**CREATE A CHATBOT IN PYTHON**

**COLLEGE NAME:ARULMIGU MEENAKSHI AMMAN COLLEGE OF ENGINEERING**

**COLLEGE CODE:5103**

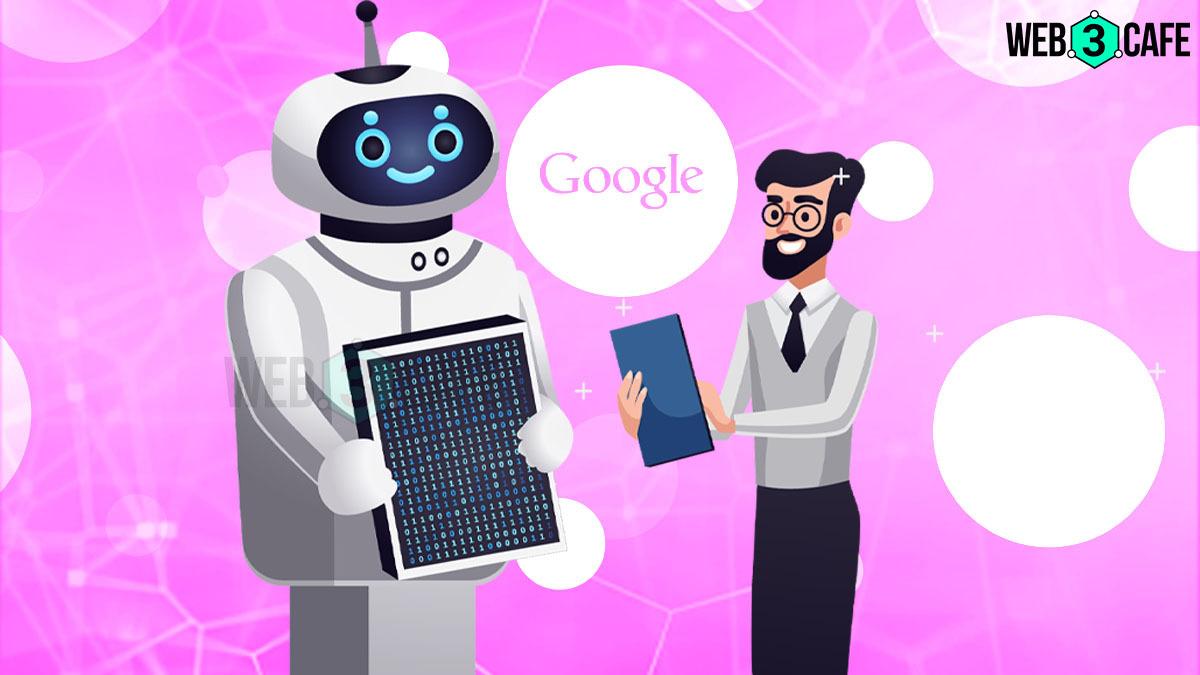
**TEAM MEMBER**

**REG NO: 510321205003**

**NAME:GOWTHAM.V**

**PHASE-2 PROJECT SUBMISSION**

**PROJECT:CHATBOT IN PYTHON**



Here is how to apply design thinking to chatbot design:

**1. Empathize:**

Start by understanding your users and their needs. What are their goals? What challenges do they face? What kind of experience do they want to have with your chatbot?

You can gather this information through user research methods such as interviews, surveys, and usability testing.

**2. Define:**

Once you understand your users, you can define the problem you want to solve with your chatbot. What specific task or task do you want it to be able to perform?

Be as specific as possible, and avoid defining the problem in terms of the solution. For example, instead of saying "I want to create a chatbot that helps people book flights," you could say "I want to create a chatbot that helps people book flights on our airline, without having to go through our website or call our customer service center."

**3. Ideate:**

Once you have a clear definition of the problem, you can start generating ideas for solutions. This is where design thinking gets creative!

Brainstorm as many ideas as you can, no matter how crazy they seem. The goal is to come up with as many different ways to solve the problem as possible.

**4. Prototype:**

Once you have a list of ideas, it's time to start prototyping. This means building a working model of your chatbot so you can test it with users.

Prototypes don't have to be perfect. They can be as simple as a wireframe or a mockup. The important thing is that they are functional and allow you to test the core user experience.

**5. Test:**

Once you have a prototype, you can start testing it with users. This is the most important step in the design thinking process, as it allows you to get feedback on your chatbot and make improvements.

Pay attention to how users interact with your chatbot. What are they doing easily? What are they struggling with? What features do they like? What features do they not like?

Use this feedback to iterate on your prototype and make it better.

Repeat the process

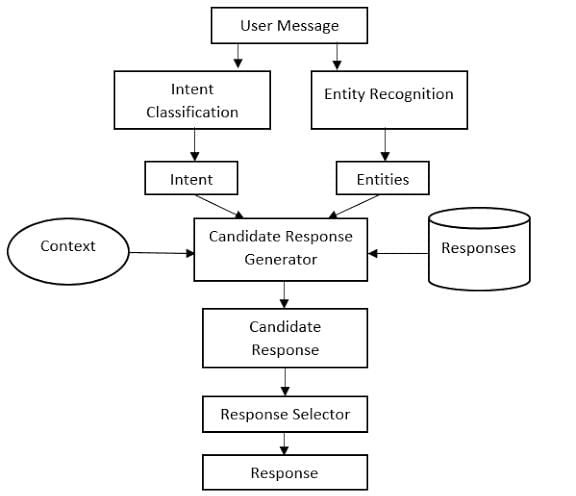
Design thinking is an iterative process, so be prepared to repeat the steps above as needed. The more you test your chatbot with users, the better it will be.

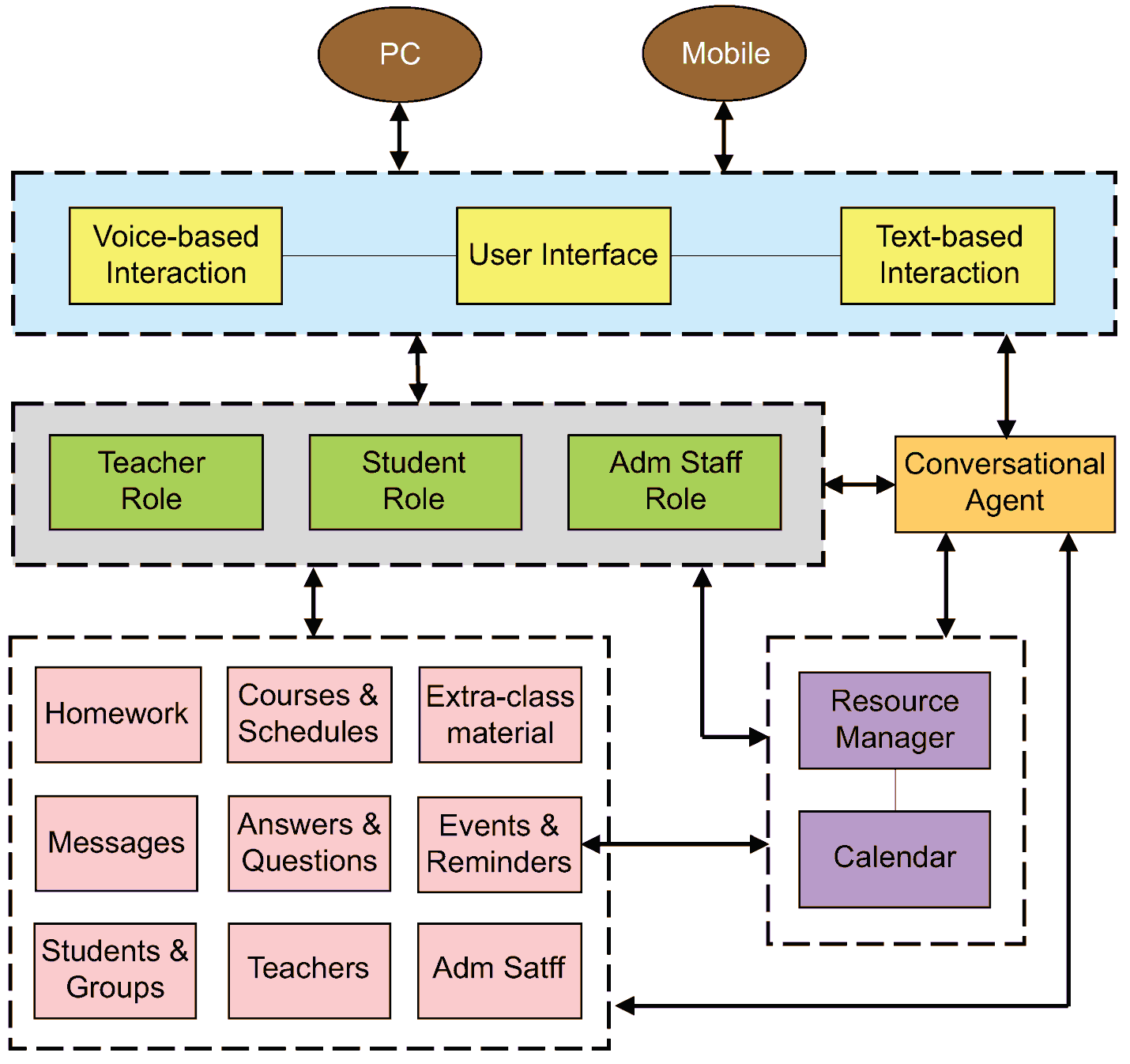
**Here are some additional tips for designing chatbots**:

* Keep your chatbot's purpose simple. Focus on one or two core tasks that it can do well.
* Use natural language processing (NLP) to create a chatbot that can understand and respond to human language in a natural way.
* Design a chatbot that is easy to use and navigate.
* Make your chatbot's personality consistent with your brand.
* Test your chatbot thoroughly before launching it to the public.

By following these tips, you can use design thinking to create a chatbot that is useful, engaging, and effective.

**DESIGNING ARCHITECTURE OF CHATBOT:**



****

**OBJECTIVCE:**

The objective for creating a chatbot in Python can vary depending on the specific needs of the user or organization. However, some common objectives include.

**PROVIDE CUSTOMER SUPPORT:**

Chatbots can be used to provide 24/7 customer support, answer questions, and resolve issues. This can free up human customer support representatives to focus on more complex tasks**.**

**AUTOMATE TASKS:**

Chatbots can be used to automate a variety of tasks, such as scheduling appointments, booking reservations, and placing orders. This can save time and improve efficiency for businesses and organizations**.**

**GENERATE LEADS AND SALES:**

Chatbots can be used to generate leads and sales by qualifying potential customers and providing them with information about products and services

****

**DATA SOURCE :**

Once you have chosen a data source, you will need to clean and prepare the data for training your chatbot. This may involve removing stop words, correcting spelling errors, and formatting the data in a consistent way.

**DATA SET LINK**:( <https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot>.)

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.

**DATA PREPROCESSING:**

Data preprocessing is an important step in creating a chatbot in Python. It involves cleaning and preparing the data that will be used to train the chatbot. This can help to improve the performance of the chatbot and make it more accurate and engaging.

**HERE ARE SOME COMMON DATA PREPROCESSING STEPS FOR CHATBOTS:**

**REMOVE STOP WORDS:**

Stop words are common words that do not add much meaning to a sentence, such as "the", "is", and "of". Removing stop words can help to reduce the size of the training data and improve the performance of the chatbot.

**CORRECT SPELLING ERRO:**

Spelling errors can make it difficult for the chatbot to understand the user's input. Correcting spelling errors can help to improve the accuracy of the chatbot.

**FORMAT THE DATA IN A CONSISTENT WAY:**

The training data should be formatted in a consistent way to make it easier for the chatbot to learn. This may involve converting the data to lowercase, removing punctuation, and splitting the data into tokens.

**LEMMATIZE THE DATA:** Lemmatization is the process of converting a word to its base form. This can help to improve the accuracy of the chatbot, especially for chatbots that are designed to answer questions about a specific topic.

**REMOVE DUPLICATE DATA:**

Duplicate data can make it difficult for the chatbot to learn. Removing duplicate data can help to improve the performance of the chatbot

**PYTHON CODE:**

**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()

**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

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)

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>

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 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

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]}')

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}')

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]} ...')

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=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)

**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('---------------------')

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>

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

---------------------

**CREATE INFERENCE MODEL:**

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\_state\_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\_state=[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:

decoder\_outputs,new\_states=self.decoder([target\_seq,states],training=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]']

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]']

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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)

**TIME TO CHAT:**

def print\_conversation(texts):

for text in texts:

print(f'You: {text}')

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

print('========================')

print\_conversation([‘hi,how are you doing?’,

‘how about your self?’,

‘what about you?’

‘what school do you go to?’

‘do you like it there?’

‘how’s it going?’

‘are you enjoying it there?’

‘how are you doing today?’])

**OUTPUT:**

**You:hi,how are you doing?**

**Bot:I am fine.**

**=============================================**

**You:how about your self?**

**Bot:I am pretty good.**

**==============================================**

**You:what about you?**

**Bot:I’ve been good.i’m in school right now.**

**===============================================**

**You:what school do you go to?**

**Bot:I go to pcc.**

**================================================**

**You:how’s it going?**

**Bot:I am doing well.**

**=================================================**

**You:do you like it there?**

**Bot:it’s ok.it’s a really big campus.**

**==================================================**

**You:are you enjoing it there?**

**Bot:it’s not bad.there are lot of people there.**

**===================================================**

**You:how are doing today?**

**Bot:I am doing great.**

**====================================================**

**CONCLUSION:**

In conclusion, building a chatbot in Python is a rewarding and educational experience that can expose you to a variety of useful skills, including natural language processing, machine learning, and software engineering. Chatbots can be used for a wide range of purposes, from customer service to education to entertainment.