and special characters.

Stemming or Lemmatization: Reduce words to their base form

2. *Text Vectorization*:

Bag of Words (BoW):Create a numerical representation of the text data by counting the frequency of each word in the document.

Term Frequency-Inverse Document Frequency (TF-IDF): Weigh words based on their importance within the document and across a corpus.

Word Embeddings (e.g., Word2Vec, GloVe): Create dense vector representations of words that capture semantic meaning.

3. Feature Selection:

Dimensionality Reduction:If the feature space is too large, reduce it using techniques like Principal Component Analysis (PCA) or feature selection methods.

Selecting Relevant Features: Choose the most informative features that contribute to the task at hand. For a chatbot, this may involve identifying keywords, entities, or context words.

4. Contextual Features:

23/23 [==============================] - - - - -

-

Incorporate contextual information, such as the previous user messages, to help the chatbot understand the conversation's flow and user intent.

5. Named Entity Recognition (NER):

Identify and extract entities like names, dates, locations, and other specific information that the chatbot may need to work with.

6. Sentiment Analysis:

Analyze the sentiment of the user's input or generated responses to better understand user emotions and tailor responses accordingly.

7. Intent Classification:

Train a model to classify user intents based on their input. This involves labeling user messages with specific intents (e.g., "greeting," "information request," "goodbye").

8. Multi-Turn Conversation Handling:

Track and store the history of the conversation to enable context-aware responses. This can involve maintaining a conversation history and using it as a contextual feature.

9. User Profiling:

Create user profiles that contain information about users' preferences, previous interactions, and any relevant user-specific data.

10. External Data Integration:

Incorporate external data sources or APIs to enrich the chatbot's knowledge and capabilities. For example, integrating weather data, news feeds, or databases for specific queries.

11. Error Handling Features:

Implement features for handling misunderstandings and errors, such as a "clarify" intent or responses like "I'm sorry, I don't understand."

23/23 [==============================] - - -
- -

-

12. Evaluation Metrics:

Consider including features for tracking and analyzing chatbot performance, such as user satisfaction ratings, response times, and user engagement data.

Remember that the specific feature engineering steps for your Python chatbot will depend on its purpose, the type of chatbot and the available data and resources. Careful feature engineering can significantly improve the chatbot's ability to understand user input and provide meaningful responses.

Conclusion:

In conclusion, Python chatbot development is a dynamic and multifaceted process that combines natural language processing, machine learning, and user experience design. To create a successful chatbot, it's crucial to define clear objectives, choose the right model, focus on user interaction, maintain context, and prioritize privacy and security. Continuous improvement, ethical considerations, and engagement with users are key for building an effective and reliable Python chatbot.