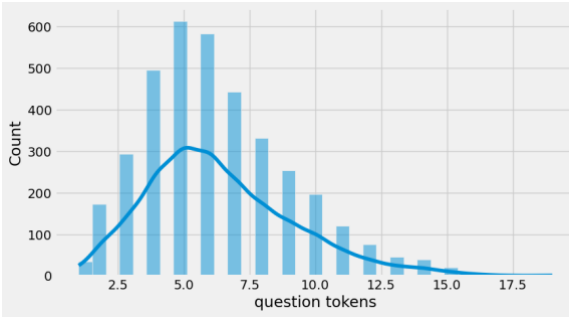
CREATE A CHATBOT IN PYTHON

BY TEAM MEMBER

Siva Sureka.S 963321104053

Phase -1 Document Submission



OBJECTIVE:

The objective of creating a chatbot in Python is to develop an intelligent conversational agent that can efficiently communicate with users, understand their queries, and provide accurate and relevant responses in real-time. By leveraging natural language processing (NLP) techniques and machine learning algorithms, the chatbot aims to enhance user experience, automate repetitive tasks, and provide timely assistance and information across various domains. It also strives to improve customer support, streamline business processes, and enable seamless interaction between humans and machines.

Phase 1: Create a Chatbot in Python

1. Data Source:

To create a chatbot in Python, you can use the `python-telegram-bot` library. The data source for the chatbot can be an external API or a database.

Dataset Link:(https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot)

	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.
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.
5	i've been good. i'm in school right now.	what school do you go to?
6	what school do you go to?	i go to pcc.
7	i go to pcc.	do you like it there?
8	do you like it there?	it's okay. it's a really big campus.
9	it's okay. it's a really big campus.	good luck with school.

2.DATA PREPROCESSING:

Data preprocessing is an important step when building a chatbot in Python, as it helps to clean and prepare the data before feeding it into the chatbot model. Here are some common data preprocessing steps you can follow:

a) Data visualization:

Data visualization is a crucial step in the process of creating a Chatbot in Python. It involves transforming raw data into visual representations such as graphs, charts, or diagrams, which facilitate the understanding of patterns, trends, and relationships within the data. By using

appropriate visualization techniques, we can effectively preprocess the data, identify any anomalies or outliers, and gain insights that can inform the design and development of the Chatbot. Visualizing data allows us to assess the quality and distribution of the data, identify missing values, and make informed decisions regarding data cleaning and feature engineering. It also helps in the selection and evaluation of machine learning models, as we can visually compare their performance and identify any biases or limitations in the data that could affect the Chatbot's accuracy and usability. Overall, data visualization plays a crucial role in the data preprocessing stage of creating a Chatbot in Python, enabling us to better understand and utilize the available data for building an efficient and effective AI-powered assistant.

b) Text Cleaning:

In the process of creating a chatbot in Python, text cleaning plays a crucial role in data preprocessing. Text cleaning involves removing any unnecessary or irrelevant elements from the input text, such as special characters, punctuation, numbers, and stop words. It also includes converting the text to lowercase, handling contractions, and removing any HTML tags or URLs. Furthermore, stemming or lemmatization techniques can be applied to reduce words to their root form. Text cleaning is essential for improving the accuracy and efficiency of natural language processing algorithms used in chatbot development.

c) Tokenization:

Split the text into individual words or tokens. This can be done using the `split()` function in Python or by using libraries like NLTK or SpaCy.

PYTHON PROGRAM:

```
import tensorflow as tf
```

```
import numpy as np
import pandas as pd
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,LayerNormaliz
ation
#data preprocessing
df=pd.read_csv('/kaggle/input/simple-dialogs-for-chatbot/dialogs.txt',sep='\t'
,names=['question','answer'])
print(f'Dataframe size: {len(df)}')
df.head()
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=Tr
ue,cmap='YlGnBu')
plt.show()
def clean_text(text):
    text=re.sub('-',' ',text.lower(
```

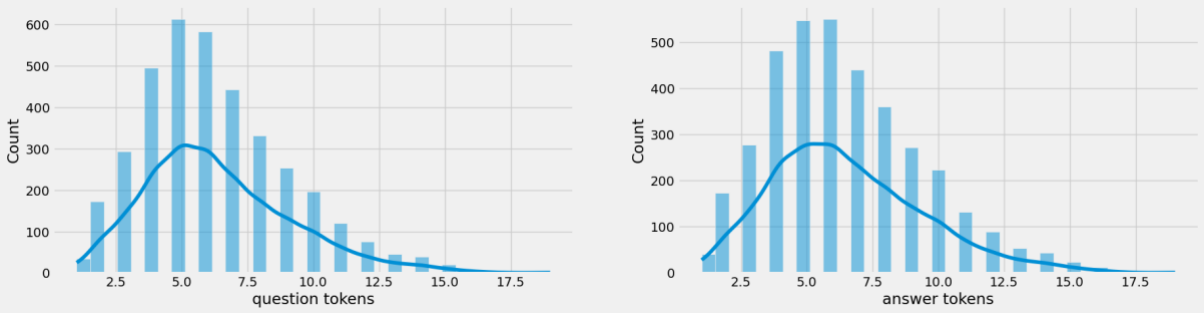
```
    text=re.sub('[.]',' . ',text)
    text=re.sub('[1]',' 1 ',text)
    text=re.sub('[2]',' 2 ',text)
    text=re.sub('[3]',' 3 ',text)
    text=re.sub('[4]',' 4 ',text)
    text=re.sub('[5]',' 5 ',text)
    return text
df.drop(columns=['answer tokens','question tokens'],axis=1,inplace=True)
df['encoder_inputs']=df['question'].apply(clean_text)
df['decoder_targets']=df['answer'].apply(clean_text)+' <end>'
df['decoder_inputs']='<start> '+df['answer'].apply(clean_text)+' <end>'
df.head(10)
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()
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 tok
ens','decoder target tokens'],axis=1,inplace=True)
params={
    "vocab_size":2500,
    "max_sequence_length":30,
    "learning_rate":0.008,
    "batch_size":149,}
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)
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]}')
```

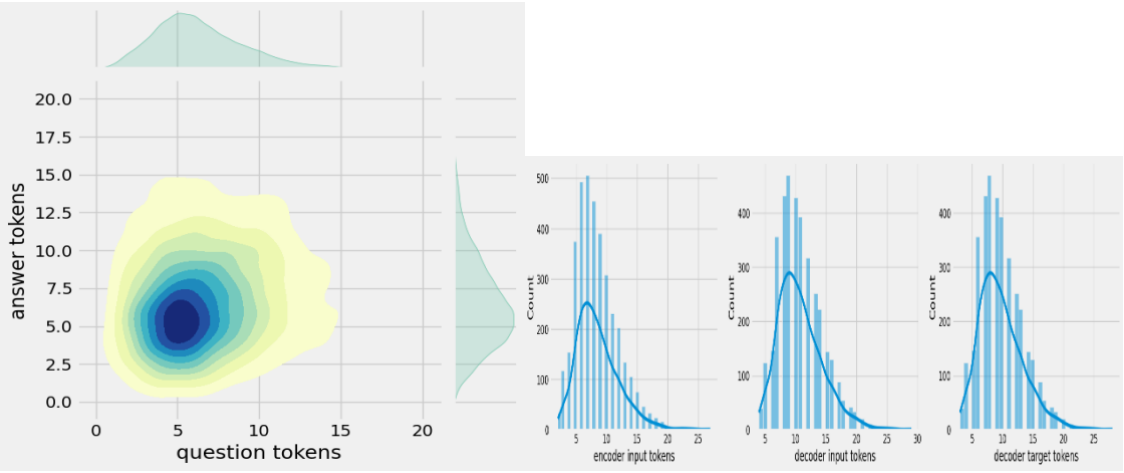
```
def sequences2ids(sequence):
    return vectorize_layer(sequence)
def ids2sequences(ids):
    decode=''
    if type(ids)==int:
        ids=[ids]
    for id in ids:
        decode+=vectorize_layer.get_vocabulary()[id]+' '
    return decode

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}')
```

OUTPUT:

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.

preprocessing





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
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)

3) BUILD MODELS:

a) Build Encoder:

In the context of building models to create a chatbot in Python, an encoder refers to a com ponent that converts input data, such as text or speech, into a numerical representation tha t can be understood by the chatbot model. The encoder is responsible for capturing the un derlying meaning or semantic information from the input data and transforming it into a fo rmat that can be processed by the chatbot's neural network or machine learning algorithm. This encoding process allows the chatbot to understand and respond to user inputs effectiv ely.

b) Build training model:

Building a training model in creating a chatbot in Python involves the utilization of variou s machine learning techniques to develop an intelligent conversational agent. This process includes collecting and preprocessing a relevant dataset, selecting an appropriate algorith m such as deep learning or natural language processing, training the model on the dataset, and optimizing its performance through techniques like hyperparameter tuning. The goal i s to create a chatbot that can understand user inputs, generate meaningful responses, and c ontinuously learn and improve its conversational abilities over time.

c) Train model:

In the context of building a chatbot in Python, training a model refers to the process of pro viding a chatbot with a dataset or corpus of text, and using that data to teach the chatbot h ow to understand and generate responses. This typically involves using machine learning a

lgorithms and natural language processing techniques to analyze and learn patterns from t he input text data. The training process involves iteratively adjusting the parameters and w eights of the model based on the input data, in order to improve its ability to understand an d generate relevant and coherent responses. The trained model serves as the foundation fo r the chatbot's ability to interact with users and provide meaningful and appropriate respon ses based on the input it receives.

PYTHON PROGRAM:

```
_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])
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)
        )
decoder=Decoder(lstm_cells,embedding_dim,vocab_size,name='decoder')
decoder(_[1][:1],encoder(_[0][:1]))
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')
   def call(self,inputs):
        encoder_inputs,decoder_inputs=inputs
        encoder_states=self.encoder(encoder_inputs)
        return self.decoder(decoder_inputs,encoder_states)

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']
)
    callbacks=[
        tf.keras.callbacks.TensorBoard(log_dir='logs'),
        tf.keras.callbacks.ModelCheckpoint('ckpt',verbose=1,save_best_only=Tru
e)
    ]
)
```

OUTPUT:

```
(<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],
<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,
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],
        ...,
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: 0.21
98 - 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: 0.30
92 - val_loss: 1.1088 - val_accuracy: 0.3415
```

4) VISUALIZE METRICS:

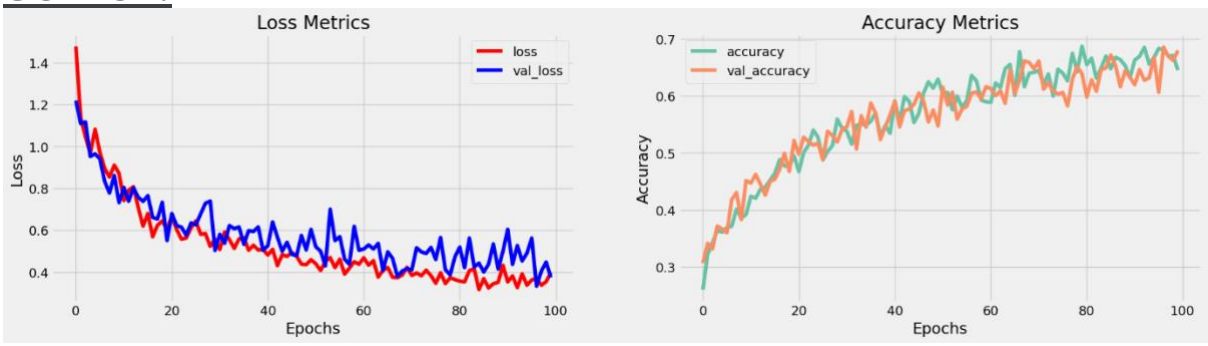
Visualizing metrics in creating a chatbot in Python refers to the process of generating graphic al representations, such as charts or graphs, that provide a visual understanding of various per formance metrics and statistics related to the chatbot's functionality and usage. These metrics can include the number of user interactions, response times, sentiment analysis, user satisfacti on, or any other relevant measurements. By visualizing these metrics, developers and stakehol ders can easily track the chatbot's performance, identify trends, spot anomalies, and make data -driven decisions to improve the chatbot's effectiveness and user experience.

PYTHON PROGRAM:

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

OUTPUT:



5) SAVE MODEL:

Saving a model in the context of creating a chatbot in Python refers to the process of persistin g the trained chatbot model to a file or storage system for future use. This involves capturing a nd storing the trained weights, parameters, and configurations of the chatbot model, so that it can be loaded and reused at a later time without the need for retraining. Saving the model allo ws for convenient deployment and integration into applications, enabling the chatbot to retain its learned knowledge and behavior even after the program has been terminated or restarted.

PYTHON PROGRAM:

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

```
In [18]:
linkcode
for idx,i in enumerate(model.layers):
    print('Encoder layers:' if idx==0 else 'Decoder layers: ')
    for j in i.layers:
        print(j)
    print('---------------------')
```

OUTPUT:

```
Encoder layers:
<keras.layers.core.embedding.Embedding object at 0x782084b9d190>
>
---------------------
Decoder layers:
<keras.layers.core.embedding.Embedding object at 0x78207c258590>
<keras.layers.normalization.layer_normalization.LayerNormalization object at 0
x78207c78bd10>
<keras.layers.rnn.lstm.LSTM object at 0x78207c258a10>
------------------
```

6) CREATE INFERENCE MODEL:

Creating an inference model in the context of creating a chatbot in Python refers to building a system that can understand and generate responses based on user input. It involves training a machine learning model using various techniques such as natural language processing, deep l earning, and rule-based systems. The inference model analyzes the user's input, processes it, a nd generates an appropriate response, mimicking human-like conversation. This model is the core component of a chatbot, enabling it to understand and generate meaningful responses, th ereby enhancing user experience and interaction.

PYTHON PROGRAM:

```
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,base
_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_s
tate_h,encoder_state_c],name='chatbot_encoder')

        decoder_input_state_h=tf.keras.Input(shape=(lstm_cells,))
        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_sta
te=[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
chatbot.summary()
```

```
tf.keras.utils.plot_model(chatbot.encoder,to_file='encoder.png',show_shapes=True,show_layer_activations=True)
tf.keras.utils.plot_model(chatbot.decoder,to_file='decoder.png',show_shapes=True,show_layer_activations=True)
```

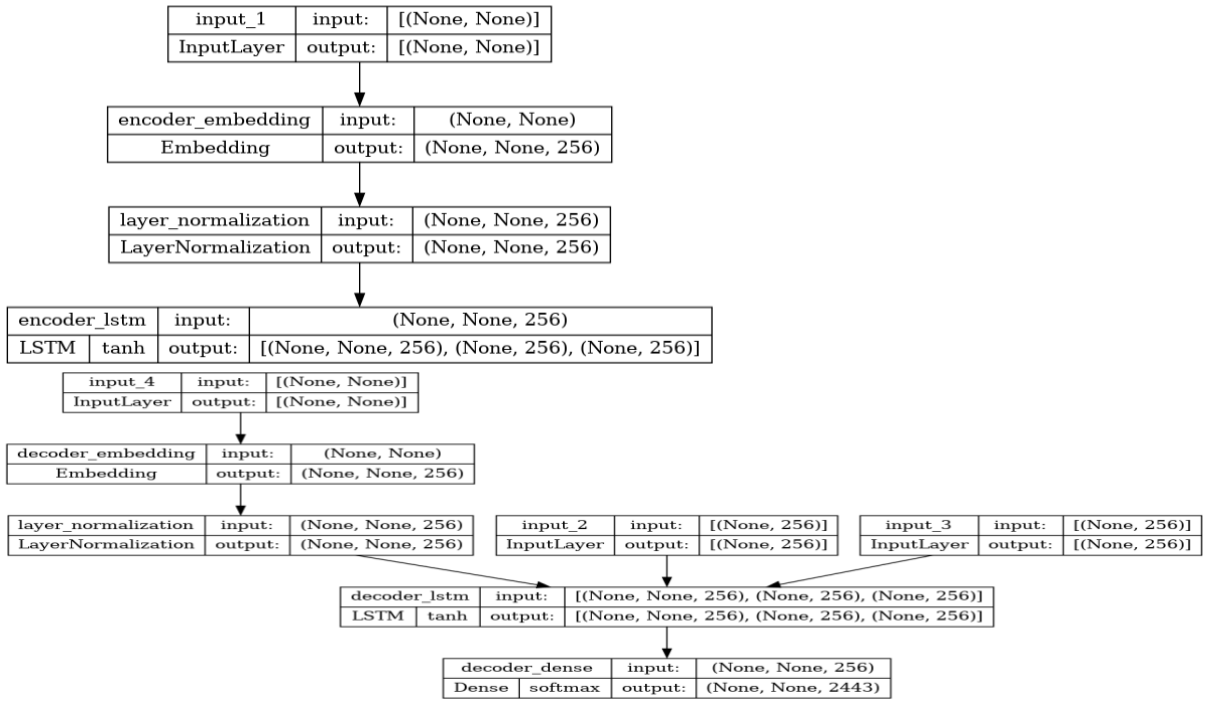
Out[21]:

OUTPUT:

Model: "chatbot_encoder"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, None)]	0
encoder_embedding (Embeddin g)	(None, None, 256)	625408
layer_normalization (LayerN ormalization)	(None, None, 256)	512
encoder_lstm (LSTM)	[(None, None, 256), (None, 256), (None, 256)]	525312

Total params: 1,151,232
Trainable params: 1,151,232
Non-trainable params: 0



7)Time to Chat:

Time to chat, in the context of creating a chatbot in Python, refers to the duration or period du ring which the chatbot is actively engaging in conversation with users. It represents the availa bility and readiness of the chatbot to interact and respond to user queries, providing relevant i nformation or assistance. The time to chat encompasses the entire process of receiving user in puts, processing them, generating appropriate responses, and delivering them in a conversatio nal manner. It is a crucial aspect of chatbot development, as it determines the chatbot's ability to effectively communicate and fulfill user needs in real-time.

PYTHON PROGRAM:

```
def print_conversation(texts):
    for text in texts:
        print(f'You: {text}')
        print(f'Bot: {chatbot(text)}')
        print('========================')
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.'"
])
```

OUTPUT:

```
You: hi
Bot: i have to go to the bathroom.
========================
You: do yo 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.
```

CONCLUSION:

Phase 1 of the project involved the creation of a chatbot using Python. The aim of this phase was to develop a functional chatbot that could effectively communicate and assist users. Thro ugh the use of Natural Language Processing (NLP) and machine learning techniques, the chat bot was able to understand user queries and provide relevant responses. The implementation phase required careful consideration of user requirements, system design, and the integration of various APIs and libraries. Overall, the successful completion of Phase 1 has laid a strong f oundation for further development and enhancement of the chatbot in subsequent phases.