CREATE A CHATBOT USING PYTHON

PHASE 5 : PROJECT DOCUMENTATION

AND SUBMISSION ......

PROJECT : Create Chatbot Using Python.

Table of Contents

1.Introduction:

- Creating a chatbot using Python involves developing a program that can simulate human conversation.
- You'll need to decide on the type of chatbot, such as rule-based or AI-powered, select the right tools and libraries, design the conversation flow, write the code, test it thoroughly, and then deploy it.
- Continuous maintenance and improvement are crucial for keeping your chatbot effective.
- Python's versatility and the availability of NLP and ML libraries make it a powerful choice for chatbot development, offering the potential to enhance user experiences, automate tasks, and provide valuable support in various applications.
- Chatbots have gained tremendous popularity in recent years, transforming the way we interact with technology and the internet.
- These conversational agents are designed to simulate human-like conversations and can be found in a wide range of applications, from customer support to virtual assistants.
- Python, a versatile and widely-used programming language, is an excellent choice for developing chatbots due to its extensive libraries and frameworks for natural language processing.
- In this guide, we will delve into the fascinating world of chatbot development using Python.
- We will explore the essential steps required to create a functional chatbot, from designing its purpose and personality to implementing the underlying logic and deploying it for real-world use.
- Whether you're a seasoned developer or just getting started, this document will provide you with the fundamental knowledge needed to embark on your chatbot development journey.

Let's embark on this exciting journey of creating a chatbot that can engage with users, provide information, and offer a personalized experience.

2. Prerequisites:

Before you start creating a chatbot using Python, there are several prerequisites you should consider. These prerequisites will help ensure a smooth and successful development process:

1) Programming Knowledge: You should have a good understanding of Python programming. If you're new to Python, it's essential to learn the language's fundamentals.
2) Python Installed: Make sure Python is installed on your computer. You can download the latest version of Python from the official Python website.
3) Code Editor or IDE: Choose a code editor or integrated development environment (IDE) for writing and running your Python code. Some popular options include Visual Studio Code, PyCharm, Jupyter Notebook, or a simple text editor like Notepad or Sublime Text.
4) Python Libraries: Familiarize yourself with relevant Python libraries and frameworks for chatbot development, such as:
5) ChatterBot: A Python library for creating chatbots.
6) Natural Language Toolkit (NLTK): An essential library for natural language processing tasks.
7) TensorFlow or PyTorch (for advanced machine learning): If you plan to develop more advanced chatbots using deep learning.
8) Data: Consider the source of data or a dataset if you plan to train your chatbot. You may need conversational data or a specific domain's data, depending on your chatbot's purpose.

9) : Chatbots heavily rely on NLP techniques. Familiarize yourself with the basics of NLP, including concepts like tokenization, part-of-speech tagging, and named entity recognition.

10)Design Plan: Define the purpose of your chatbot, its personality, and its primary use cases. Create a design plan outlining the conversation flow, expected user inputs, and desired responses.

11)APIs and Integrations (if applicable): If your chatbot will interact with external services or databases, be prepared to understand and integrate with APIs.

12)Testing Environment: Set up a testing environment where you can experiment with your chatbot without impacting users. This can be a local development environment or a staging server.

13)Git and Version Control: Familiarity with version control systems like Git can be helpful for maintaining and collaborating on your chatbot's code.

14)Server or Hosting: If you plan to deploy your chatbot for public use, ensure you have access to a server or cloud hosting service for deployment.

15)Privacy and Security Considerations: Depending on your chatbot's purpose, you may need to consider data privacy and security aspects.

Data Source :

A good data source for creating a chatbot should contain accurate ,complete ,and easy accessible one for users.

Dataset 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.	thank you very much.
how's it going?	i'm doing well. how about you?
i'm doing well. how about you?	never better, thanks.
never better, thanks.	so how have you been lately?
so how have you been lately?	i've actually been pretty good. you?
i've actually been pretty good. you?	i'm actually in school right now.
i'm actually in school right now.	which school do you attend?
which school do you attend?	i'm attending pcc right now.
i'm attending pcc right now.	are you enjoying it there?
are you enjoying it there?	it's not bad. there are a lot of people there.
it's not bad. there are a lot of people there.	good luck with that.
good luck with that.	thanks.
how are you doing today?	i'm doing great. what about you?
i'm doing great. what about you?	i'm absolutely lovely, thank you.
i'm absolutely lovely, thank you.	everything's been good with you?
everything's been good with you?	i haven't been better. how about yourself?
i haven't been better. how about yourself?	i started school recently.
i started school recently.	where are you going to school?
where are you going to school?	i'm going to pe.............

....

3. Designing the Chatbot :

- Designing a chatbot is a critical step that sets the foundation for its functionality and user experience. It involves defining the chatbot's purpose, persona, and features, as well as designing its conversation flow and user interface.
- The design should align with the intended use case and cater to user preferences, making the chatbot effective and engaging.
- Elements such as language, tone, input handling, privacy considerations, and integration with external systems must be carefully planned.
- Furthermore, scalability, maintenance, and user testing should also be part of the design process to ensure a successful chatbot project.
- A well-designed chatbot not only streamlines the development process but also enhances user satisfaction when the chatbot is deployed.

The design of your chatbot is a critical step in the development process. You should consider the following aspects:

a.) Purpose :

Define the purpose of your chatbot. What problem will it solve or what service will it provide? Is it for customer support, information retrieval, or just for fun?

b.)User Interface :

Decide on the user interface. Will it be a text-based chatbot or a voice-based one? For this document, we will focus on a text-based chatbot.

c.) Personality :

Think about the personality of your chatbot. Is it formal or informal? Does it have a name or persona?

d.) Features :

List the features your chatbot will provide. Will it answer frequently asked questions, provide recommendations, or engage in casual conversation?

e.) Conversation Flow:

Design the conversation flow. What questions will the chatbot ask, and how will it respond to user inputs?

Pre-trained Language Models :

Pre-trained language models, such as GPT-3 (Generative Pre-trained Transformer 3),are a type of artificial intelligence model used for various natural language processing tasks.These models have gained significant attention and popularity in recent years due to theirability to understand and generate human-like text.

Here's an explanation of pre-trained language models:

1.)Pre-training:

Pre-trained language models are created by training on a massive corpus of textfrom the internet. They use a type of neural network architecture called a Transformer.During pre-training, the model learns to predict the next word in a sentence based on thecontext of the preceding words. This process helps the model learn grammar, vocabulary,and a general understanding of language from diverse sources of text.

2.)Transfer Learning:

After pre-training on this vast amount of text, the model is fine-tuned for specificdownstream tasks. Transfer learning is a key concept here. Instead of training a separatemodel for each task (e.g., text classification, language translation, question answering), pretrained models can be adapted for various tasks by fine-tuning them on smaller, taskspecific datasets. This fine-tuning process allows the model to specialize and adapt tospecific tasks without having to start from scratch.

3.) Versitality:

Pre-trained language models like GPT-3 are known for their versatility. They can be applied to a wide range of natural language understanding and generation tasks, including but not limited to:

- Text generation: Creating human-like text, such as articles, stories, or code.
- Sentiment analysis: Determining the sentiment (positive, negative, neutral) of a piece of text.
- Language translation: Translating text from one language to another.
- Text summarization: Condensing long texts into shorter summaries.
- Question answering: Providing answers to questions based on a given text.
- Fine-tuning and Customization:One of the strengths of pre-trained models is their adaptability.

Users can fine-tunethese models on specific datasets to make them more domain-specific or task-specific. This allows organizations and developers to customize the model's behavior for their particular applications.

✓ In summary, pre-trained language models like GPT-3 are powerful tools that have thepotential to revolutionize natural language processing tasks.

✓ They are versatile, adaptable, and accessible, but their use also comes with ethical responsibilities and the need to address biases and potential misuse.

A High-level overview for creating a chatbot using the Hugging Face Transformers library and the GPT-3 model :

- ✓ Install necessary libraries:

  You will need the Transformers library from Hugging Face and the Torch library for PyTorch.

- ✓ You can install them using pip:

---->pip install transformers torch

4. Implementing the Chatbot :

Now, let's start implementing the chatbot:

Set up your chatbot code:

```
import torch
from transformers import GPT2LMHeadModel, GPT2Tokenizer
# Load pre-trained GPT-3 model and tokenizer
model_name = "gpt2"
tokenizer = GPT2Tokenizer.from_pretrained(model_name)
model = GPT2LMHeadModel.from_pretrained(model_name)
# Set the device (CPU or GPU)
device = "cuda" if torch.cuda.is_available() else "cpu"
model.to(device)
# Function to generate a response
def generate_response(prompt):
 input_ids = tokenizer.encode(prompt, return_tensors="pt").to(device)
 output = model.generate(input_ids, max_length=100,
num_return_sequences=1,
pad_token_id=50256)
 response = tokenizer.decode(output[0], skip_special_tokens=True)
 return response

# Chat loop
print("Chatbot: Hi, how can I help you today?")
```

```
while True:
 user_input = input("You: ")
 if user_input.lower() == 'exit':
 print("Chatbot: Goodbye!")
 break
 response = generate_response(user_input)
 print("Chatbot:", response)
```

• This code sets up a chatbot using the GPT-2 model from Hugging Face's Transformers

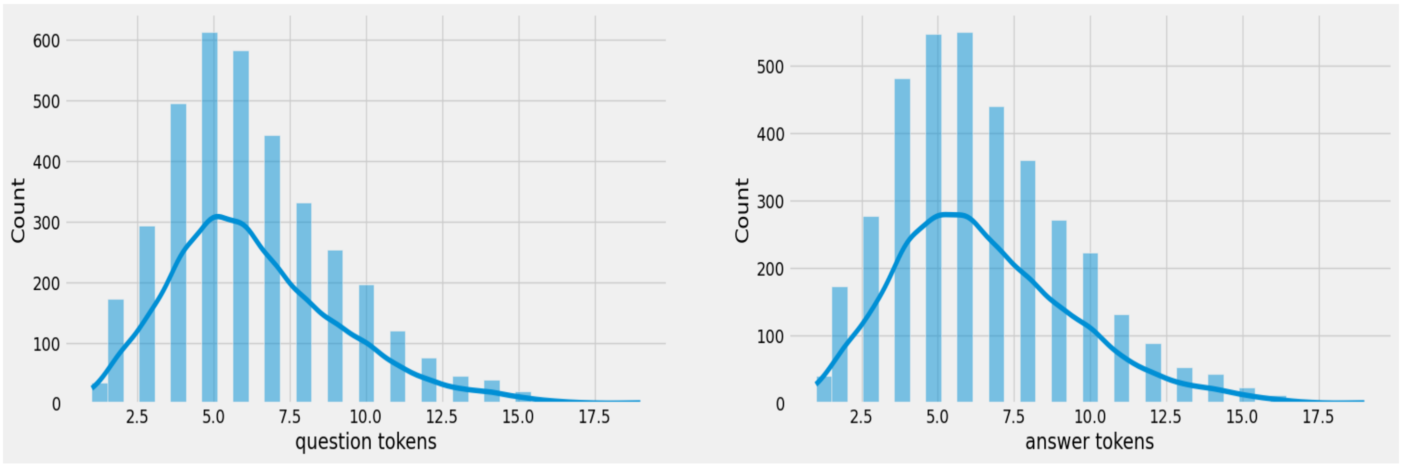
library.

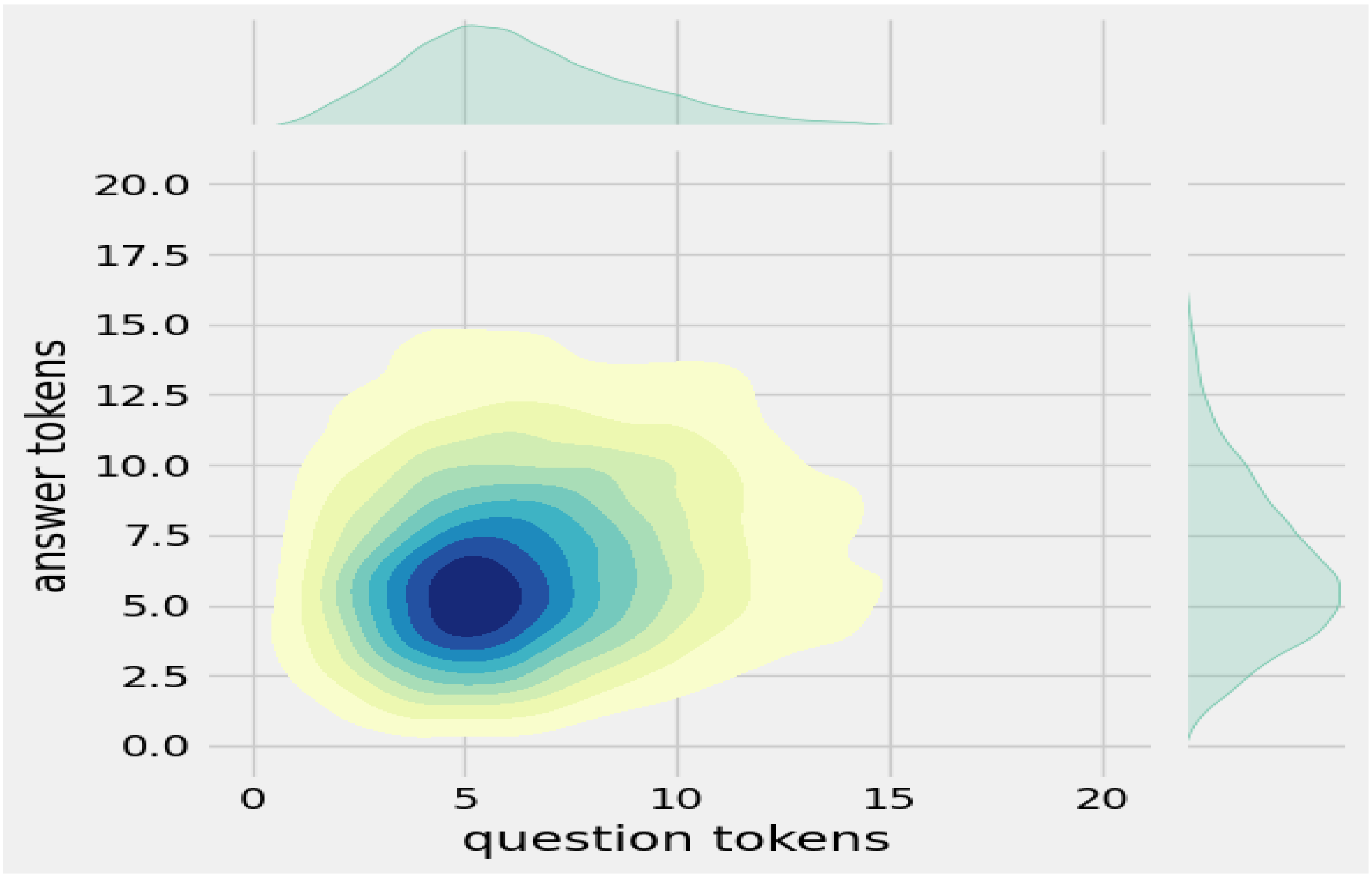
• It takes user input, generates a response, and prints it

• You can extend and refine this code to fit your specific requirements.

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()
```



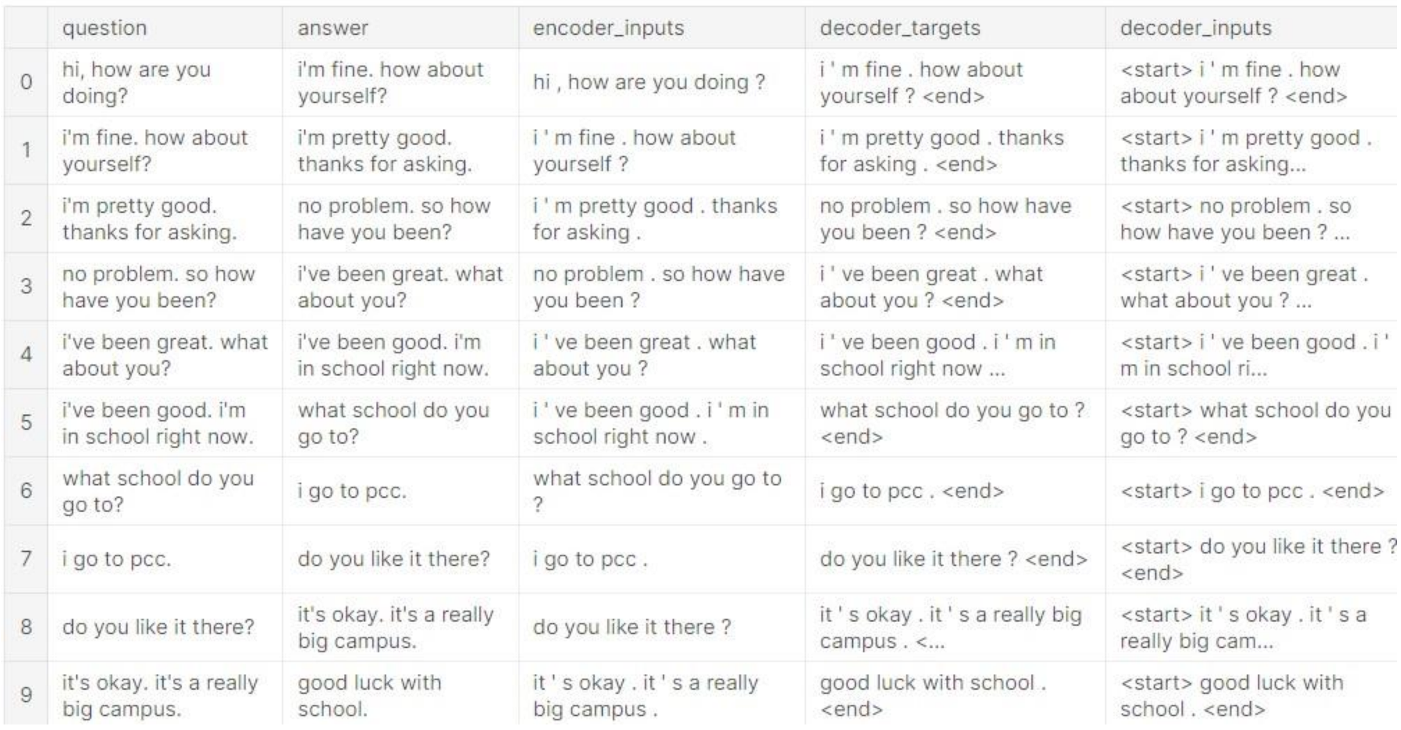


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
answer tokens
0
5
10
15
20
question tokens

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
```

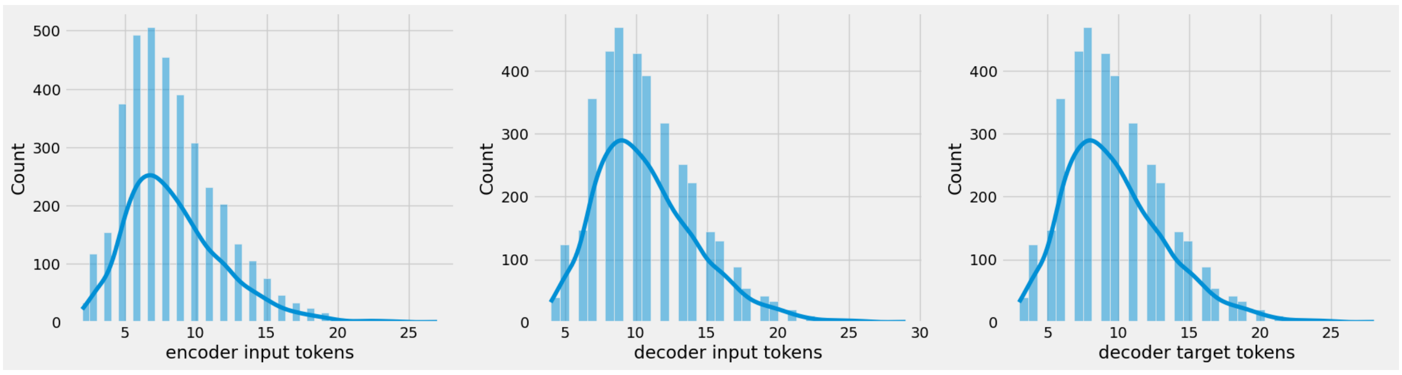
```
df.drop(columns=['answer tokens','question tokens'],axis=1,inplace=True)

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>'
```

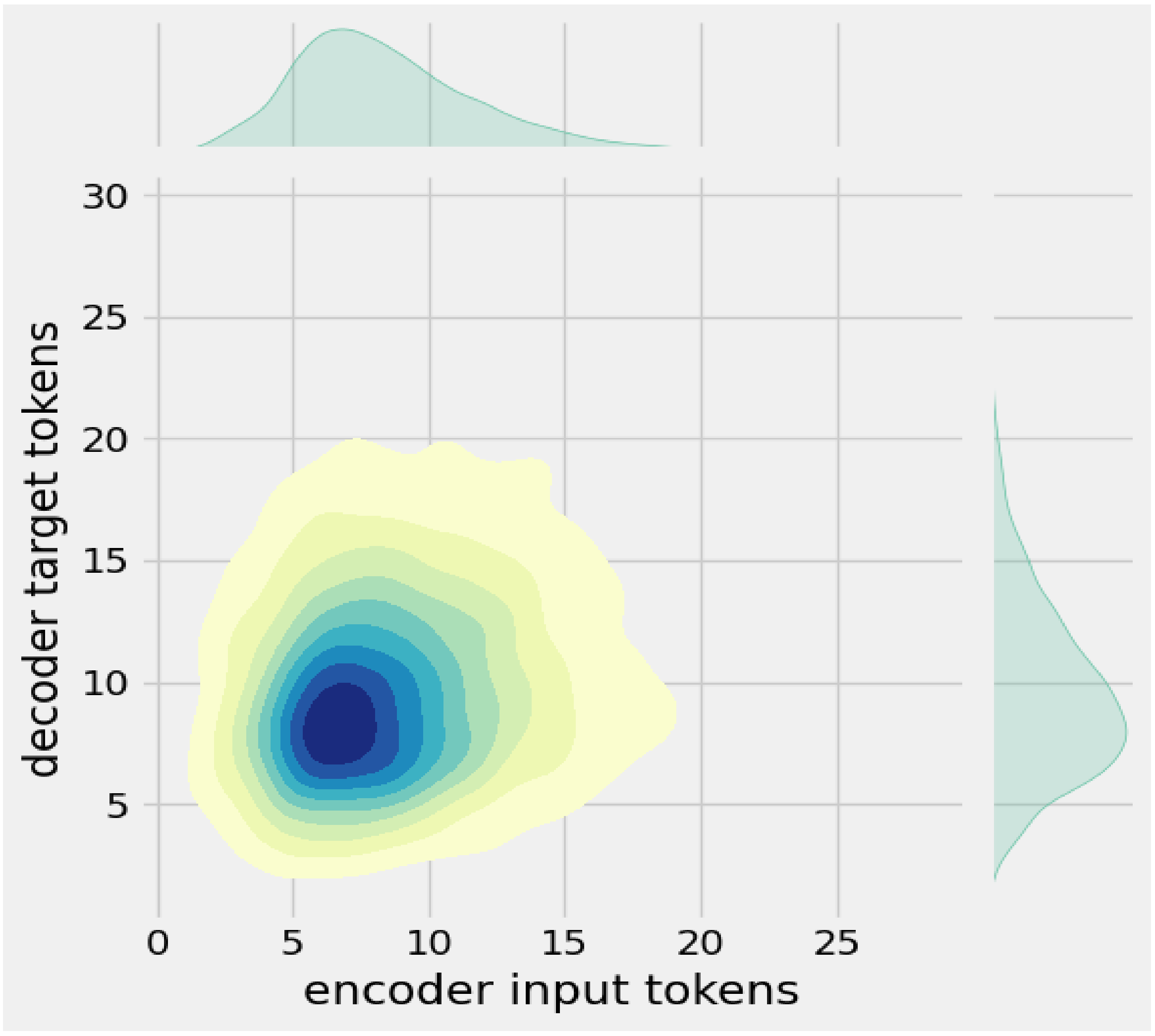


	question	answer	encoder_inputs	decoder_targets	decoder_inputs
0	hi, how are you doing?	i'm fine. how about yourself?	hi , how are you doing ?	i ' m fine . how about yourself ? <end>	<start> i ' m fine . how about yourself ? <end>
1	i'm fine. how about yourself?	i'm pretty good. thanks for asking.	i '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

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]}')
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}')
```

```
Number of train batches: 23
Number of training data: 3427
Number of validation batches: 3
Number of validation data: 447
```

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])
```

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]))
```

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
```

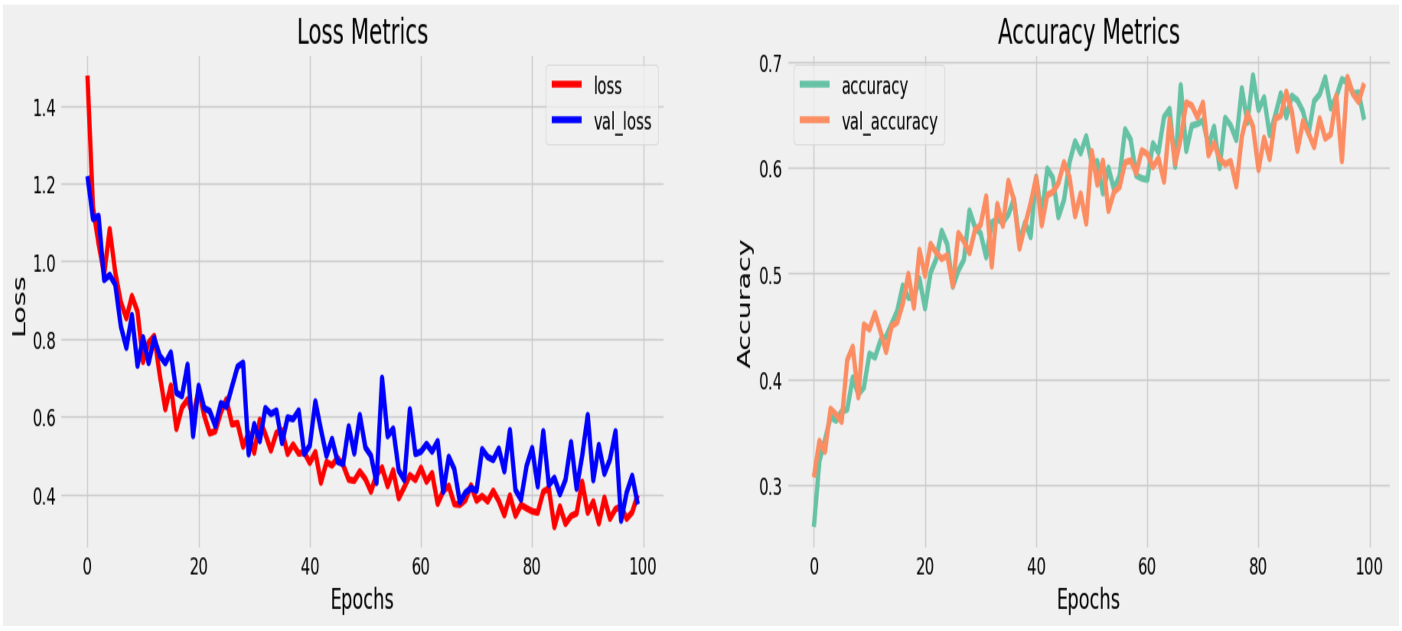
```
model=ChatBotTrainer(encoder,decoder,name='chatbot_trainer')
model.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
    weighted_metrics=['loss','accuracy']
)
```

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)
   ]
)
```

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()
```


Loss Metrics
loss
val_loss
Loss
Epochs
Accuracy Metrics
accuracy
val_accuracy
Accuracy
Epochs

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('---------------------')
```

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_decod
er)


    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=[dec
oder_input_state_h,decoder_input_state_c])
```

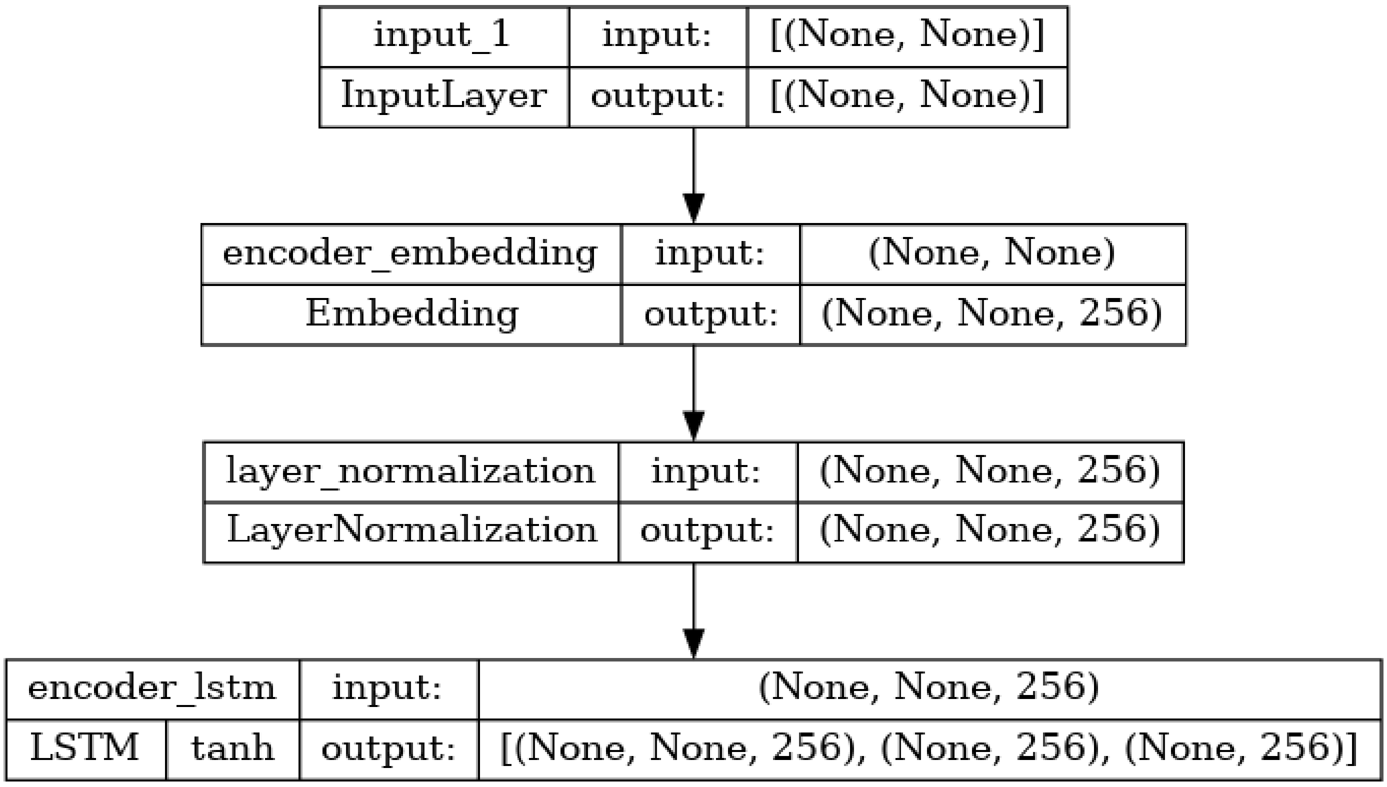
```
        decoder_outputs=base_decoder.layers[-1](x)
        decoder=tf.keras.models.Model(

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)
```

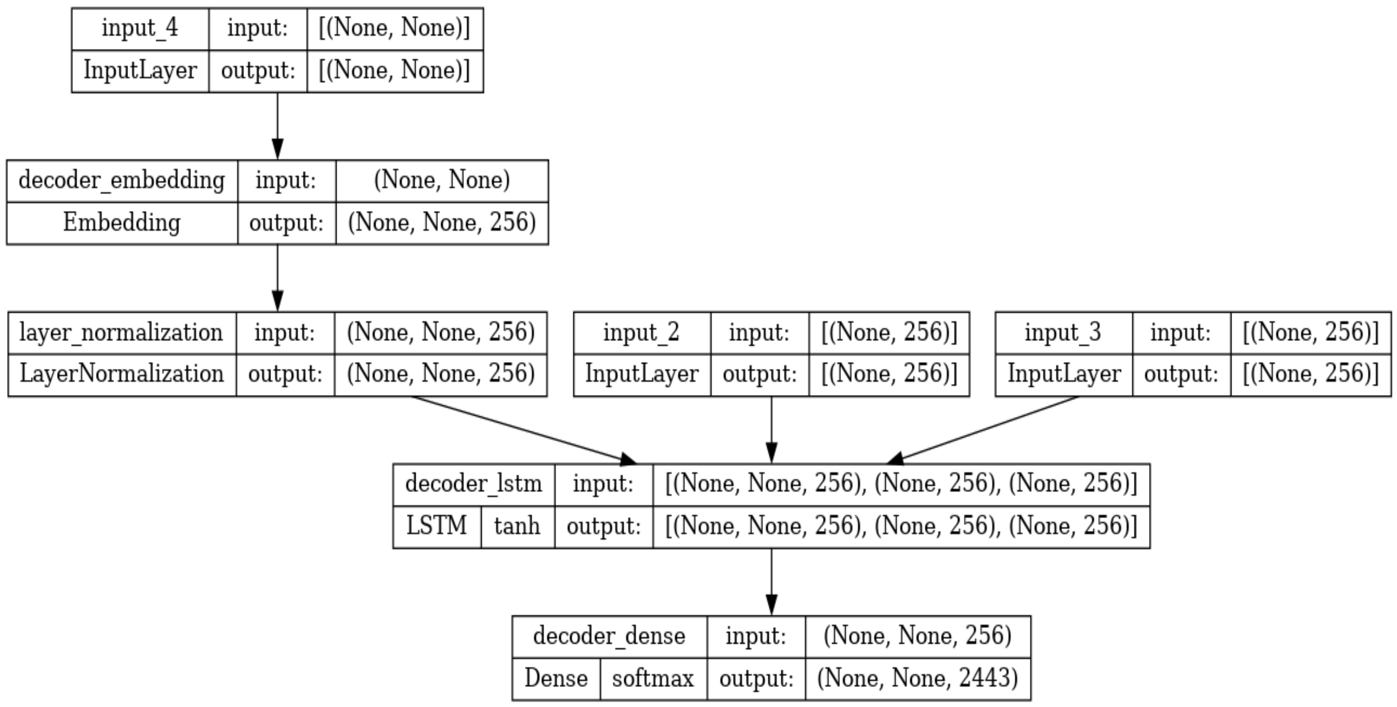
```
      text=re.sub(' [:] ',': ',text)
      text=re.sub(' [;] ','; ',text)
      text=re.sub(' [*] ','* ',text)
      text=re.sub(' [\'] ','\'',text)
      text=re.sub(' [\"] ','\"',text)
      return text

   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()
```

#ENCODER..



#DECODER..


input_4 | input: | [(None, None)]
InputLayer | output: | [(None, None)]
decoder_embedding | input: | (None, None)
Embedding | output: | (None, None, 256)
layer_normalization | input: | (None, None, 256)
LayerNormalization | output: | (None, None, 256)
input_2 | input: | [(None, 256)]
InputLayer | output: | [(None, 256)]
input_3 | input: | [(None, 256)]
InputLayer | output: | [(None, 256)]
decoder_lstm | input: | [(None, None, 256), (None, 256), (None, 256)]
LSTM | tanh | output: | [(None, None, 256), (None, 256), (None, 256)]
decoder_dense | input: | (None, None, 256)
Dense | softmax | output: | (None, None, 2443)

5. Testing and Debugging :

Testing is essential to ensure your chatbot functions as intended. Test the chatbot by engaging in conversations and verifying that it responds correctly to various inputs. Debug and refine the chatbot as needed.

Testing and debugging are crucial stages in chatbot development using Python. Testing ensures your chatbot functions correctly and provides accurate responses, while debugging helps identify and fix issues.

- Testing includes unit tests for individual components, integration tests for overall functionality, user testing to gather real-world feedback, automation tests for web interfaces, regression tests for changes, and load tests for performance.
- Debugging involves using logging, print statements, interactive debuggers, and error handling to identify and resolve issues. Code reviews, testing with sample data, and version control also play key roles in debugging.

These processes are essential for delivering a reliable and user-friendly chatbot.

6. Deployment :

Once your chatbot is working as expected, you can deploy it. Deployment can vary depending on your chatbot's use case. You might deploy it as a web service, integrate it into a mobile app, or use it on a chat platform like Slack or Facebook Messenger.Time to Chat :

```
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.

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

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

You: Don't ever be in a hurry

Bot: it's not a great.

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

You: I'm gonna put some dirt in your eye

Bot: that's a good idea.
```

```
=========================

You: You're trash

Bot: the tv news is reporting a bank robbery.

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

You: I've read all your research on nano-technology

Bot: it's the weather. i've gone around the world.

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

You: You want forgiveness? Get religion

Bot: no, i'll be my.

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

You: While you're using the bathroom, i'll order some food.

Bot: don't order for me. i've been a cheater.

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

You: Wow! that's terrible.

Bot: never park your car under the house.

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

You: We'll be here forever.

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

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


You: I need something that's reliable.

Bot: you need a car with low mileage.

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

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

Bot: what happened?

=========================
```

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

Bot: i'll make a sandwich.

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

You: I like this restaurant because they give you free bread.

Bot: well, i think that's a good idea.

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

7. Conclusion :

- In conclusion, creating a chatbot using Python for your project is a rewarding and versatile endeavor. Chatbots have the potential to revolutionize various domains, from customer support to automation of routine tasks. By following a structured approach, including design, implementation, testing, and debugging, you can develop a chatbot that is not only functional but also user-friendly.

- Throughout the development process, it's essential to keep your project's goals and objectives in mind. A well-designed chatbot that aligns with the intended purpose and user expectations will have a higher chance of success. Regular testing and debugging are crucial to ensure that your chatbot operates smoothly and responds accurately to user inputs.

- As you embark on your chatbot project, stay open to user feedback, continuously improve your chatbot's capabilities, and keep up with the latest advancements in natural language processing and chatbot technologies. With dedication and a clear vision, you can create a chatbot that adds value to your project and provides a seamless user experience.

Creating a chatbot using Python can be a rewarding project. Designing, implementing, testing, and deploying your chatbot allows you to provide automated, responsive interactions with users.