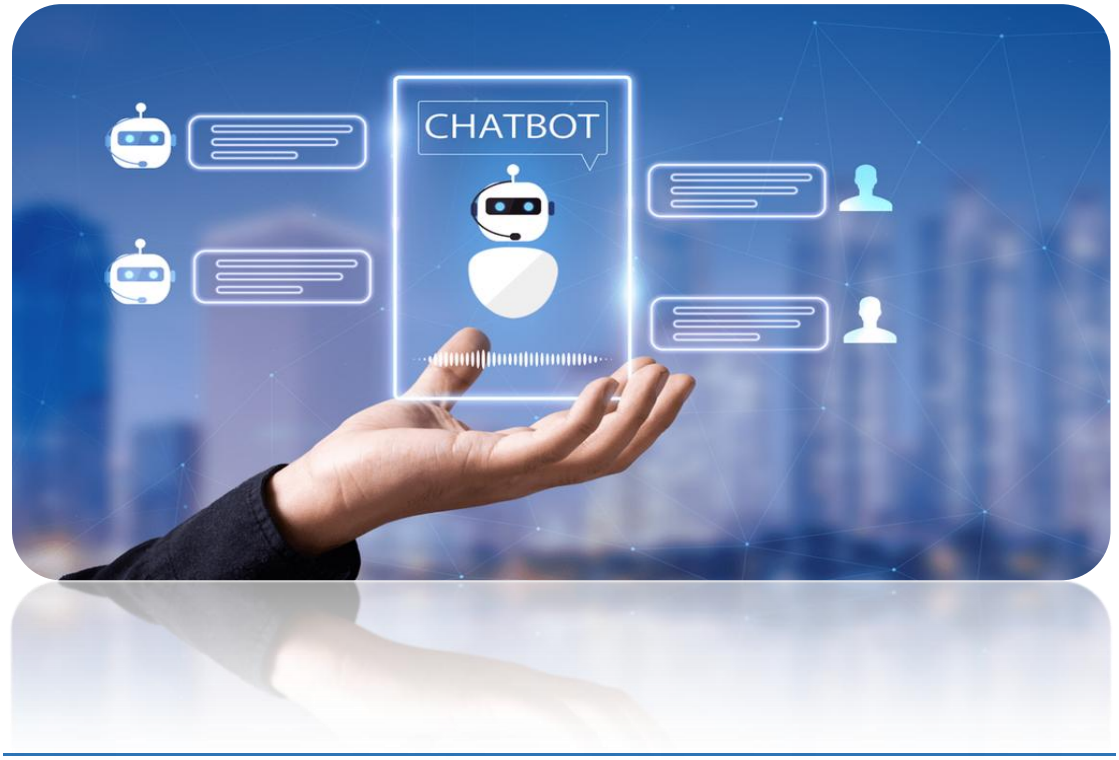
CREATE A CHATBOT IN PYTHON

TEAM MEMBER: Sinaz Afrin .H

Phase-1: Document Submission



Project: *ChatBot in Python*

Abstract:

ChatBots have gained significant popularity in various industries, offering efficient and automated ways to engage with users, answer questions, and perform tasks. Creating a chatbot in Python requires a structured approach, leveraging different modules and tools to build an effective conversational agent. This abstract presents a high-level overview of the essential modules and steps involved in developing a chatbot in Python.

Data Source:

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

1.Functionality:

The chatbot's scope is meticulously defined, encompassing its core abilities such as answering common questions, offering guidance, and directing users to relevant resources. This initial phase sets the foundation for the chatbot's purpose and functionality.

2.User Interface:

The choice of integration, whether it be within a website or a mobile application, is carefully considered. The design of an intuitive and user-friendly interface is paramount to ensure seamless interactions with users.

3.Natural Language Processing (NLP):

NLP techniques are integrated into the chatbot's architecture to enable the understanding and processing of user input in a conversational manner. NLP empowers the chatbot to comprehend and respond effectively to diverse user queries and prompts.

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.

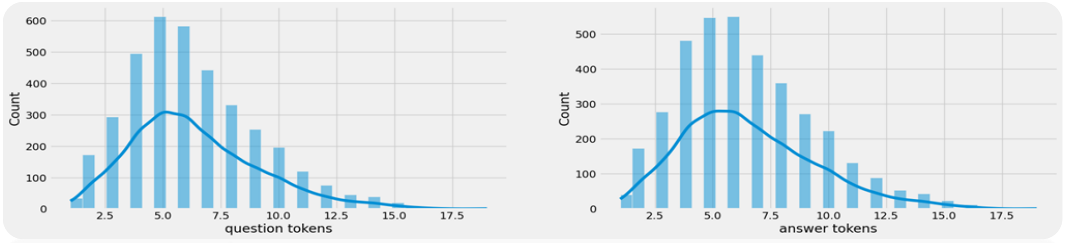
4.Responses:

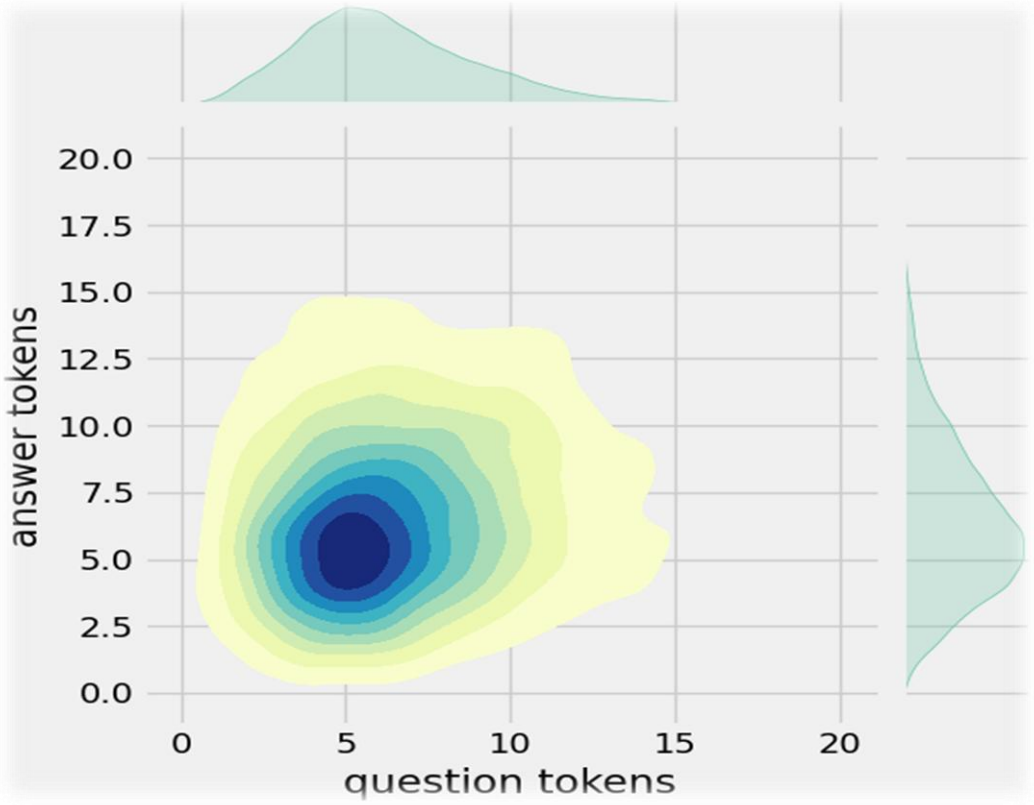
Planning and crafting responses remain central, ensuring that the chatbot provides accurate answers, relevant suggestions, and valuable assistance to users. Responses must align with the chatbot's objectives and user expectations.

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()
```





5.Integration:

Decisions are made regarding how the chatbot will seamlessly integrate with the chosen platform, including technical aspects such as APIs, SDKs, and integration tools. This ensures a cohesive user experience.

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>

6.Testing and Improvement:

An iterative cycle of testing and refinement is instituted to enhance the chatbot's performance based on real user interactions. Feedback from users is invaluable for making data-driven improvements to the chatbot's functionality, responses, and underlying models.

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:

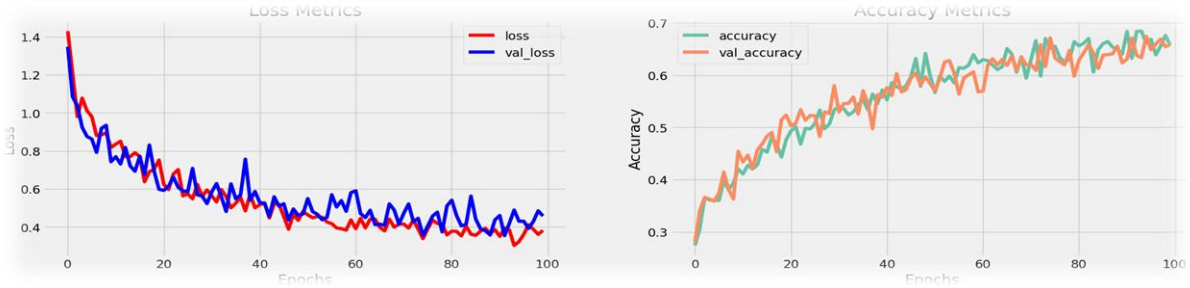
```
class ChatBotTrainer(tf.keras.models.Model):
    def __init__(self,encoder,decoder,*args,**kwargs):
        super().__init__(*args,**kwargs)
        self.encoder=encoder
        self.decoder=decoder
    def loss_fn(self,y_true,y_pred):
```

```
        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']
)
model(_[:2])
```

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()
```



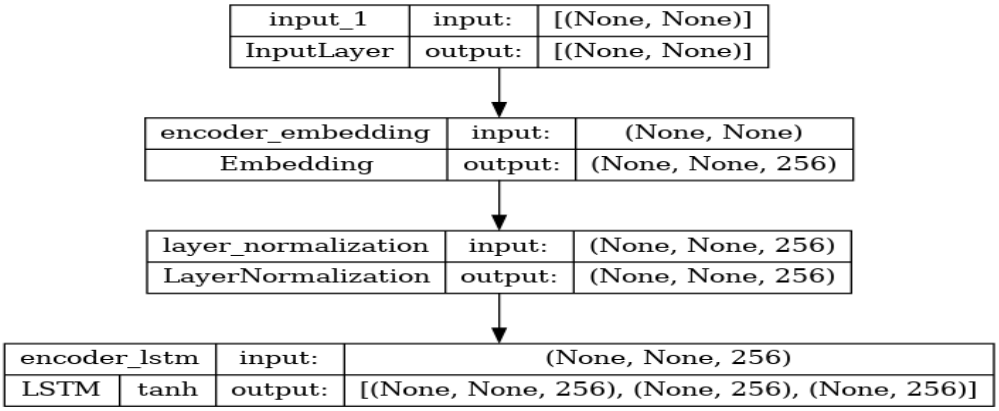
Save Model:

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

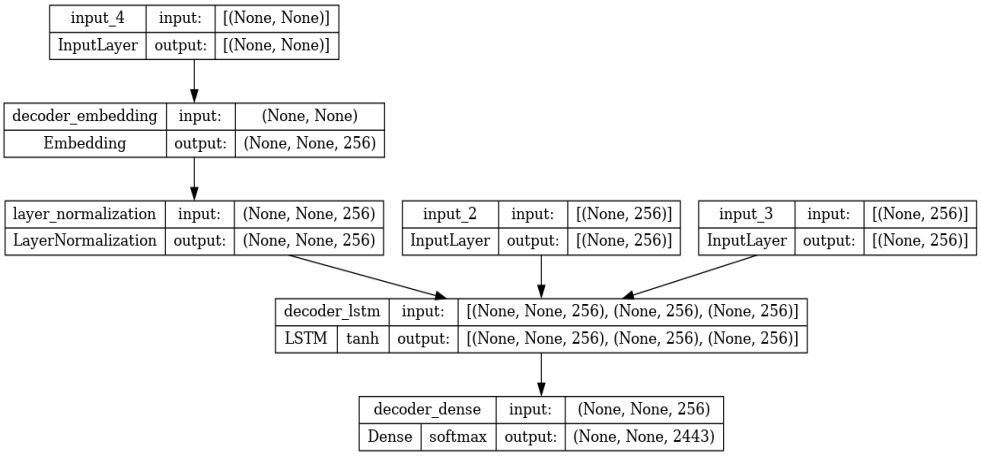
Create Inference Model:

```
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_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)
  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)
        return text
chatbot=ChatBot(model.encoder,model.decoder,name='chatbot')
chatbot.summary()
tf.keras.utils.plot_model(chatbot.encoder,to_file='encoder.png',show_shapes=True,show_layer_activ
ations=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)}')
```

7.Conclusion:

In conclusion, the development of a Python-based chatbot aligns with the principles of Design Thinking and encompasses various facets, including functionality, user interface, NLP implementation, data processing, feature selection, model selection, model training, evaluation metrics, integration, and continuous testing and improvement. This holistic approach ensures the creation of an intelligent and effective chatbot that meets user needs and expectations.