Course : Artificial Intelligence

Project : Create a Chatbot in Python

Introduction :

Chatbot is a computer program that humans will interact with in natural spoken language and including artificial intelligence techniques such as NLP (Natural language processing) that makes the chatbot more interactive and more reliable.

Based on the recent epidemiological situation, the increasing demand and reliance on electronic education has become very difficult to access to the university due to the curfew imposed, and this has led to limited access to information for academics at the university.

This project aims to build a chatbot for Admission and Registration to answer every person who asks about the university, colleges, majors and admission policy.

```
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 TextVe
ctorization
import re,string
from tensorflow.keras.layers import LSTM,D
ense,Embedding,Dropout,LayerNormalization
```

In [2]:

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

```
Dataframe size: 3725
```

Out[2]:

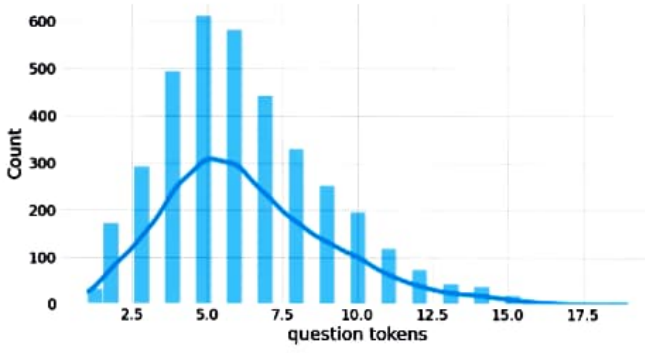
	question	answer
0	hi, how are you doing?	i'm fine. how about yourself?
1	i'm fine. how about yourself?	i'm pretty good. thanks for asking.
2	i'm pretty good. thanks for asking.	no problem. so how have you been?
3	no problem. so how have you been?	i've been great. what about you?
4	i've been great. what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

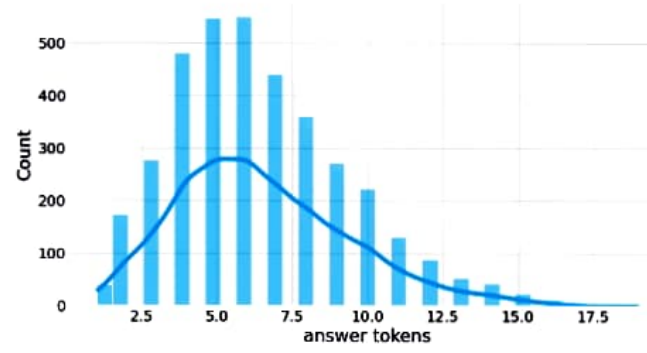
Data Preprocessing

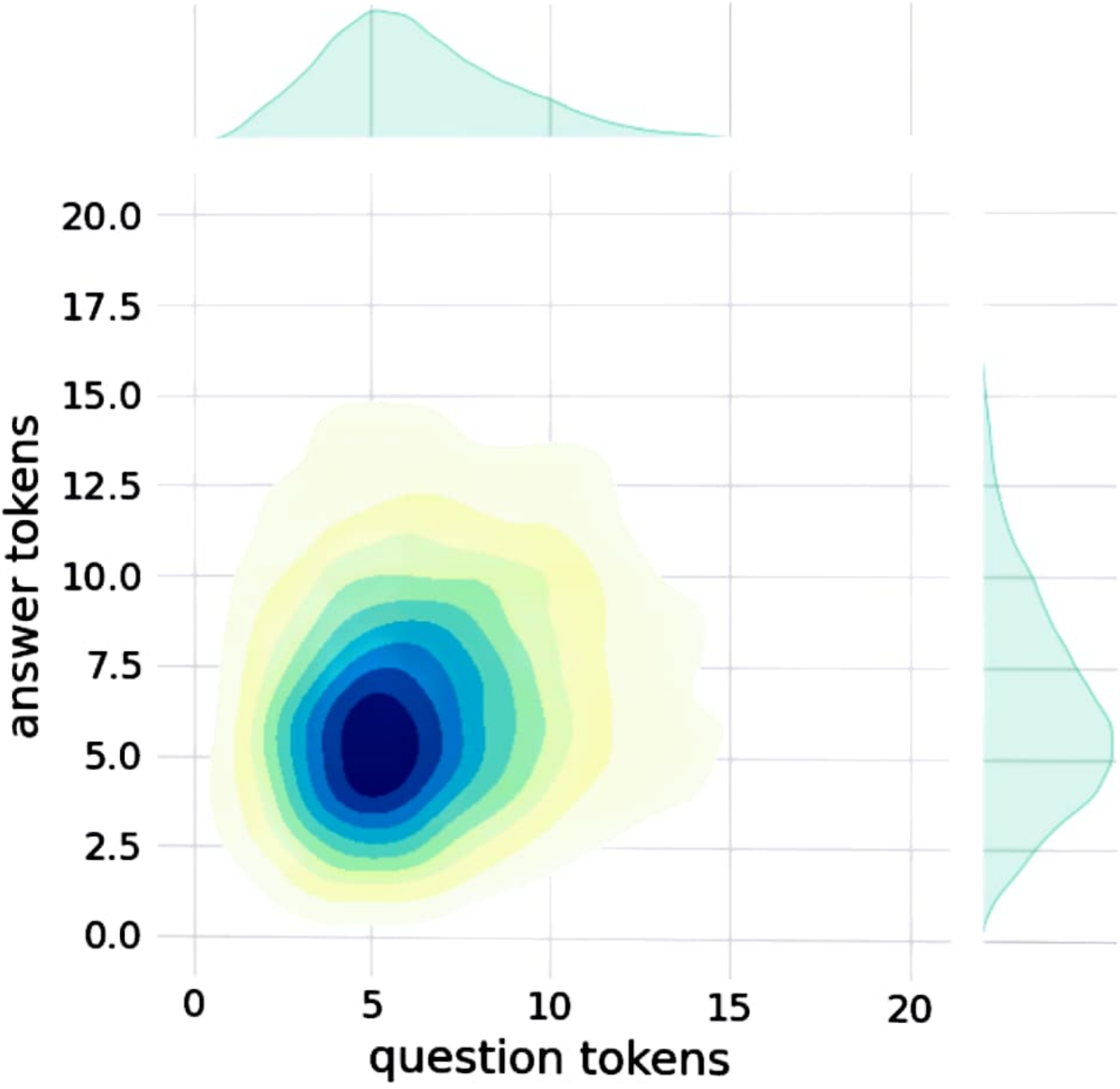
Data Visualization

In [3]:

```
df['question tokens']=df['question'].apply
(lambda x:len(x.split()))
df['answer tokens']=df['answer'].apply(lam
bda x:len(x.split()))
plt.style.use('fivethirtyeight')
fig,ax=plt.subplots(nrows=1,ncols=2,figsiz
e=(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=d
f,kde=True,ax=ax[1])
sns.jointplot(x='question tokens',y='answe
r tokens',data=df,kind='kde',fill=True,cma
p='YlGnBu')
plt.show()
```






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

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
0	hi, how are you doing?	i'm fine. how about yourself?	hi , how are you doing ?	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>
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>
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>

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 ...
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>
6	what school do you go to?	i go to pcc.	what school do you go to ?	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>
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 . <...
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>

In [5]:

```
df['encoder input tokens']=df['encoder_inp
uts'].apply(lambda x:len(x.split()))
df['decoder input tokens']=df['decoder_inp
uts'].apply(lambda x:len(x.split()))
df['decoder target tokens']=df['decoder_ta
```

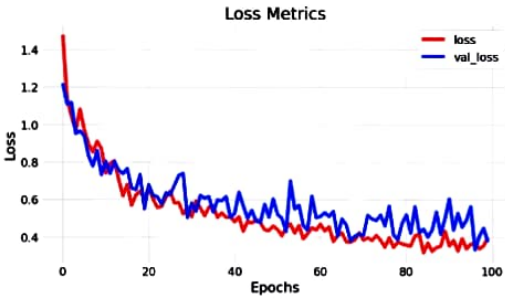
```
0.6791 - val_loss: 0.4475 - val_accuracy:
0.6622
Epoch 100/100
23/23 [==============================] -
ETA: 0s - loss: 0.3358 - accuracy: 0.6787
Epoch 100: val_loss did not improve from
0.33265
23/23 [==============================] -
22s 968ms/step - loss: 0.3385 - accuracy:
0.6773 - val_loss: 0.3742 - val_accuracy:
0.6796
```

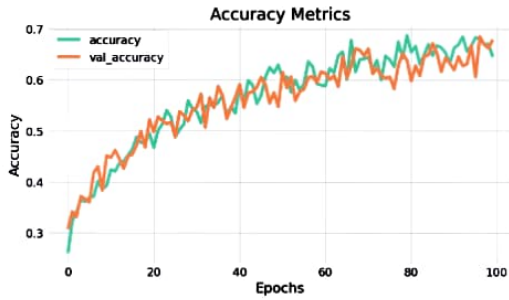
Visualize Metrics

In [16]:

```
fig,ax=plt.subplots(nrows=1,ncols=2,figsiz
e=(20,5))
ax[0].plot(history.history['loss'],label
='loss',c='red')
ax[0].plot(history.history['val_loss'],lab
el='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'],lab
el='accuracy')
ax[1].plot(history.history['val_accurac
y'],label='val_accuracy')
ax[0].legend()
ax[1].legend()
plt.show()
```





Save Model

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 ob
ject at 0x782084b9d190>
<keras.layers.normalization.layer_normali
zation.LayerNormalization object at 0x782
0e56f1b90>
<keras.layers.rnn.lstm.LSTM object at 0x7
820841bd650>
----------------------
Decoder layers:
<keras.layers.core.embedding.Embedding ob
ject at 0x78207c258590>
<keras.layers.normalization.layer_normali
zation.LayerNormalization object at 0x782
07c78bd10>
<keras.layers.rnn.lstm.LSTM object at 0x7
8207c258a10>
<keras.layers.core.dense.Dense object at
0x78207c2636d0>
----------------------
```

In [19]:

```
class ChatBot(tf.keras.models.Model):
    def __init__(self,base_encoder,base_de
coder,*args,**kwargs):
        super().__init__(*args,**kwargs)
        self.encoder,self.decoder=self.bui
ld_inference_model(base_encoder,base_decod
er)

    def build_inference_model(self,base_en
coder,base_decoder):
        encoder_inputs=tf.keras.Input(shap
e=(None,))
        x=base_encoder.layers[0](encoder_i
nputs)
        x=base_encoder.layers[1](x)
        x,encoder_state_h,encoder_state_c=
base_encoder.layers[2](x)
        encoder=tf.keras.models.Model(inpu
ts=encoder_inputs,outputs=[encoder_state_
h,encoder_state_c],name='chatbot_encoder')

        decoder_input_state_h=tf.keras.Inp
ut(shape=(lstm_cells,))
        decoder_input_state_c=tf.keras.Inp
ut(shape=(lstm_cells,))
        decoder_inputs=tf.keras.Input(shap
e=(None,))
        x=base_decoder.layers[0](decoder_i
nputs)
        x=base_encoder.layers[1](x)
```

```
        x,decoder_state_h,decoder_state_c=
base_decoder.layers[2](x,initial_state=[de
coder_input_state_h,decoder_input_state_
c])
        decoder_outputs=base_decoder.layer
s[-1](x)
        decoder=tf.keras.models.Model(
            inputs=[decoder_inputs,[decode
r_input_state_h,decoder_input_state_c]],
            outputs=[decoder_outputs,[deco
der_state_h,decoder_state_c]],name='chatbo
t_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_probabilit
y,temperature=0.5):
        conditional_probability = np.asarr
ay(conditional_probability).astype("float6
4")
        conditional_probability = np.log(c
onditional_probability) / temperature
        reweighted_conditional_probabilit
= 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_lengt
h),dtype=np.int32)
        for i,word in enumerate(text.split
()):
            seq[:,i]=sequences2ids(word).n
umpy()[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,trai
ning=False)
        target_seq=np.zeros((1,1))
        target_seq[:,:]=sequences2ids(['<s
tart>']).numpy()[0][0]
        stop_condition=False
        decoded=[]
        while not stop_condition:
            decoder_outputs,new_states=sel
f.decoder([target_seq,states],training=Fa
se)
```

```
#                 index=tf.argmax(decoder_outpu
ts[:,-1,:],axis=-1).numpy().item()
                index=self.sample(decoder_outp
uts[0,0,:]).item()
                word=ids2sequences([index])
                if word=='<end> ' or len(decod
ed)>=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(ids2sequen
ces(decoded))

chatbot=ChatBot(model.encoder,model.decode
r,name='chatbot')
chatbot.summary()
```

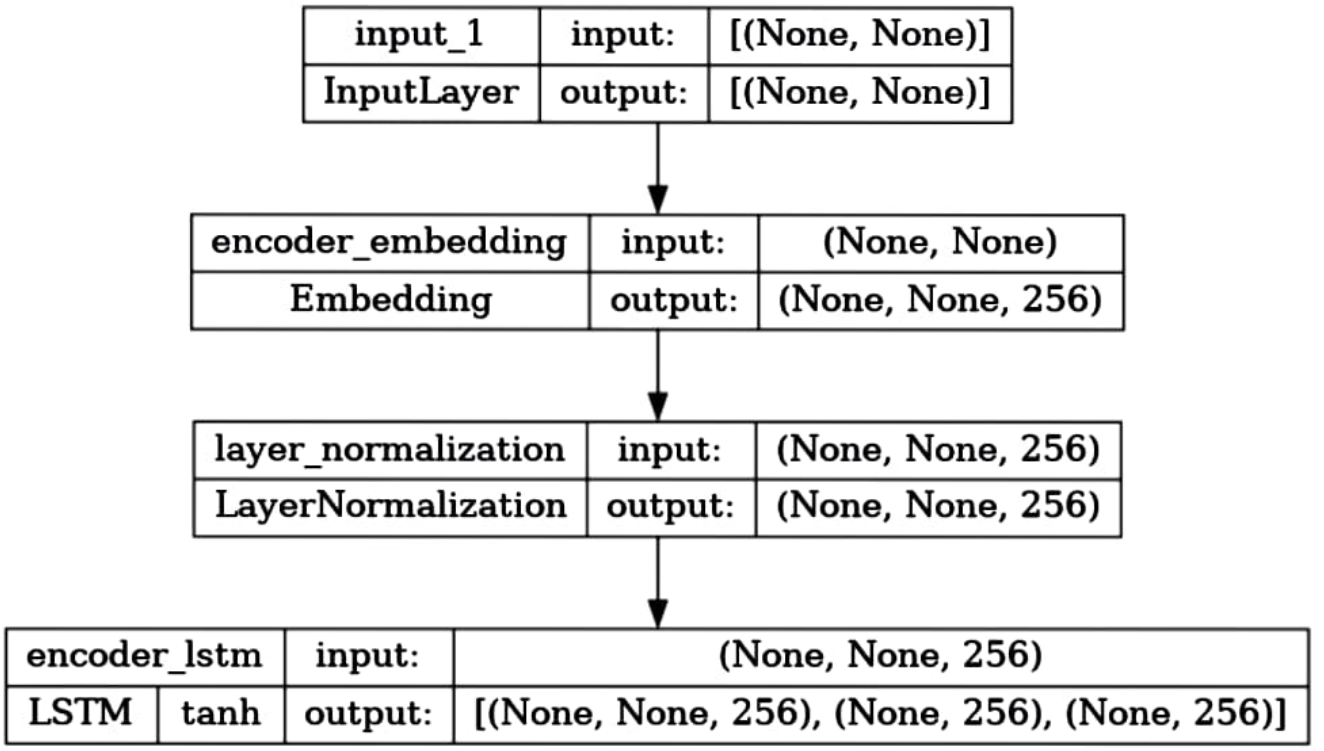
```
Model: "chatbot_encoder"
_________________________________________
________________________
 Layer (type)                Output Shape
Param #
=========================================
========================
 input_1 (InputLayer)        [(None, Non
e)]               0
```

```
Total params: 1,779,083
Trainable params: 1,779,083
Non-trainable params: 0
_________________________________________________________________
_________________________________________________________________
```

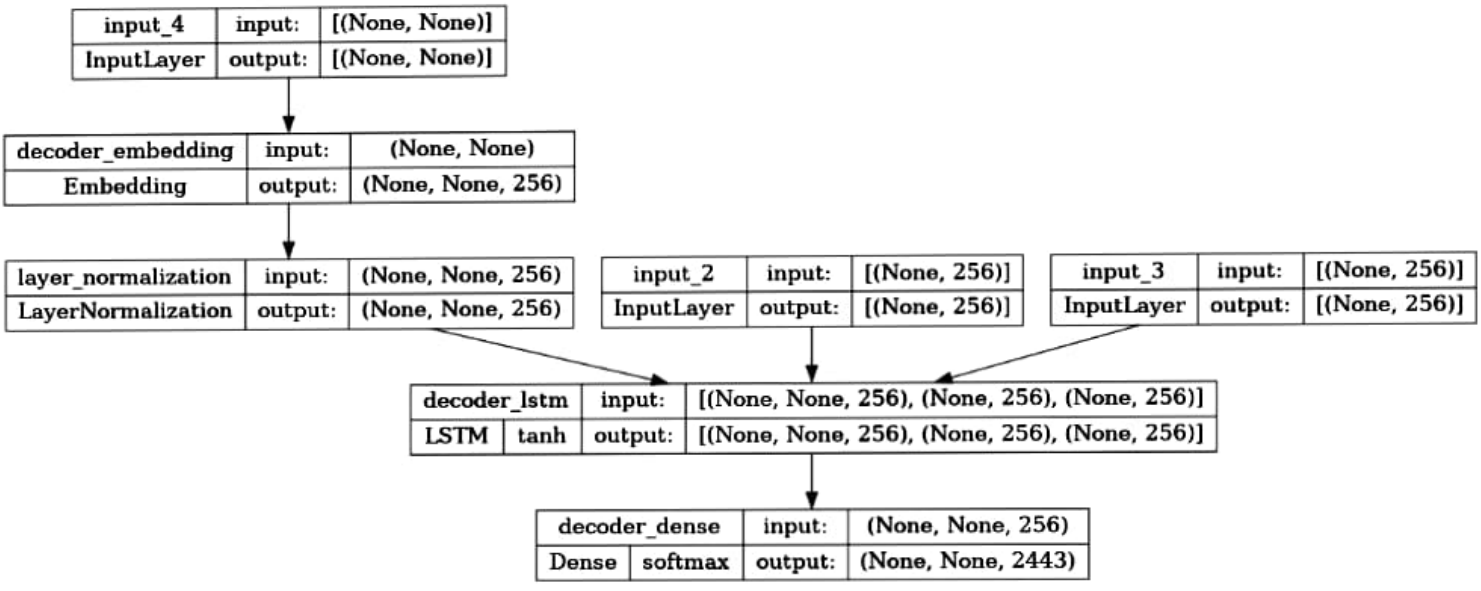
In [20]:

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

Out[20]:



Out[21]:



Time to Chat

In [22]:

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

Conclusion

This bot was built to respond to the inquiries of the Tawjihi students regarding each of the university's faculties and their specializations, with extracted information for each specialization, familiarizing students with the level exams that students submit about their enrollment in the university, introducing the educational qualification diploma program and the mechanism for joining it. Giving students notes on the electronic enrollment application package, the locations of approved banks, and how to fill out the application.