Create a chatbot in python

Team member:

Praveen G,

CSE-III year,

AI101(IBM-Artificial Group 2),

Team Name: The Bug Smashers,

SRG ENGINEERING COLLEGE.

Problem Definition:

The challenge is to create a chatbot in Python that provides exceptional customer service, answering user queries on a website or application. The objective is to deliver high-quality support to users, ensuring a positive user experience and customer satisfaction.

Aim/objective:

In the realm of app and website development, users today expect nothing less than exceptional service when interacting with digital platforms. The challenge at hand lies in addressing user inquiries promptly and effectively, as failure to do so can lead to user disengagement and the potential loss of valuable customers. To safeguard against the erosion of user trust and its adverse impact on the financial bottom line, it becomes imperative to prioritize and consistently deliver the highest quality service throughout the entire lifecycle of website and application development.

It emphasizes the importance of developing web and app solutions that not only meet but exceed user expectations, fostering user retention and sustained organizational success.

Design Techniques:

1) Functionality:

Natural Language Understanding (NLU): Implement advanced NLP techniques for understanding user inputs.

Intent Recognition: Develop algorithms for accurately identifying user intentions.

Entity Extraction: Extract relevant entities from user queries for context-aware responses.

Context Management: Implement mechanisms to maintain conversation context for seamless interactions.

2) User Interface:

Intuitive Design: Prioritize intuitive and user-friendly interface design to enhance usability.

Accessibility: Ensure the app or website is accessible to users with diverse needs.

Responsive Design: Create designs that adapt to different devices and screen sizes.

Voice Interface: Optionally, include voice-based interactions with speech recognition and text-to-speech capabilities.

3) Natural Language Processing (NLP):

Data Preprocessing: Preprocess user inputs by tokenizing, cleaning, and normalizing text.

NLP Model Selection: Choose appropriate NLP models (e.g., spaCy, BERT) for specific tasks.

Intent Classification: Develop models for accurately classifying user intents.

Sentiment Analysis: Implement sentiment analysis to understand user emotions.

4) Responses:

Response Generation: Develop algorithms to generate contextually relevant, coherent responses.

Personalization: Customize responses based on user preferences and historical interactions.

Fallback Responses: Implement fallback responses for unrecognized queries.

Error Handling: Develop mechanisms to gracefully manage errors and maintain conversation flow.

5) Integration:

External APIs: Integrate with external services and APIs to provide real-time data or perform actions.

Multi-Platform Support: Ensure seamless functionality across various platforms and devices.

Security: Implement robust security measures to protect user data, including input validation and encryption.

6) Testing and Improvement for Design Techniques:

Unit Testing: Conduct unit tests to validate individual components, including NLP models and intent recognition.

Integration Testing: Test the entire chatbot system to ensure cohesive operation.

User Testing: Gather user feedback and usability data for iterative design improvements.

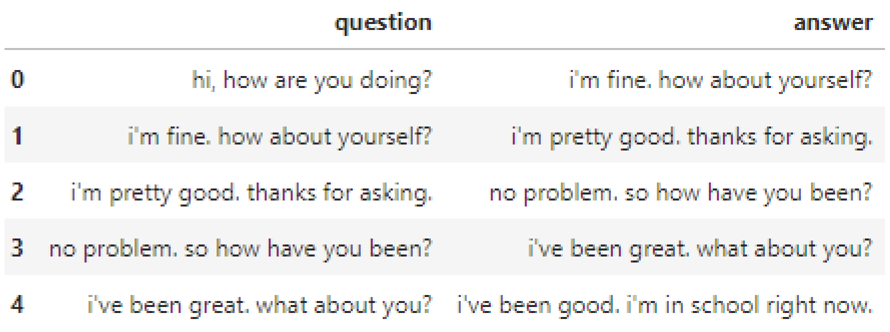
Performance Monitoring: Continuously monitor response times, accuracy, and user satisfaction metrics

Iterative Development: Embrace iterative development cycles to incorporate user feedback and adapt to changing requirements

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,LayerNormalization


 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()
```

Output



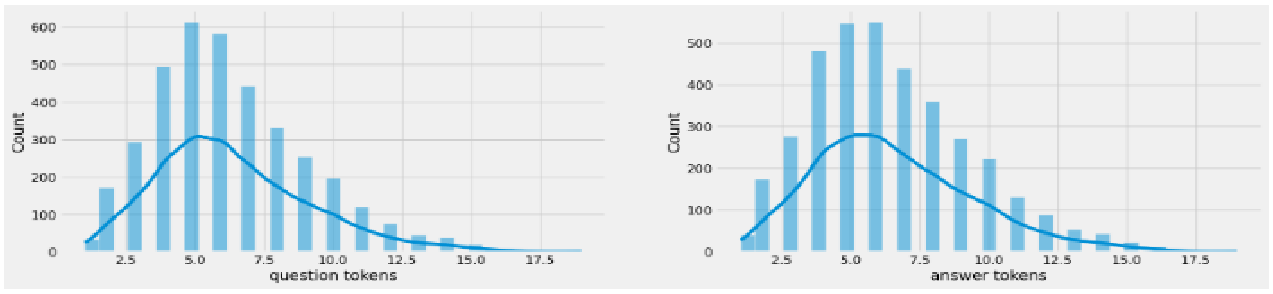
	question	answer
0	hi, how are you doing?	i'm fine. how about yourself?
1	i'm fine. how about yourself?	i'm pretty good. thanks for asking.
2	i'm pretty good. thanks for asking.	no problem. so how have you been?
3	no problem. so how have you been?	i've been great. what about you?
4	i've been great. what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

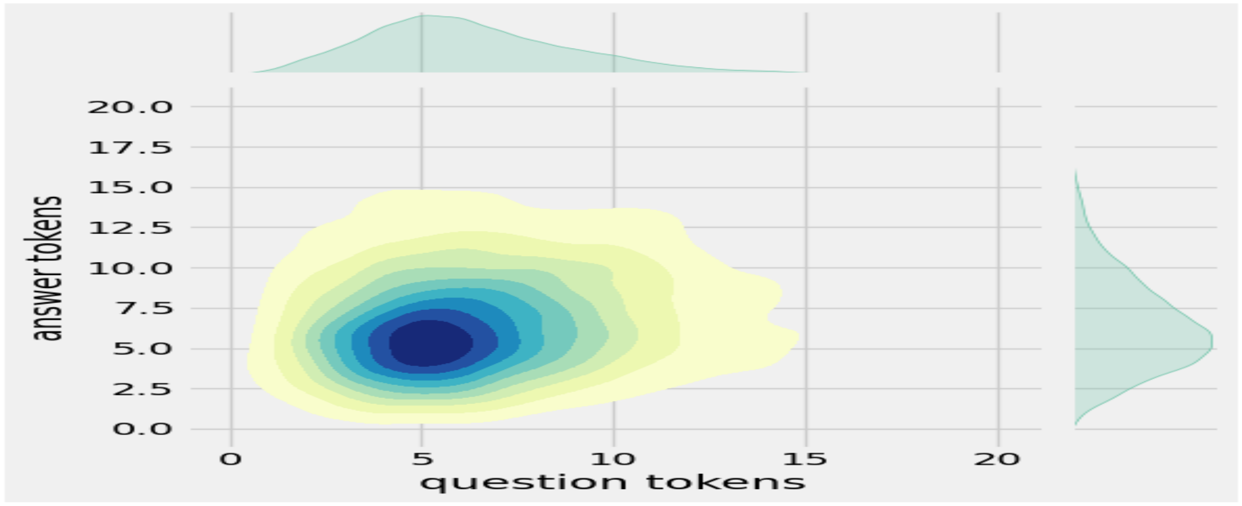
Data Preprocessing

Data Visualization

```
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()
```

Output

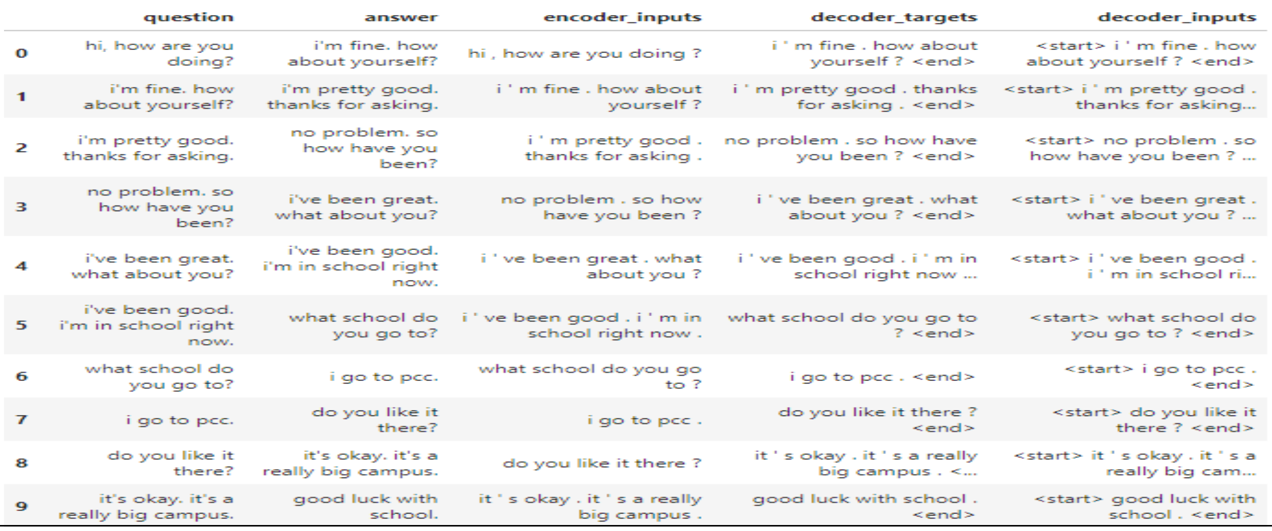




Text Cleaning

```
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
```

Output

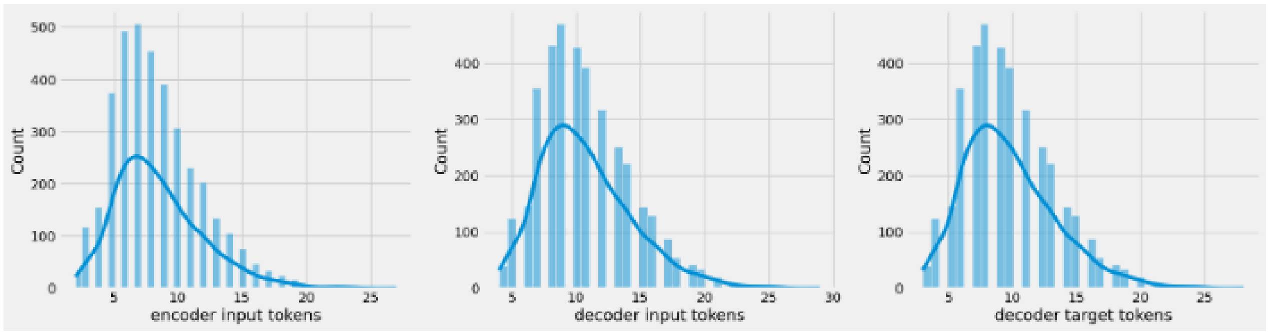


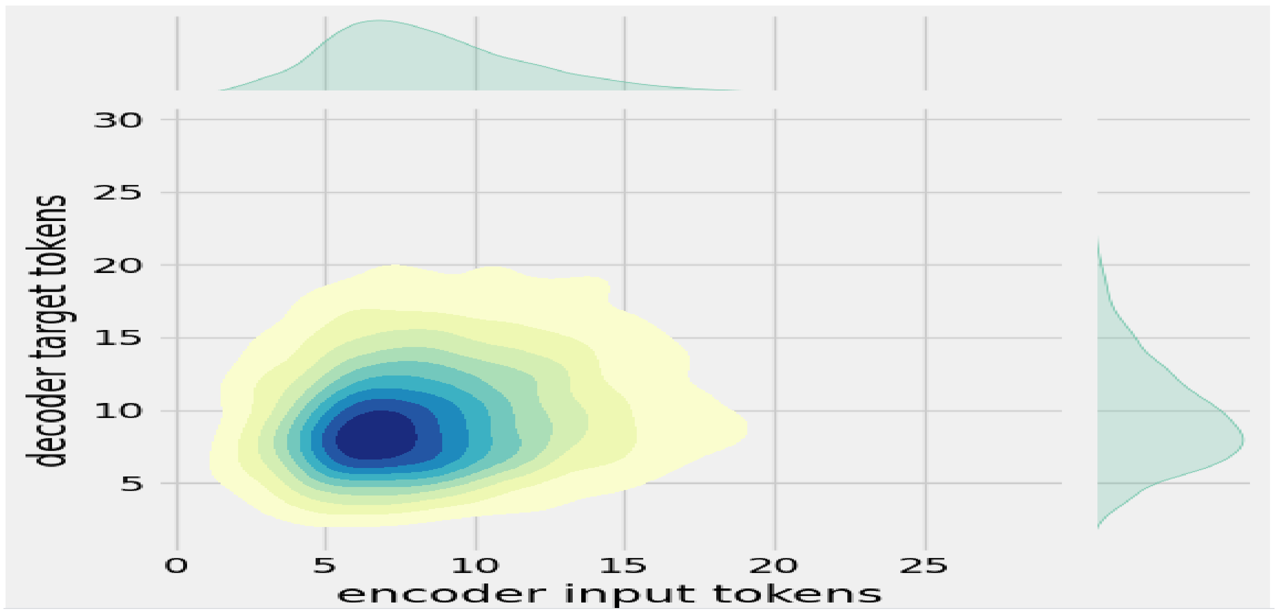
	question	answer	encoder_inputs	decoder_targets	decoder_inputs
0	hi, how are you doing?	i'm fine. how about yourself?	hi , how are you doing ?	i ' m fine . how about yourself ? <end>	<start> i ' m fine . how about yourself ? <end>
1	i'm fine. how about yourself?	i'm pretty good. thanks for asking.	i '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

Input Tokens

```
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()
```

Output

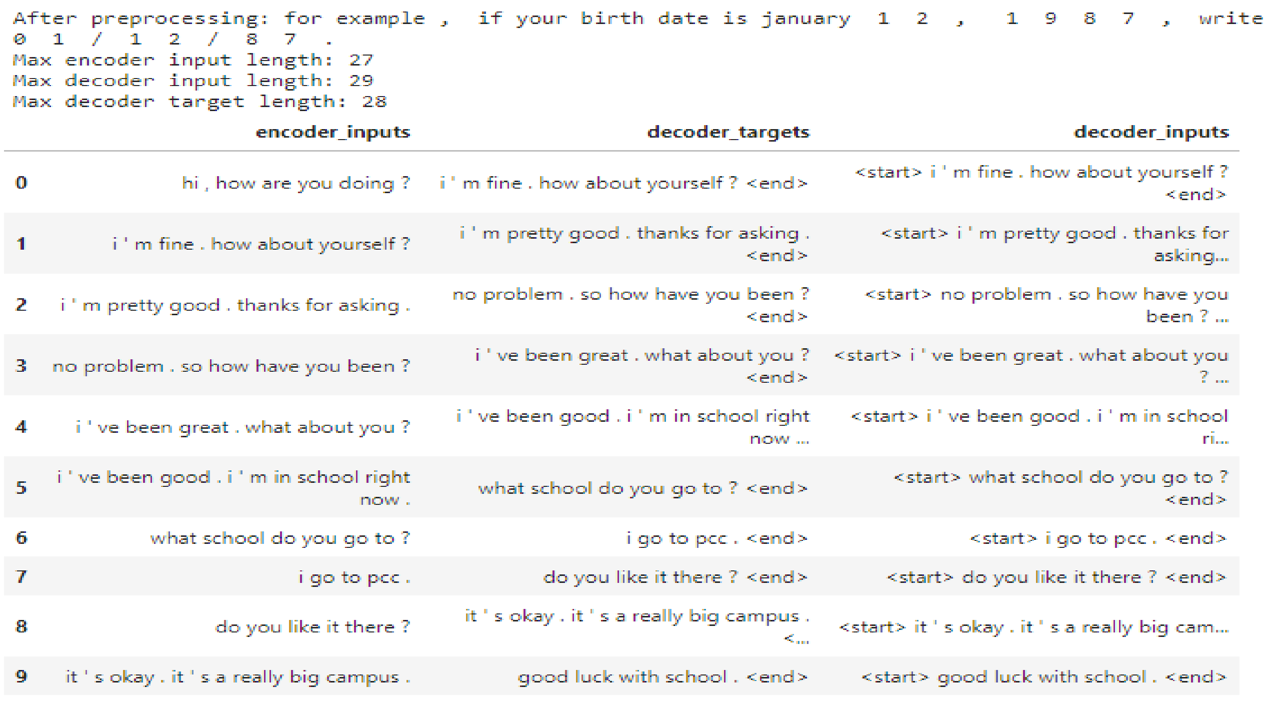




Encoder& Decoder Inputs

```
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)
```

Output



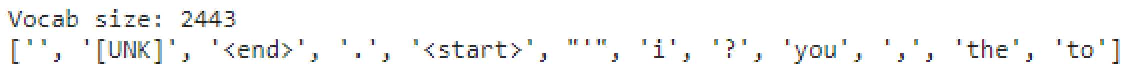
```
After preprocessing: for example ,  if your birth date is january  1  2  ,   1  9  8  7  ,  write
0  1  /  1  2  /  8  7  .
Max encoder input length: 27
Max decoder input length: 29
Max decoder target length: 28
```

	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 ? <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 ...	<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

```
 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]}')
```

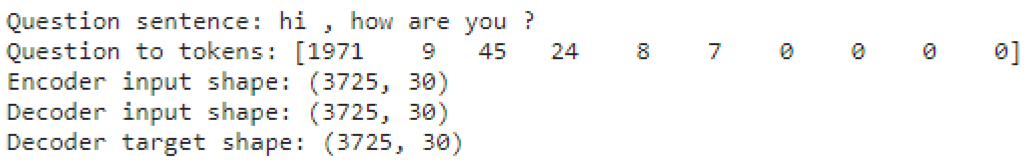
Output



```
Vocab size: 2443
['', '[UNK]', '<end>', '.', '<start>', "'", 'i', '?', 'you', ',', 'the', 'to']
```

```
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}')
```

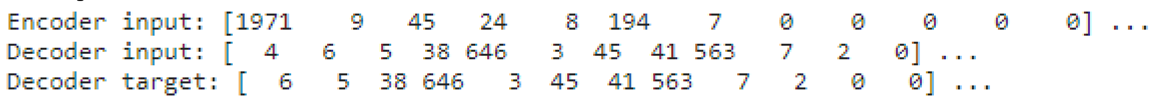
Output



```
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)
```

```
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]} ...')
```

Output



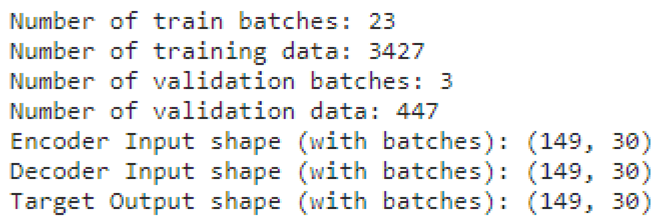
```
Encoder input: [1971    9   45   24    8  194    7    0    0    0    0    0] ...
Decoder input: [   4    6    5   38  646    3   45   41  563    7    2    0] ...
Decoder target: [   6    5   38  646    3   45   41  563    7    2    0    0] ...
```

Dataset Frame

```
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}')
```

Output



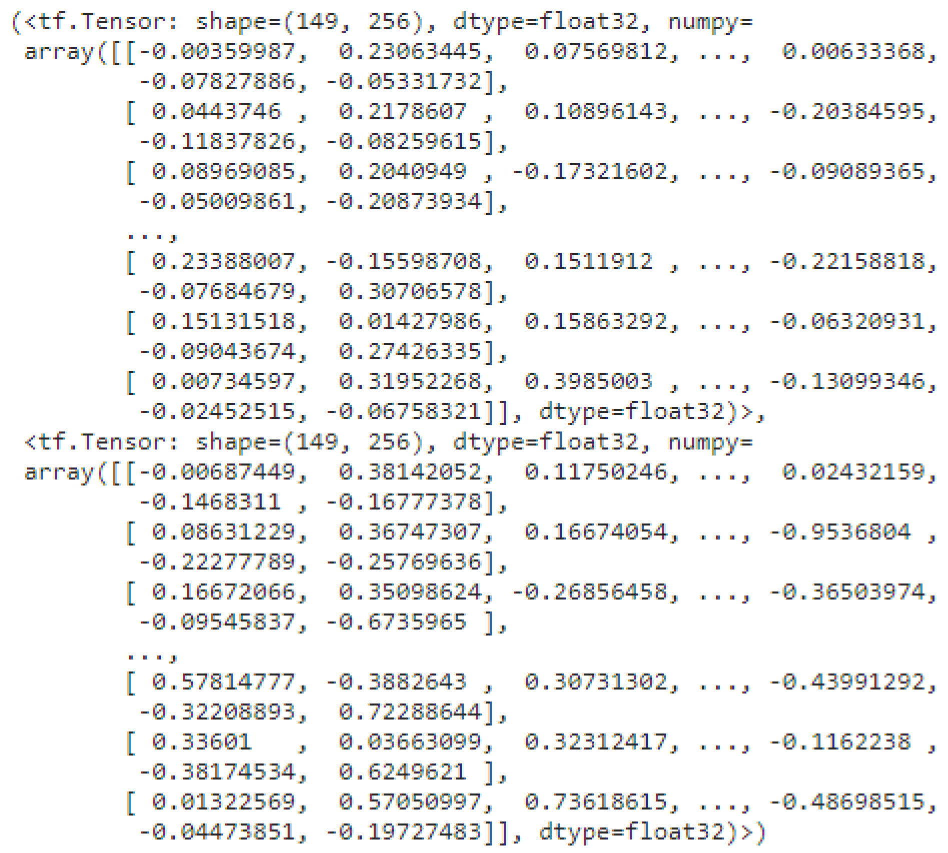
```
Number of train batches: 23
Number of training data: 3427
Number of validation batches: 3
Number of validation data: 447
Encoder Input shape (with batches): (149, 30)
Decoder Input shape (with batches): (149, 30)
Target Output shape (with batches): (149, 30)
```

Build Models

Build Encoder

```
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])
```

Output



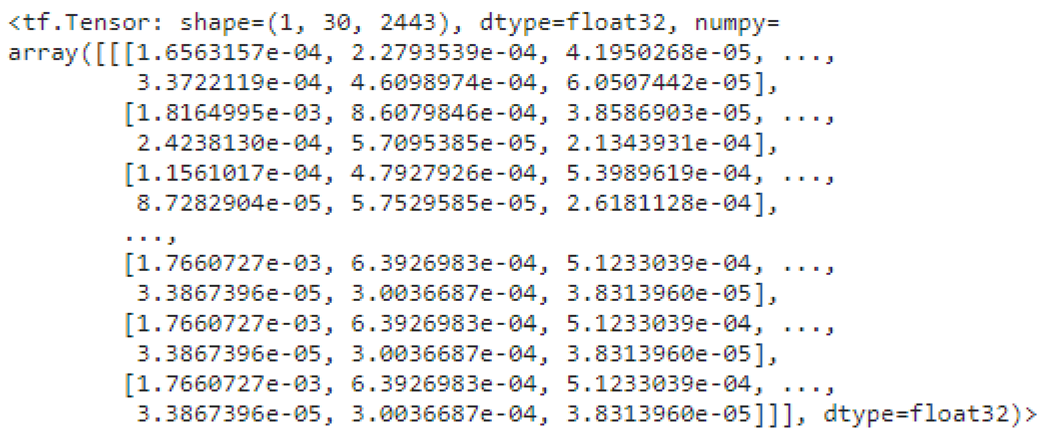
```
(<tf.Tensor: shape=(149, 256), dtype=float32, numpy=
 array([[-0.00359987,  0.23063445,  0.07569812, ...,  0.00633368,
         -0.07827886, -0.05331732],
        [ 0.0443746 ,  0.2178607 ,  0.10896143, ..., -0.20384595,
         -0.11837826, -0.08259615],
        [ 0.08969085,  0.2040949 , -0.17321602, ..., -0.09089365,
         -0.05009861, -0.20873934],
        ...,
        [ 0.23388007, -0.15598708,  0.1511912 , ..., -0.22158818,
         -0.07684679,  0.30706578],
        [ 0.15131518,  0.01427986,  0.15863292, ..., -0.06320931,
         -0.09043674,  0.27426335],
        [ 0.00734597,  0.31952268,  0.3985003 , ..., -0.13099346,
         -0.02452515, -0.06758321]], dtype=float32)>,
 <tf.Tensor: shape=(149, 256), dtype=float32, numpy=
 array([[-0.00687449,  0.38142052,  0.11750246, ...,  0.02432159,
         -0.1468311 , -0.16777378],
        [ 0.08631229,  0.36747307,  0.16674054, ..., -0.9536804 ,
         -0.22277789, -0.25769636],
        [ 0.16672066,  0.35098624, -0.26856458, ..., -0.36503974,
         -0.09545837, -0.6735965 ],
        ...,
        [ 0.57814777, -0.3882643 ,  0.30731302, ..., -0.43991292,
         -0.32208893,  0.72288644],
        [ 0.33601   ,  0.03663099,  0.32312417, ..., -0.1162238 ,
         -0.38174534,  0.6249621 ],
        [ 0.01322569,  0.57050997,  0.73618615, ..., -0.48698515,
         -0.04473851, -0.19727483]], dtype=float32)>)
```

Build Encoder## Build Decoder

```
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_states)
        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]))
```

Output

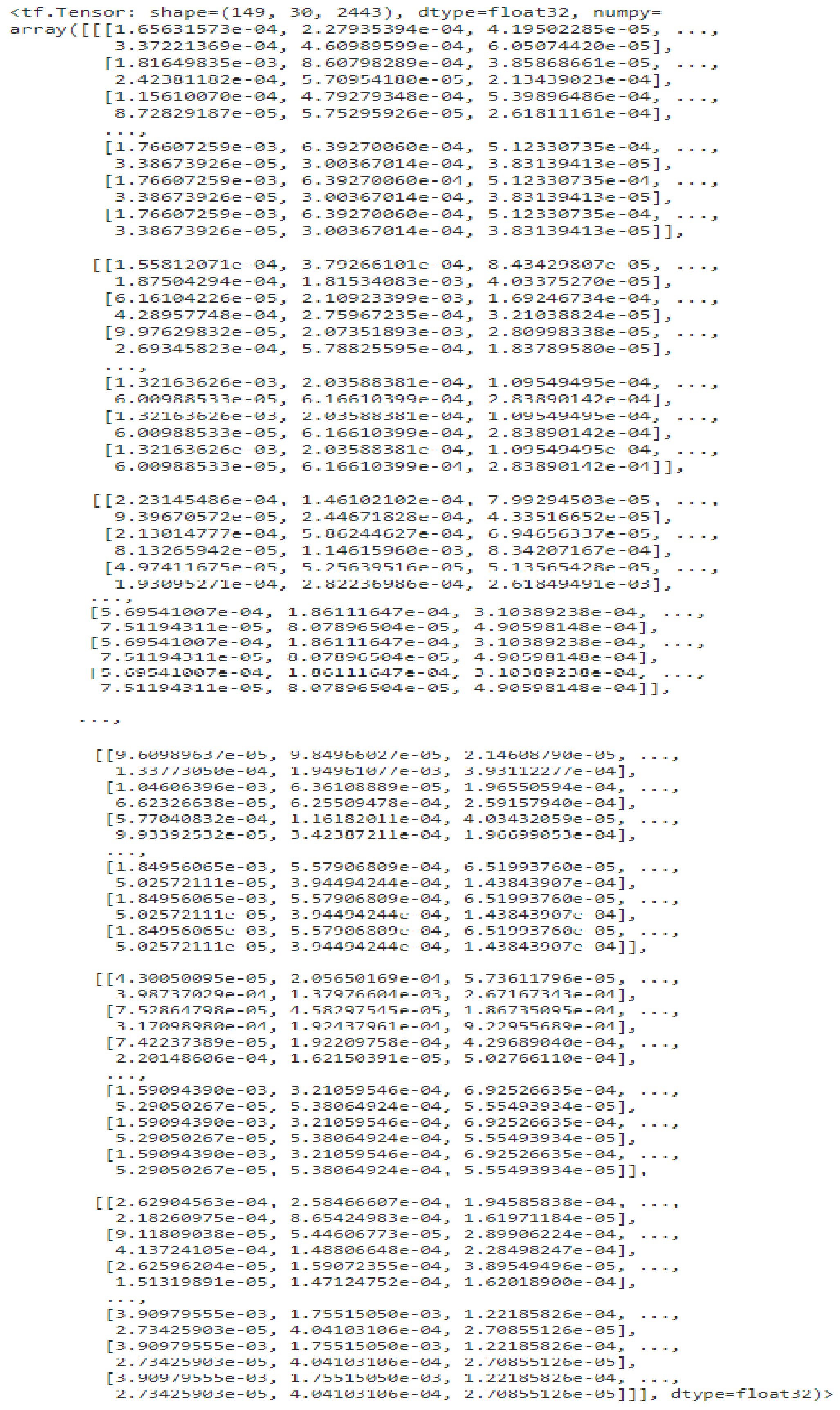


```
<tf.Tensor: shape=(1, 30, 2443), dtype=float32, numpy=
array([[[1.6563157e-04, 2.2793539e-04, 4.1950268e-05, ...,
          3.3722119e-04, 4.6098974e-04, 6.0507442e-05],
        [1.8164995e-03, 8.6079846e-04, 3.8586903e-05, ...,
          2.4238130e-04, 5.7095385e-05, 2.1343931e-04],
        [1.1561017e-04, 4.7927926e-04, 5.3989619e-04, ...,
          8.7282904e-05, 5.7529585e-05, 2.6181128e-04],
        ...,
        [1.7660727e-03, 6.3926983e-04, 5.1233039e-04, ...,
          3.3867396e-05, 3.0036687e-04, 3.8313960e-05],
        [1.7660727e-03, 6.3926983e-04, 5.1233039e-04, ...,
          3.3867396e-05, 3.0036687e-04, 3.8313960e-05],
        [1.7660727e-03, 6.3926983e-04, 5.1233039e-04, ...,
          3.3867396e-05, 3.0036687e-04, 3.8313960e-05]]], dtype=float32)>
```

Build Training Model

```
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_variables
    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
    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']
```

Output

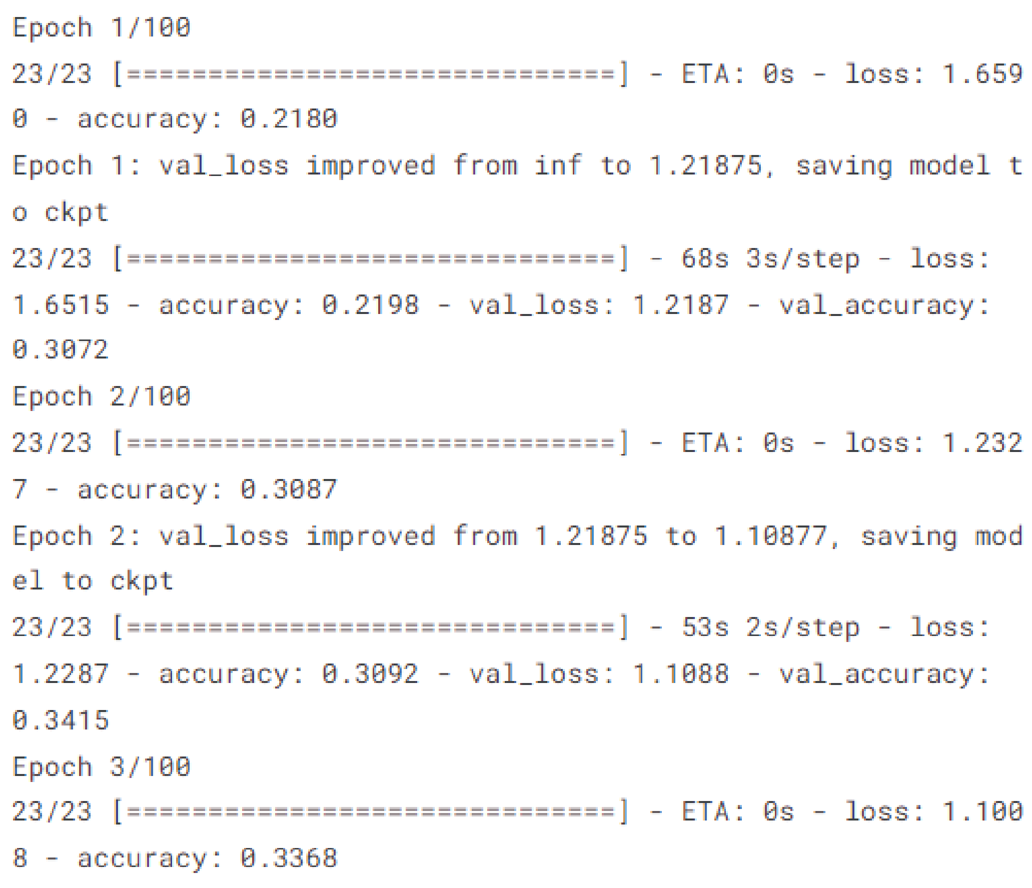


```
<tf.Tensor: shape=(149, 30, 2443), dtype=float32, numpy=
array([[[1.65631573e-04, 2.27935394e-04, 4.19502285e-05, ...,
         3.37221369e-04, 4.60989599e-04, 6.05074420e-05],
        [1.81649835e-03, 8.60798289e-04, 3.85868661e-05, ...,
         2.42381182e-04, 5.70954180e-05, 2.13439023e-04],
        [1.15610070e-04, 4.79279348e-04, 5.39896486e-04, ...,
         8.72829187e-05, 5.75295926e-05, 2.61811161e-04],
        ...,
        [1.76607259e-03, 6.39270060e-04, 5.12330735e-04, ...,
         3.38673926e-05, 3.00367014e-04, 3.83139413e-05],
        [1.76607259e-03, 6.39270060e-04, 5.12330735e-04, ...,
         3.38673926e-05, 3.00367014e-04, 3.83139413e-05],
        [1.76607259e-03, 6.39270060e-04, 5.12330735e-04, ...,
         3.38673926e-05, 3.00367014e-04, 3.83139413e-05]],

       [[1.55812071e-04, 3.79266101e-04, 8.43429807e-05, ...,
         1.87504294e-04, 1.81534083e-03, 4.03375270e-05],
        [6.16104226e-05, 2.10923399e-03, 1.69246734e-04, ...,
         4.28957748e-04, 2.75967235e-04, 3.21038824e-05],
        [9.97629832e-05, 2.07351893e-03, 2.80998338e-05, ...,
         2.69345823e-04, 5.78825595e-04, 1.83789580e-05],
        ...,
        [1.32163626e-03, 2.03588381e-04, 1.09549495e-04, ...,
         6.00988533e-05, 6.16610399e-04, 2.83890142e-04],
        [1.32163626e-03, 2.03588381e-04, 1.09549495e-04, ...,
         6.00988533e-05, 6.16610399e-04, 2.83890142e-04],
        [1.32163626e-03, 2.03588381e-04, 1.09549495e-04, ...,
         6.00988533e-05, 6.16610399e-04, 2.83890142e-04]],

       [[2.23145486e-04, 1.46102102e-04, 7.99294503e-05, ...,
         9.39670572e-05, 2.44671828e-04, 4.33516652e-05],
        [2.13014777e-04, 5.86244627e-04, 6.94656337e-05, ...,
         8.13265942e-05, 1.14615960e-03, 8.34207167e-04],
        [4.97411675e-05, 5.25639516e-05, 5.13565428e-05, ...,
         1.93095271e-04, 2.82236986e-04, 2.61849491e-03],
       ...,
       [5.69541007e-04, 1.86111647e-04, 3.10389238e-04, ...,
        7.51194311e-05, 8.07896504e-05, 4.90598148e-04],
       [5.69541007e-04, 1.86111647e-04, 3.10389238e-04, ...,
        7.51194311e-05, 8.07896504e-05, 4.90598148e-04],
       [5.69541007e-04, 1.86111647e-04, 3.10389238e-04, ...,
        7.51194311e-05, 8.07896504e-05, 4.90598148e-04]],

      ...,

       [[9.60989637e-05, 9.84966027e-05, 2.14608790e-05, ...,
         1.33773050e-04, 1.94961077e-03, 3.93112277e-04],
        [1.04606396e-03, 6.36108889e-05, 1.96550594e-04, ...,
         6.62326638e-05, 6.25509478e-04, 2.59157940e-04],
        [5.77040832e-04, 1.16182011e-04, 4.03432059e-05, ...,
         9.93392532e-05, 3.42387211e-04, 1.96699053e-04],
        ...,
        [1.84956065e-03, 5.57906809e-04, 6.51993760e-05, ...,
         5.02572111e-05, 3.94494244e-04, 1.43843907e-04],
        [1.84956065e-03, 5.57906809e-04, 6.51993760e-05, ...,
         5.02572111e-05, 3.94494244e-04, 1.43843907e-04],
        [1.84956065e-03, 5.57906809e-04, 6.51993760e-05, ...,
         5.02572111e-05, 3.94494244e-04, 1.43843907e-04]],

       [[4.30050095e-05, 2.05650169e-04, 5.73611796e-05, ...,
         3.98737029e-04, 1.37976604e-03, 2.67167343e-04],
        [7.52864798e-05, 4.58297545e-05, 1.86735095e-04, ...,
         3.17098980e-04, 1.92437961e-04, 9.22955689e-04],
        [7.42237389e-05, 1.92209758e-04, 4.29689040e-04, ...,
         2.20148606e-04, 1.62150391e-05, 5.02766110e-04],
        ...,
        [1.59094390e-03, 3.21059546e-04, 6.92526635e-04, ...,
         5.29050267e-05, 5.38064924e-04, 5.55493934e-05],
        [1.59094390e-03, 3.21059546e-04, 6.92526635e-04, ...,
         5.29050267e-05, 5.38064924e-04, 5.55493934e-05],
        [1.59094390e-03, 3.21059546e-04, 6.92526635e-04, ...,
         5.29050267e-05, 5.38064924e-04, 5.55493934e-05]],

       [[2.62904563e-04, 2.58466607e-04, 1.94585838e-04, ...,
         2.18260975e-04, 8.65424983e-04, 1.61971184e-05],
        [9.11809038e-05, 5.44606773e-05, 2.89906224e-04, ...,
         4.13724105e-04, 1.48806648e-04, 2.28498247e-04],
        [2.62596204e-05, 1.59072355e-04, 3.89549496e-05, ...,
         1.51319891e-05, 1.47124752e-04, 1.62018900e-04],
        ...,
        [3.90979555e-03, 1.75515050e-03, 1.22185826e-04, ...,
         2.73425903e-05, 4.04103106e-04, 2.70855126e-05],
        [3.90979555e-03, 1.75515050e-03, 1.22185826e-04, ...,
         2.73425903e-05, 4.04103106e-04, 2.70855126e-05],
        [3.90979555e-03, 1.75515050e-03, 1.22185826e-04, ...,
         2.73425903e-05, 4.04103106e-04, 2.70855126e-05]]], dtype=float32)>
```

Train Model

```
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=True)
  ]
```

Output

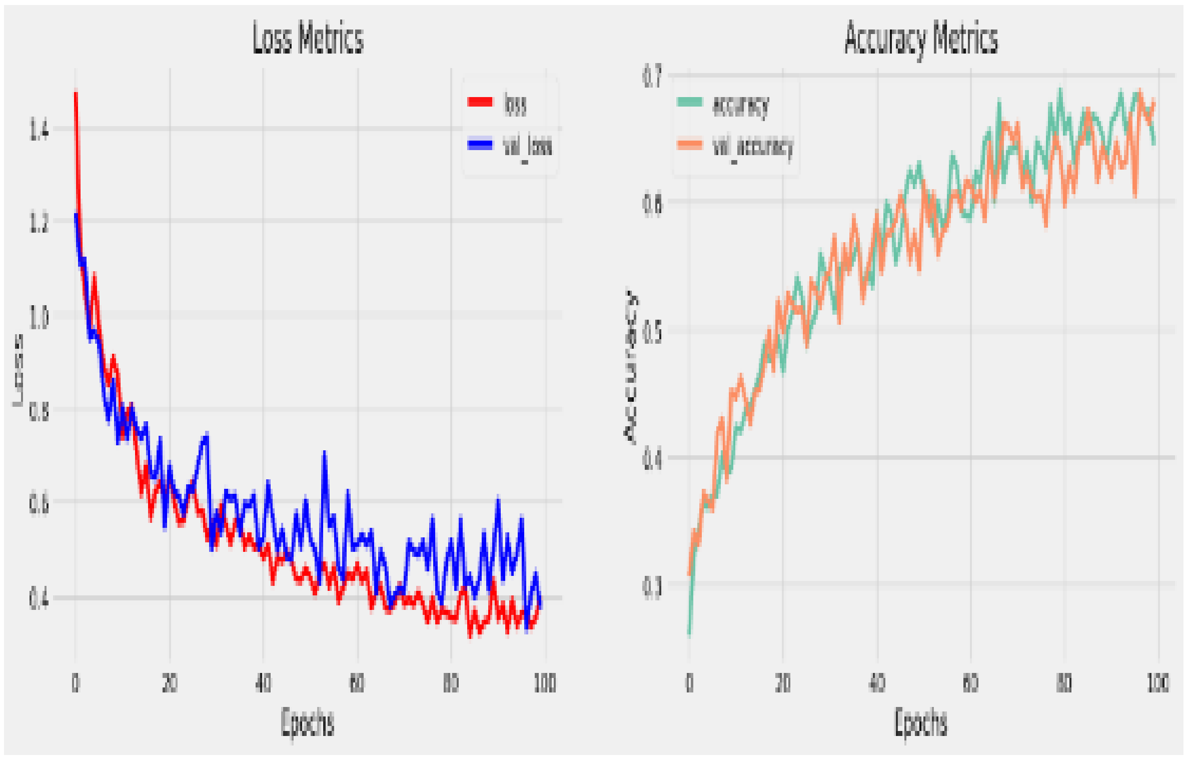


```
Epoch 1/100
23/23 [==============================] - ETA: 0s - loss: 1.659
0 - accuracy: 0.2180
Epoch 1: val_loss improved from inf to 1.21875, saving model t
o ckpt
23/23 [==============================] - 68s 3s/step - loss:
1.6515 - accuracy: 0.2198 - val_loss: 1.2187 - val_accuracy:
0.3072
Epoch 2/100
23/23 [==============================] - ETA: 0s - loss: 1.232
7 - accuracy: 0.3087
Epoch 2: val_loss improved from 1.21875 to 1.10877, saving mod
el 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: 1.100
8 - accuracy: 0.3368
```

Visualize Metrics

```
                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

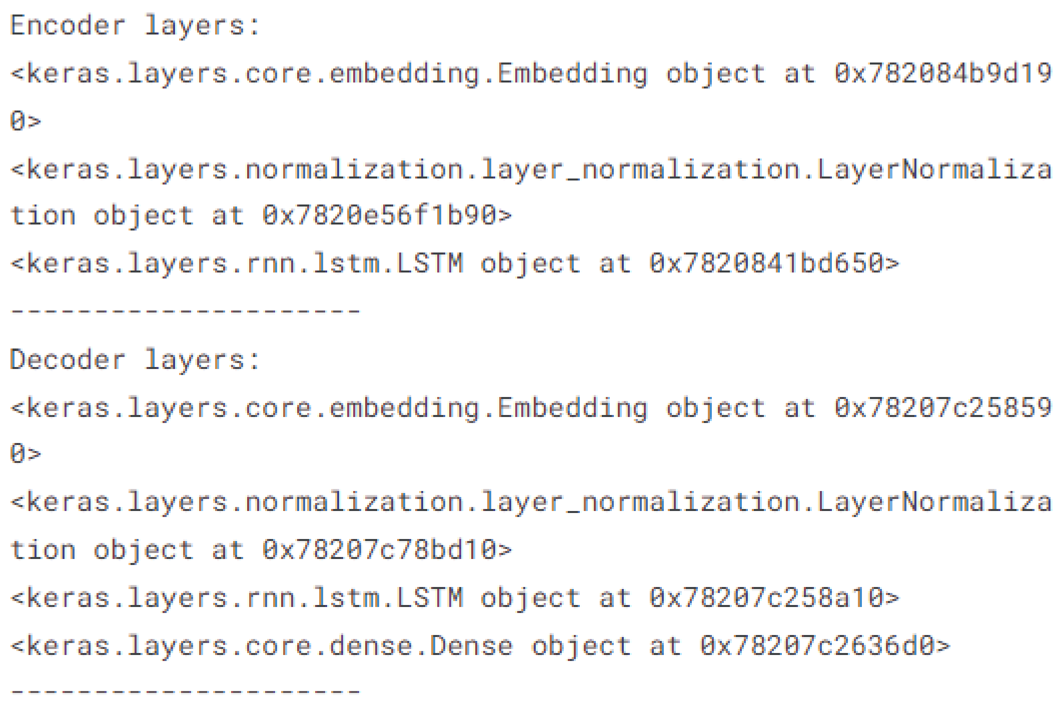


Save Model

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

```
   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 0x782084b9d19
0>
<keras.layers.normalization.layer_normalization.LayerNormaliza
tion object at 0x7820e56f1b90>
<keras.layers.rnn.lstm.LSTM object at 0x7820841bd650>
---------------------
Decoder layers:
<keras.layers.core.embedding.Embedding object at 0x78207c25859
0>
<keras.layers.normalization.layer_normalization.LayerNormaliza
tion object at 0x78207c78bd10>
<keras.layers.rnn.lstm.LSTM object at 0x78207c258a10>
<keras.layers.core.dense.Dense object at 0x78207c2636d0>
---------------------
```

Model

Conclusion:

In conclusion, the "Create a Chatbot in Python" project has provided a foundational understanding of chatbot development using Python and natural language processing techniques. By leveraging NLTK for text processing and TF-IDF for response generation, we have built a simple yet functional chatbot. This project serves as a stepping stone for more complex and advanced chatbot implementations. Further enhancements could include incorporating machine learning models, deep learning techniques, and larger, domain-specific datasets to improve the chatbot's accuracy and conversational abilities. As technology continues to evolve, the potential for chatbots in various applications remains substantial, making this project a valuable starting point for those interested in exploring the field of conversational AI.

Thank you