3. Response Generation:

process user input in a way that feels natural and intuitive.

2. User Input Handling: Implement a mechanism for the chatbot to receive, understand, and

Develop a chatbot using Python that can engage in text-based conversations with users.

1.Chatbot Framework:

Requirements:

8. Testing And Deployment.

7. Dialogue Flow Implementation.

6. Select Development Platform.

5. Data Collection.

4. Choose Apps For Integration.

3. Design Conversational Language And Architecture.

2. Decide A Communication Channel.

1. Define Goals For Your Chatbot.

**Let's take a quick look at these steps.**

across different domains.

meaningful conversations and assist users

versatile chatbot that can engage in

entertainment. The objective is to create a

support to information retrieval and

a wide range of purposes, from customer

Organizations and individuals use chatbots for

integral part of online communication.

In the digital age, chatbots have become an

Background:

Problem Statement: Building a General-Purpose Chatbot

CREATE A CHATBOT USING PYTHON

PROJECT TITLE:

ARTIFICIAL INTELLIGENCE-GROUP 1

211321104009 : S.S.SOWMYA

NAAN MUDHALVAN-IBM(AI) PROJECT



Evaluation Criteria: The success of the project will be evaluated based on:

messaging app, etc.)

•

Consider the limitations of the chosen platform for deployment (e.g., web server,

•

Ensure that the chatbot respects data privacy and adheres to any relevant regulations.

additional languages can be a future enhancement.

•

The chatbot's primary language for interaction should be English, but support for

Constraints:

•

Deployment instructions for putting the chatbot into production.

•

Test reports demonstrating the chatbot's performance in different scenarios.

•

Documentation detailing how to use, maintain, and extend the chatbot.

•

A functional chatbot built in Python, meeting the specified requirements.

Deliverables:

app, or custom application.

9.Deployment: Deploy the chatbot on a suitable platform, whether it's a website, messaging

responses in various conversation scenarios.

Conduct comprehensive testing to ensure the chatbot performs well and provides meaningful

8.Testing:

external data sources.

Make the chatbot extensible, allowing for easy integration with additional functionality or

7.Extensibility:

understand the user's input or encounters unexpected issues.

Implement robust error handling to gracefully handle situations where the chatbot doesn't

6.Error Handling:

includes appropriate greetings, farewells, and handling of user queries or requests.

Design the chatbot to provide a user-friendly and engaging conversational experience. This

5.User Interaction:

be able to switch between different conversation topics seamlessly.

Ensure that the chatbot can handle conversations on a variety of topics or domains. It should

4.Multi-domain Capability:

statements. Responses should make sense in the context of the conversation.

Train the chatbot to generate contextually relevant and coherent responses to user queries or



responses = ["Hello!", "Hi there!", "Hey!", "How can I help you today?"]

greetings = ["hello", "hi", "hey", "howdy"]

# Define a list of greetings and responses

import random

python

example of how to create a basic chatbot using Python:

Creating a chatbot in Python can be a fun and educational project. Here's a simple

Simple step:

•

Performance and reliability in a production environment.

•

Extensibility and potential for integration with external systems.

•

Robust error handling and graceful degradation during issues.

•

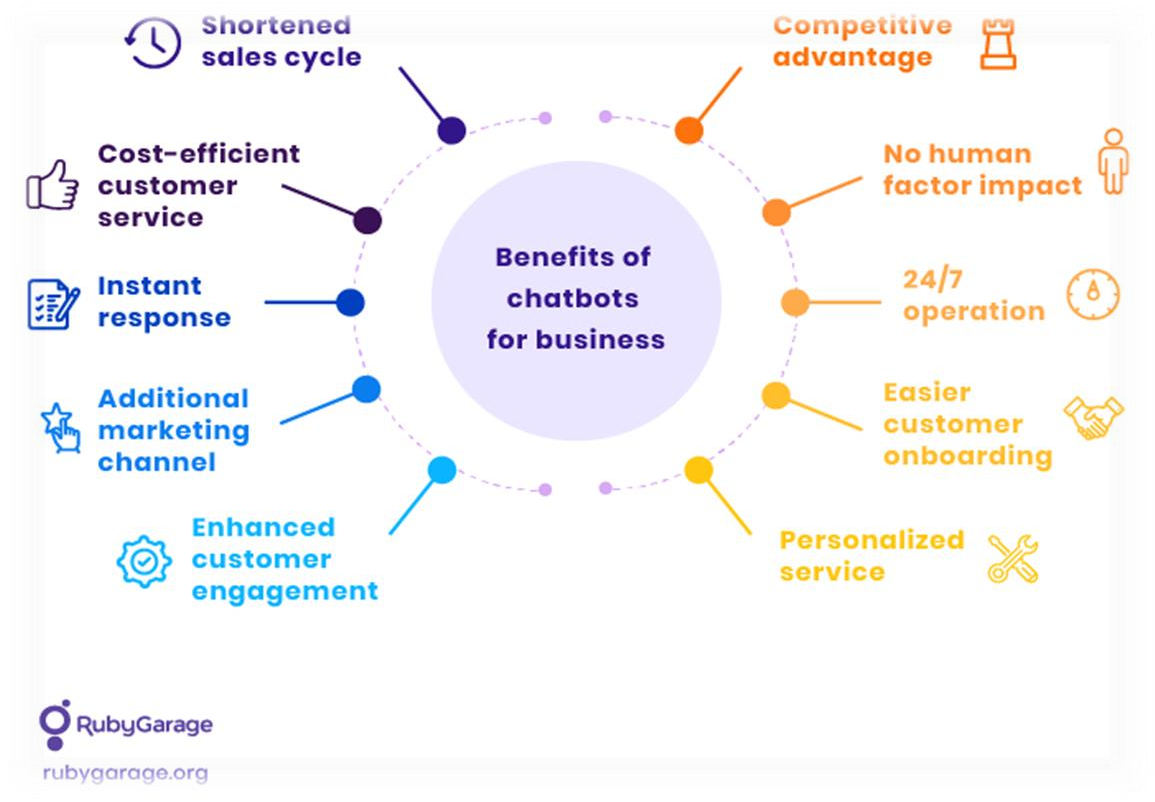
User-friendliness and engagement of the chatbot's interface.

•

Versatility in handling conversations across different domains.

•

The chatbot's ability to engage in meaningful and coherent conversations.



and make your chatbot stand out:

problem-solving. Let's explore innovative design elements to solve common problems

be innovative and effective by incorporating cutting-edge technologies and creative

Designing a chatbot using Python can

Innovative Design for Creating a Python Chatbot

specific project goals and constraints.

provides a foundation for creating a versatile chatbot, and you can adapt it to meet your

problem statement to address more specialized requirements. This problem statement

Depending on your specific use case or industry, you may need to tailor this

IMPORTANT NOTES:

tasks like weather information or news updates.

for more advanced interactions. Additionally, you could use external APIs for more specific

of user input, and even integrating natural language processing libraries like NLTK or spacy

You can expand and improve this chatbot by adding more responses, handling different types

will continue the conversation until you type "bye."

This basic chatbot recognizes a few simple greetings and responds with random replies. It

print("Chatbot:", response)

response = chatbot\_response(user\_input)

break

print("Chatbot: Goodbye!")

if user\_input.lower() == "bye":

user\_input = input("You: ")

while True:

# Main loop for the chatbot

return "I'm just a simple chatbot. I don't understand that."

else:

return random.choice(responses)

if user\_input in greetings:

user\_input = user\_input.lower()

def chatbot\_response(user\_input):

# Define a function to generate a response



2.Multimodal Interaction:

understand and generate human-like responses with context and coherence.

Utilize state-of-the-art NLP models like GPT-3 or BERT to enable your chatbot to

1.Advanced Natural Language Processing (NLP):

16. AI Chatbot Marketplaces

15. Ethical AI

14. Quantum Computing for Speed

13. Blockchain for Data Security

12. Augmented Reality (AR) Integration

11. Voice Biometrics

10. Continuous Learning

9. Voice Synthesis

8. Multilingual Support

7. IoT Integration

6. Predictive Typing

5. Contextual Understanding

4. Personalization

3. Emotion Recognition

2. Multimodal Interaction

1. Advanced Natural Language Processing (NLP)



5.Contextual Understanding:

products, content, or services tailored to individual users.

behavior, preferences, and historical interactions. Consider recommending

Use machine learning to personalize the chatbot's responses based on user

4.Personalization:

The chatbot can adapt its responses to provide empathy or assistance accordingly.

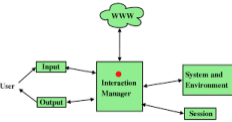
recognition algorithms to gauge the user's emotional state based on text input or voice tone.

Implement sentiment analysis and emotion

3.Emotion Recognition:

expands its capabilities to assist users with a wide range of queries and interaction modes.

Innovate by allowing the chatbot to process text, images, voice, and even gestures. This



8.Multilingual Support:

Things (IoT) devices, such as smart home appliances, through voice or text commands.

Extend the chatbot's functionality by enabling it to control and interact with Internet of

7.IoT Integration:

experience.

assist users in formulating queries faster and with greater accuracy, improving user

Implement predictive typing suggestions using machine learning models.

This feature can

6. Predictive Typing:

understood.

This enables more meaningful and coherent conversations over time, making users feel

Enhance the chatbot's contextual awareness by storing and recalling previous interactions.



11.Voice Biometrics:

its responses over time based on user feedback and interactions.

Implement reinforcement learning algorithms to allow your chatbot to learn and improve

10.Continuous Learning:

enhancing the quality of voice interactions and user engagement.

Develop a natural-sounding voice for your chatbot using text to-speech (TTS) synthesis,

9.Voice Synthesis:

and translation features to facilitate seamless communication in different languages.

Make your chatbot multilingual to cater to a

global audience. Implement language detection



interactions, instilling trust in users.

Explore blockchain technology to ensure the security and integrity of user data and chatbot

13.Blockchain for Data Security:

to provide visual assistance, such as overlaying instructions on a user's camera feed.

For mobile chatbots, consider integrating AR features

12.Augmented Reality (AR) Integration:

such as account access or transactions.

Enhance security by incorporating voice biometric authentication for sensitive interactions,



User Interface (UI): The user interface is how users interact with the chatbot. This can be a

components:

the specific requirements and capabilities of the chatbot, but here are some common

communication between the bot and users. These blocks can vary in complexity depending on

Chatbots are built using various components or blocks that work together to enable

BLOCKS OF CHATBOT

problems but also offers a truly exceptional user experience.

your Python chatbot design, you can create a solution that not only addresses specific

By integrating these detailed innovative elements into

fostering a community of innovation.

their chatbot with AI plugins, allowing for greater customization and functionality,

Create a marketplace where users can enhance

16.AI Chatbot Marketplaces:

follows responsible AI practices to gain user trust and compliance with regulations.

Ensure that your chatbot adheres to ethical AI principles, respects user privacy, and

15. Ethical AI:

processing and decision-making even faster and more efficient.

In the future, consider harnessing the power of quantum computing to make real-time

14. Quantum Computing for Speed:



the chatbot is functioning correctly and can trigger alerts for issues.

including user engagement, error rates, and frequently asked questions. Monitoring ensures

Analytics and Monitoring: Analytics tools are used to track the chatbot's performance,

enhance performance.

over time. Testing helps identify issues, while training involves updating models and data to

Testing and Training: Continuous testing and training are essential for chatbots to improve

authentication and authorization to ensure security and privacy.

If the chatbot interacts with user accounts or sensitive data, it needs mechanisms for user

User Authentication and Authorization:

reservations, retrieving weather information, or accessing user accounts.

user requests. Integration with APIs allows the chatbot to perform actions like making

API Integration: Chatbots often need to interact with external systems or services to fulfill

systems, or more advanced natural language generation techniques.

information, it generates a response. This can be done using pre-defined templates, rule-based

Response Generation: Once the chatbot understands the user's intent and extracts relevant

that involve structured data.

input, such as names, dates, locations, or product names. This is crucial for handling requests

Entity Recognition: Entity recognition identifies specific pieces of information within user

accordingly.

goal behind a message. It helps the chatbot understand what the user wants and respond

Intent Recognition: Intent recognition is a subset of NLP that determines the user's intent or

predictions. Common models include Recurrent Neural Networks (RNNs) and Transformers.

used to improve the chatbot's language understanding, generate responses, and make

Machine Learning Models: Machine learning models, such as deep learning models, can be

chatbot accesses this information to answer user queries.

This knowledge base can be a structured database or unstructured text documents. The

Knowledge Base: Some chatbots rely on a knowledge base to provide information to users.

learning models.

history. Dialog management can be rule-based, state-machine-based, or based on machine

determines how the chatbot responds to user inputs and manages context and conversation

Dialog Management: Dialog management handles the flow of the conversation. It

tokenization, entity recognition, sentiment analysis, and language understanding.

to understand and process natural language inputs from users. NLP involves tasks like text

Natural Language Processing (NLP): NLP is a crucial component that enables the chatbot

chat window on a website, a messaging app, or a voice interface in a smart device.

+--------------------------------------+

|

Natural Language Processing |

+--------------------------------------+

v

|

| (User Input)

+--------------------------------------+

|

User Interface

|

+--------------------------------------+

Here's a brief description of each block:

a chatbot system. Below is a simplified block diagram of a chatbot:

Achatbot block diagram illustrates the various components and their interactions within

BLOCK DIAGRAM

and the technologies used in its development.

components and their complexity can vary depending on the chatbot's purpose, complexity,

These are some of the fundamental blocks that make up a chatbot system. The specific

performance. Updates may involve adding new features or improving existing ones.

Chatbots require ongoing maintenance to address issues, update knowledge, and improve

Maintenance and Updates:

can include surveys, rating systems, or direct user input.

Collecting feedback from users is valuable for improving the chatbot. Feedback mechanisms

User Feedback Mechanisms:

considerations.

or cloud platform to make it accessible to users. This involves hosting and infrastructure

Deployment and Hosting: Once the chatbot is developed, it needs to be deployed on a server



|

User Interface

|

+--------------------------------------+

v

|

| (Generated Response)

+--------------------------------------+

|

Response Generation

|

+--------------------------------------+

v

|

| (Information)

+--------------------------------------+

|

External APIs

|

|

Knowledge Base /

|

+--------------------------------------+

v

|

| (Context, State)

+--------------------------------------+

|

Dialog Management

|

+--------------------------------------+

v

|

| (Intent, Entities)

+--------------------------------------+

|

Entity Recognition

|

|

Intent Recognition &

|

+--------------------------------------+

v

|

| (Extracted Text)

DEVELOPMENT PART:

users can easily type their query in natural language and retrieve information.

for proper keywords to look up in a search or browse several web pages to collect information;

A chatbot is one of the simple ways to transport data from a computer without having to think

CONCLUSION

requirements.

implementation details within each block can vary based on the chatbot's design and

complex, with additional components for authentication, analytics, and more. Additionally, the

Note: This is a simplified representation, and real-world chatbot architectures can be more

User Interface: The final response is presented to the user through the same user interface.

extracted entities, and the chatbot's knowledge.

Response Generation: This block generates a response based on the recognized intent,

services to retrieve information needed for responses.

Knowledge Base / External APIs: The chatbot may access a knowledge base or external

and decides how the chatbot should respond.

Dialog Management: Dialog management handles the conversation flow, maintains context,

and extract entities from the input.

Intent Recognition & Entity Recognition: These components determine the user's intent

tokenization, sentiment analysis, and language understanding.

Natural Language Processing (NLP): NLP processes the user's input, including

voice input.

User Interface: This is where users interact with the chatbot, sending messages or

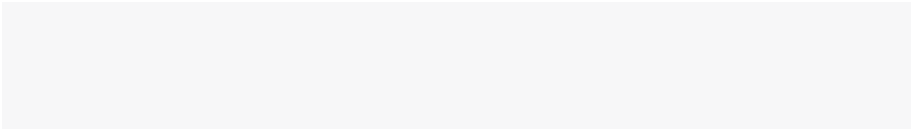
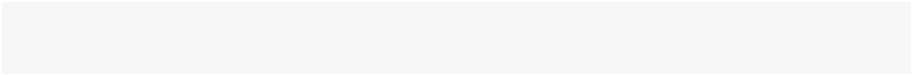
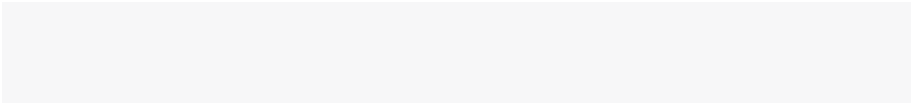
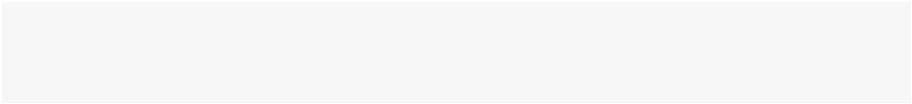
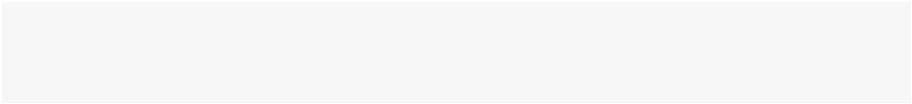
+--------------------------------------+

v

|

| (Bot Response)

+--------------------------------------+



components from TensorFlow's Keras module. It also imports the re and string modules for

This code snippet imports TensorFlow, NumPy, Pandas, Matplotlib, Seaborn, and various

**I.** Import Libraries:

 Time to Chat

 Create Inference Model

 Save Model

 Visualize Metrics

3. Train Model

2. Build Training Model

1. Build Encoder

 Build Models

3.

Tokenization

2.

Text Cleaning

1.

Data Visualization

 Data Preprocessing

 Import Libraries

3.Table of Contents:

datasets.

 Another vital part of the chatbot development process is creating the training and testing

store the python objects that are used for predicting the responses of the bot.

lemmatization that transforms a word into its lemma form. Then it creates a pickle file to

into smaller, readable chunks (like words). Once that is done, you can also go for

 This is where tokenizing helps with text data – it helps fragment the large text dataset

to perform data preprocessing on your dataset before designing an ML model.

variables you want to use in chatbot project. Also, when working with text data, we need

 To build a chatbot in Python, import all the necessary packages and initialize the

leveraging this nifty tool to drive business benefits.

adopting them. From e-commerce firms to healthcare institutions, everyone seems to be

and conversing with humans, that companies across various industrial sectors are

business sectors. These intelligent bots are so adept at imitating natural human languages

 In the past few years, chatbots in Python have become wildly popular in the tech and

2.How to Make a Chatbot in Python?

processing.

like manner. A chatbot is arguably one of the best applications of natural language

can effortlessly mimic human languages to communicate with human beings in a human-

languages. These chatbots are usually converse via auditory or textual methods, and they

 A chatbot is an AI-based software designed to interact with humans in their natural

1.What is a Chatbot?



sns.histplot(x=df['question tokens'], data=df, kde=True, ax=ax[0])

sns.set\_palette('Set2')

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(20, 5))

plt.style.use('fivethirtyeight')

import seaborn as sns

import matplotlib.pyplot as plt

df['answer tokens'] = df['answer'].apply(lambda x: len(x.split()))

df['question tokens'] = df['question'].apply(lambda x: len(x.split()))

Input 3:

and a joint distribution between 'question' and 'answer' tokens.

The resulting plots are displayed in a single figure with two subplots for token distributions

Pandas DataFrame and then visualizes the token distribution using Matplotlib and Seaborn.

This code calculates the number of tokens (words) in the 'question' and 'answer' columns of a

 Data Visualization:

**II.** Data Preprocessing:

from tensorflow.keras.layers import LSTM,Dense,Embedding,Dropout,LayerNormalization

import re,string

from tensorflow.keras.layers import TextVectorization

import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import tensorflow as tf

Input 1-2:

working with deep learning and natural language processing.

regular expressions and string manipulation. The code prepares your environment for

text = re.sub('[1]', ' 1 ', text)

text = re.sub('[.]', ' . ', text)

text = re.sub('-', ' ', text.lower())

def clean\_text(text):

Input 4:

'encoder\_inputs', 'decoder\_targets', and 'decoder\_inputs' columns.

'question' and 'answer' columns in the DataFrame. It also modifies the DataFrame by creating

This code defines a clean\_text function to clean the text and then applies this function to the

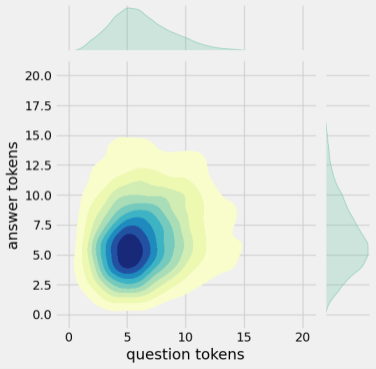
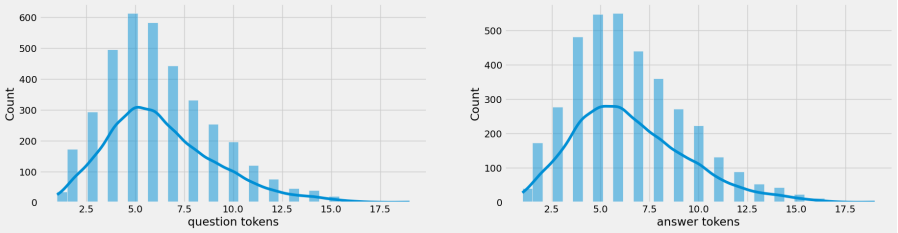
 Text Cleaning:

plt.show()

cmap='YlGnBu')

sns.jointplot(x='question tokens', y='answer tokens', data=df, kind='kde', fill=True,

sns.histplot(x=df['answer tokens'], data=df, kde=True, ax=ax[1])



df.head(10)

df['decoder\_inputs'] = '<start> ' + df['answer'].apply(clean\_text) + ' <end>'

df['decoder\_targets'] = df['answer'].apply(clean\_text) + ' <end>'

df['encoder\_inputs'] = df['question'].apply(clean\_text)

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

return text

text = re.sub('\t', ' ', 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('[0]', ' 0 ', text)

text = re.sub('[9]', ' 9 ', text)

text = re.sub('[8]', ' 8 ', text)

text = re.sub('[7]', ' 7 ', text)

text = re.sub('[6]', ' 6 ', text)

text = re.sub('[5]', ' 5 ', text)

text = re.sub('[4]', ' 4 ', text)

text = re.sub('[3]', ' 3 ', text)

text = re.sub('[2]', ' 2 ', text)

campus.

campus .

school . <end>

school.

school . <end>

9

really big

s a really big

luck with

good luck with

good luck with

it's okay. it's a

it ' s okay . it '

<start> good

campus.

campus . <...

really big cam...

there?

there ?

8

really big

a really big

okay . it ' s a

do you like it

do you like it

it's okay. it's a

it ' s okay . it ' s

<start> it ' s

<end>

there?

there ? <end>

7 i go to pcc.

i go to pcc .

like it there ?

do you like it

do you like it

<start> do you

you go to?

you go to ?

<end>

pcc . <end>

6

i go to pcc.

what school do

what school do

i go to pcc .

<start> i go to

now .

right now.

<end>

go to ? <end>

you go to?

school right

5

i'm in school

you go to ?

school do you

what school do

good . i ' m in

i've been good.

what school do

<start> what

i ' ve been

now ...

you?

right now.

about you ?

m in school ri...

school right

4

what about

i'm in school

great . what

been good . i '

good . i ' m in

i've been great.

i've been good.

i ' ve been

<start> i ' ve

i ' ve been

<end>

? ...

been?

you?

you been ?

about you ?

what about you

3

how have you

what about

so how have

great . what

been great .

no problem. so

i've been great.

no problem .

i ' ve been

<start> i ' ve

been ? ...

for asking.

been?

for asking .

been ? <end>

how have you

2

good. thanks

how have you

good . thanks

how have you

problem . so

i'm pretty

no problem. so

i ' m pretty

no problem . so

<start> no

<end>

asking...

yourself?

for asking.

?

for asking .

thanks for

1

about

good. thanks

about yourself

good . thanks

pretty good .

i'm fine. how

i'm pretty

i ' m fine . how

i ' m pretty

<start> i ' m

<end>

yourself?

? <end>

you doing?

you doing ?

about yourself ?

0

about

about yourself

hi, how are

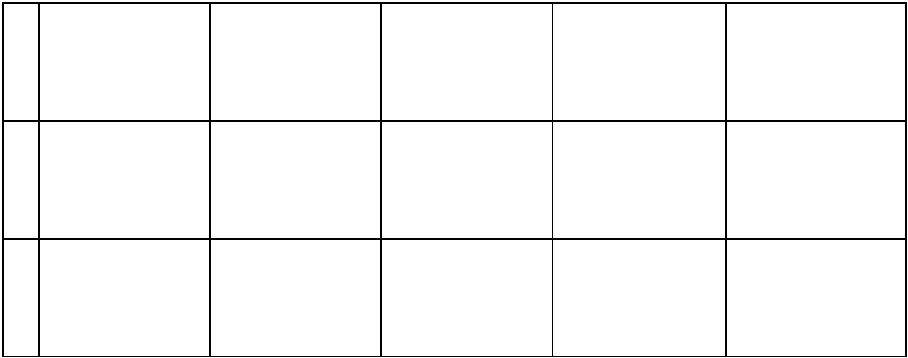
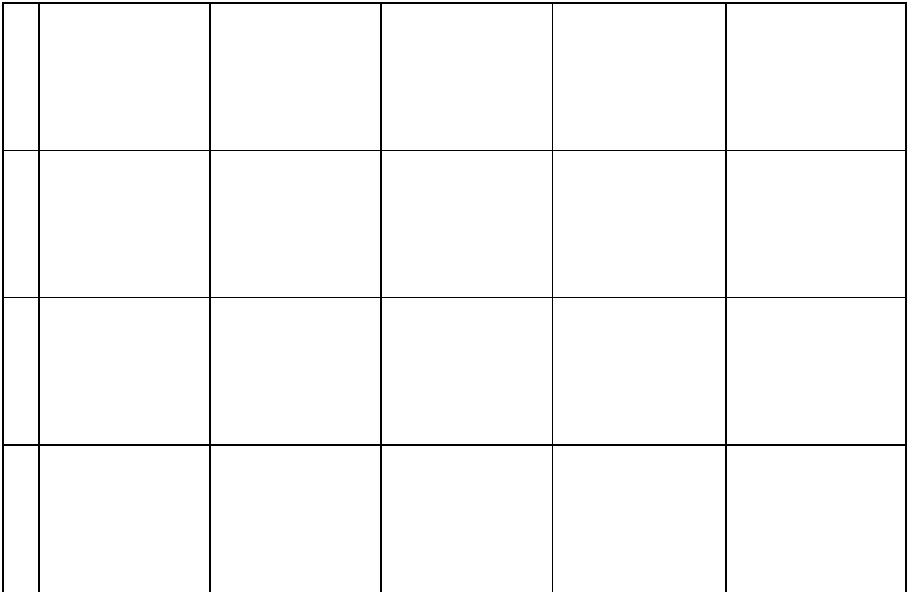
hi , how are

fine . how

i'm fine. how

i ' m fine . how

<start> i ' m



'decoder target tokens'.

subplots for the token counts and a joint distribution between 'encoder input tokens' and

Matplotlib and Seaborn. The resulting plots are displayed in a single figure with three

'decoder\_targets' columns in the DataFrame and then visualizes the token distribution using

This code calculates the token counts for 'encoder\_inputs', 'decoder\_inputs', and

plt.show()

fill=True, cmap='YlGnBu')

sns.jointplot(x='encoder input tokens', y='decoder target tokens', data=df, kind='kde',

sns.histplot(x=df['decoder target tokens'], data=df, kde=True, ax=ax[2])

sns.histplot(x=df['decoder input tokens'], data=df, kde=True, ax=ax[1])

sns.histplot(x=df['encoder input tokens'], data=df, kde=True, ax=ax[0])

sns.set\_palette('Set2')

fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(20, 5))

plt.style.use('fivethirtyeight')

import seaborn as sns

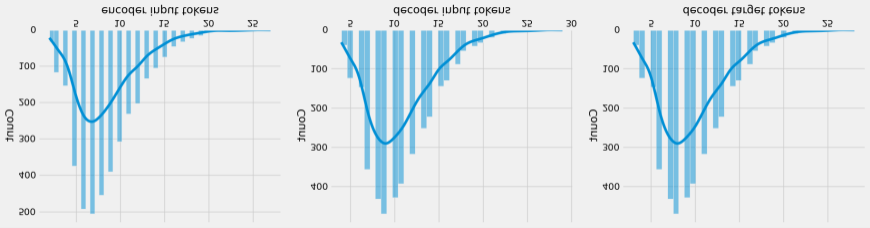
import matplotlib.pyplot as plt

df['decoder target tokens'] = df['decoder\_targets'].apply(lambda x: len(x.split()))

df['decoder input tokens'] = df['decoder\_inputs'].apply(lambda x: len(x.split()))

df['encoder input tokens'] = df['encoder\_inputs'].apply(lambda x: len(x.split()))

Input 5:



lstm\_cells=params['lstm\_cells']

embedding\_dim=params['embedding\_dim']

batch\_size=params['batch\_size']

learning\_rate=params['learning\_rate']

}

"buffer\_size":10000

"embedding\_dim":256,

"lstm\_cells":256,

"batch\_size":149,

"learning\_rate":0.008,

"max\_sequence\_length":30,

"vocab\_size":2500,

params={

target tokens'],axis=1,inplace=True)

df.drop(columns=['question','answer','encoder input tokens','decoder input tokens','decoder

print(f"Max decoder target length: {df['decoder target tokens'].max()}")

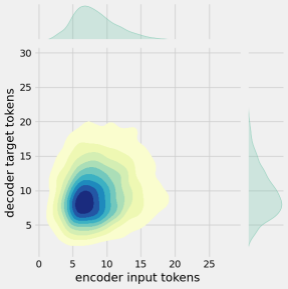
print(f"Max decoder input length: {df['decoder input tokens'].max()}")

print(f"Max encoder input length: {df['encoder input tokens'].max()}")

tokens']]['encoder\_inputs'].values.tolist())}")

print(f"After preprocessing: {' '.join(df[df['encoder input tokens'].max()==df['encoder input

Input 6:



various details about the data, such as batch sizes and shapes.

creation of training and validation datasets using TensorFlow's Dataset API. It also prints

TensorFlow's TextVectorization layer, conversion between sequences and IDs, and the

This code snippet involves data preprocessing, including text vectorization using

 Tokenization:

big campus .

<end>

school . <end>

9

it ' s okay . it ' s a really

good luck with school .

<start> good luck with

big campus . <...

a really big cam...

8

do you like it there ?

it ' s okay . it ' s a really

<start> it ' s okay . it ' s

<end>

there ? <end>

7

i go to pcc .

do you like it there ?

<start> do you like it

?

<end>

6

i go to pcc . <end>

what school do you go to

<start> i go to pcc .

school right now .

? <end>

you go to ? <end>

5

i ' ve been good . i ' m in

what school do you go to

<start> what school do

about you ?

school right now ...

. i ' m in school ri...

4

i ' ve been great . what

i ' ve been good . i ' m in

<start> i ' ve been good

you been ?

about you ? <end>

. what about you ? ...

3

no problem . so how have

i ' ve been great . what

<start> i ' ve been great

...

for asking .

you been ? <end>

2

how have you been ?

i ' m pretty good . thanks

no problem . so how have

<start> no problem . so

asking...

yourself ?

for asking . <end>

1

good . thanks for

i ' m fine . how about

i ' m pretty good . thanks

<start> i ' m pretty

yourself ? <end>

about yourself ? <end>

0

hi , how are you doing ?

i ' m fine . how about

<start> i ' m fine . how

encoder\_inputs

decoder\_targets

decoder\_inputs

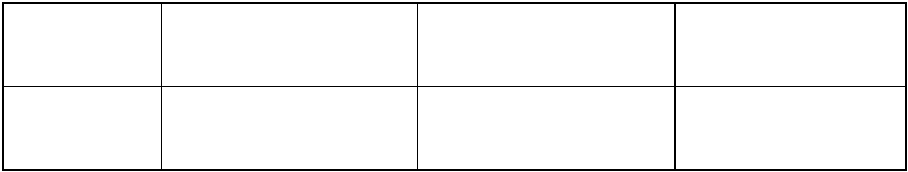
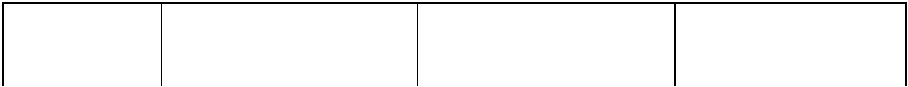
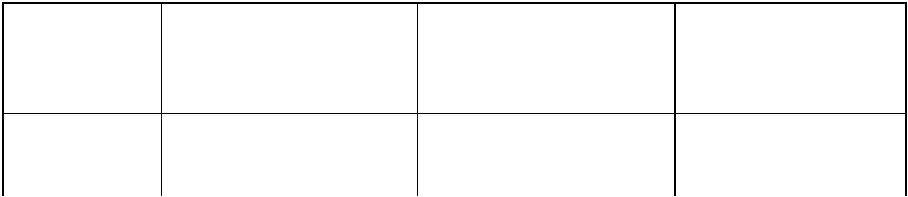
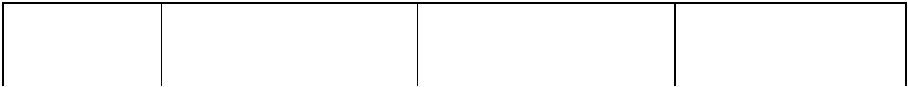
Output:

df.head(10)

max\_sequence\_length=params['max\_sequence\_length']

buffer\_size=params['buffer\_size']

vocab\_size=params['vocab\_size']



y=sequences2ids(df['decoder\_targets'])

yd=sequences2ids(df['decoder\_inputs'])

x=sequences2ids(df['encoder\_inputs'])

return decode

decode+=vectorize\_layer.get\_vocabulary()[id]+' '

for id in ids:

ids=[ids]

if type(ids)==int:

decode=''

def ids2sequences(ids):

return vectorize\_layer(sequence)

def sequences2ids(sequence):

Input 8:

['', '[UNK]', '<end>', '.', '<start>', "'", 'i', '?', 'you', ',', 'the', 'to']

Vocab size: 2443

print(f'{vectorize\_layer.get\_vocabulary()[:12]}')

print(f'Vocab size: {len(vectorize\_layer.get\_vocabulary())}')

vocab\_size=len(vectorize\_layer.get\_vocabulary())

vectorize\_layer.adapt(df['encoder\_inputs']+' '+df['decoder\_targets']+' <start> <end>')

)

output\_sequence\_length=max\_sequence\_length

output\_mode='int',

standardize=None,

max\_tokens=vocab\_size,

vectorize\_layer=TextVectorization(

Input 7:

train\_data=train\_data.batch(batch\_size)

train\_data=train\_data.shuffle(buffer\_size)

train\_data=train\_data.cache()

train\_data=data.take(int(.9\*len(data)))

data=data.shuffle(buffer\_size)

data=tf.data.Dataset.from\_tensor\_slices((x,yd,y))

Input 10:

Decoder target: [ 6 5 38 646 3 45 41 563 7 2 0 0] …

Decoder input: [ 4 6 5 38 646 3 45 41 563 7 2 0] ...

Encoder input: [1971 9 45 24 8 194 7 0 0 0 0 0] ...

print(f'Decoder target: {y[0][:12]} ...')

decoder is the output of the previous timestep

print(f'Decoder input: {yd[0][:12]} ...') # shifted by one time step of the target as input to

print(f'Encoder input: {x[0][:12]} ...')

Input 9:

Decoder target shape: (3725, 30)

Decoder input shape: (3725, 30)

Encoder input shape: (3725, 30)

Question to tokens: [1971 9 45 24 8 7 0 0 0 0]

Question sentence: hi , how are you ?

print(f'Decoder target shape: {y.shape}')

print(f'Decoder input shape: {yd.shape}')

print(f'Encoder input shape: {x.shape}')

print(f'Question to tokens: {sequences2ids("hi , how are you ?")[:10]}')

print(f'Question sentence: hi , how are you ?')

making a forward pass with example data.

the encoder and decoder components. It also demonstrates the usage of these components by

provided code includes details about the layers, embeddings, and initializations used in both

encoder processes input sequences, and the decoder generates output sequences. The

This code defines classes for the encoder and decoder in a sequence-to-sequence model. The

 Build Encoder:

I. Build Models:

Target Output shape (with batches): (149, 30)

Decoder Input shape (with batches): (149, 30)

Encoder Input shape (with batches): (149, 30)

Number of validation data: 447

Number of validation batches: 3

Number of training data: 3427

Number of train batches: 23

print(f'Target Output shape (with batches): {\_[2].shape}')

print(f'Decoder Input shape (with batches): {\_[1].shape}')

print(f'Encoder Input shape (with batches): {\_[0].shape}')

print(f'Number of validation data: {len(val\_data)\*batch\_size}')

print(f'Number of validation batches: {len(val\_data)}')

print(f'Number of training data: {len(train\_data)\*batch\_size}')

print(f'Number of train batches: {len(train\_data)}')

\_=train\_data\_iterator.next()

val\_data=val\_data.prefetch(tf.data.AUTOTUNE)

val\_data=val\_data.batch(batch\_size)

val\_data=data.skip(int(.9\*len(data))).take(int(.1\*len(data)))

train\_data\_iterator=train\_data.as\_numpy\_iterator()

train\_data=train\_data.prefetch(tf.data.AUTOTUNE)

self.outputs=[encoder\_state\_h,encoder\_state\_c]

encoder\_outputs,encoder\_state\_h,encoder\_state\_c=self.lstm(x)

x=Dropout(.4)(x)

x=self.normalize(x)

x=self.embedding(encoder\_inputs)

self.inputs=encoder\_inputs

def call(self,encoder\_inputs):

)

kernel\_initializer=tf.keras.initializers.GlorotNormal()

name='encoder\_lstm',

return\_sequences=True,

return\_state=True,

dropout=.4,

units,

self.lstm=LSTM(

self.normalize=LayerNormalization()

)

embeddings\_initializer=tf.keras.initializers.GlorotNormal()

mask\_zero=True,

name='encoder\_embedding',

embedding\_dim,

vocab\_size,

self.embedding=Embedding(

self.embedding\_dim=embedding\_dim

self.vocab\_size=vocab\_size

self.units=units

super().\_\_init\_\_(\*args,\*\*kwargs)

def \_\_init\_\_(self,units,embedding\_dim,vocab\_size,\*args,\*\*kwargs) -> None:

class Encoder(tf.keras.models.Model):

Input 11:

[ 0.43155715, 0.03135502, -0.33463806, ..., -0.47625306,

-0.04292646, -0.58768904],

[-0.2582597 , -0.25323495, -0.06649272, ..., 0.16527973,

...,

0.36470377, 0.23766585],

[ 0.07106856, -0.0739173 , -0.3641197 , ..., -0.3794833 ,

0.17292541, -0.2922624 ],

[ 0.14154069, 0.17045322, -0.17749965, ..., -0.02712595,

0.34630865, 0.2613009 ],

array([[ 0.34589 , -0.30134732, -0.43572 , ..., -0.3102559 ,

<tf.Tensor: shape=(149, 256), dtype=float32, numpy=

-0.05518465, 0.25142196]], dtype=float32)>,

[ 0.1164271 , -0.07769038, -0.06414707, ..., -0.06539135,

0.10233591, 0.20114912],

[ 0.20819993, 0.01196991, -0.09635217, ..., -0.18782297,

-0.02598592, -0.22455114],

[-0.14210635, -0.12942064, -0.03288083, ..., 0.0568463 ,

...,

0.10292757, 0.13625325],

[ 0.03628462, -0.02653611, -0.11506603, ..., -0.14669597,

0.10152507, -0.12077457],

[ 0.08443093, 0.08849293, -0.09065959, ..., -0.00959182,

0.10568858, 0.14841646],

array([[ 0.16966951, -0.10419625, -0.12700348, ..., -0.12251794,

(<tf.Tensor: shape=(149, 256), dtype=float32, numpy=

OUTPUT:

encoder.call(\_[0])

encoder=Encoder(lstm\_cells,embedding\_dim,vocab\_size,name='encoder')

return encoder\_state\_h,encoder\_state\_c

activation='softmax',

vocab\_size,

self.fc=Dense(

)

kernel\_initializer=tf.keras.initializers.HeNormal()

name='decoder\_lstm',

return\_sequences=True,

return\_state=True,

dropout=.4,

units,

self.lstm=LSTM(

self.normalize=LayerNormalization()

)

embeddings\_initializer=tf.keras.initializers.HeNormal()

mask\_zero=True,

name='decoder\_embedding',

embedding\_dim,

vocab\_size,

self.embedding=Embedding(

self.vocab\_size=vocab\_size

self.embedding\_dim=embedding\_dim

self.units=units

super().\_\_init\_\_(\*args,\*\*kwargs)

def \_\_init\_\_(self,units,embedding\_dim,vocab\_size,\*args,\*\*kwargs) -> None:

class Decoder(tf.keras.models.Model):

Input 12:

Build Encoder## Build Decoder

-0.17478186, 0.48899865]], dtype=float32)>)

[ 0.23173636, -0.20141824, -0.22034441, ..., -0.16035017,

0.33486888, 0.35035062],

1.9552530e-04, 1.7106640e-05, 1.0252406e-04]]], dtype=float32)>

[1.9002777e-03, 6.9266016e-04, 1.4346189e-04, ...,

1.9552530e-04, 1.7106640e-05, 1.0252406e-04],

[1.9002777e-03, 6.9266016e-04, 1.4346189e-04, ...,

1.9552530e-04, 1.7106640e-05, 1.0252406e-04],

[1.9002777e-03, 6.9266016e-04, 1.4346189e-04, ...,

...,

3.6009602e-05, 1.5537882e-04, 1.8397317e-04],

[9.6929165e-05, 2.7441782e-05, 1.3761305e-03, ...,

1.9187471e-05, 9.7244098e-05, 7.6433855e-05],

[1.4662130e-03, 8.0250365e-06, 5.4062020e-05, ...,

7.2067953e-05, 1.5453645e-03, 2.3599296e-04],

array([[[3.4059247e-04, 5.7348556e-05, 2.1294907e-05, ...,

<tf.Tensor: shape=(1, 30, 2443), dtype=float32, numpy=

OUTPUT:

decoder(\_[1][:1],encoder(\_[0][:1]))

decoder=Decoder(lstm\_cells,embedding\_dim,vocab\_size,name='decoder')

return self.fc(x)

x=Dropout(.4)(x)

x=self.normalize(x)

x,decoder\_state\_h,decoder\_state\_c=self.lstm(x,initial\_state=encoder\_states)

x=Dropout(.4)(x)

x=self.normalize(x)

x=self.embedding(decoder\_inputs)

def call(self,decoder\_inputs,encoder\_states):

)

kernel\_initializer=tf.keras.initializers.HeNormal()

name='decoder\_dense',

return self.decoder(decoder\_inputs,encoder\_states)

encoder\_states=self.encoder(encoder\_inputs)

encoder\_inputs,decoder\_inputs=inputs

def call(self,inputs):

return n\_correct / n\_total

n\_total = tf.keras.backend.sum(mask)

n\_correct = tf.keras.backend.sum(mask \* correct)

mask = tf.cast(tf.greater(y\_true, 0), dtype='float64')

correct = tf.cast(tf.equal(y\_true, pred\_values), dtype='float64')

pred\_values = tf.cast(tf.argmax(y\_pred, axis=-1), dtype='int64')

def accuracy\_fn(self,y\_true,y\_pred):

return tf.reduce\_mean(loss)

loss\*=mask

mask=tf.cast(mask,dtype=loss.dtype)

mask=tf.math.logical\_not(tf.math.equal(y\_true,0))

loss=self.loss(y\_true,y\_pred)

def loss\_fn(self,y\_true,y\_pred):

self.decoder=decoder

self.encoder=encoder

super().\_\_init\_\_(\*args,\*\*kwargs)

def \_\_init\_\_(self,encoder,decoder,\*args,\*\*kwargs):

class ChatBotTrainer(tf.keras.models.Model):

INPUT-13

model. The code then performs a forward pass with the model using example data.

custom loss and accuracy functions, training and testing steps, and the compilation of the

This code defines a ChatBotTrainer class for training and testing a chatbot model. It includes

 Build Training Model:

model(\_[:2])

)

weighted\_metrics=['loss','accuracy']

optimizer=tf.keras.optimizers.Adam(learning\_rate=learning\_rate),

loss=tf.keras.losses.SparseCategoricalCrossentropy(),

model.compile(

model=ChatBotTrainer(encoder,decoder,name='chatbot\_trainer')

INPUT-14

return metrics

metrics={'loss':loss,'accuracy':acc}

acc=self.accuracy\_fn(y,y\_pred)

loss=self.loss\_fn(y,y\_pred)

y\_pred=self.decoder(decoder\_inputs,encoder\_states,training=True)

encoder\_states=self.encoder(encoder\_inputs,training=True)

encoder\_inputs,decoder\_inputs,y=batch

def test\_step(self,batch):

return metrics

metrics={'loss':loss,'accuracy':acc}

self.optimizer.apply\_gradients(zip(grads,variables))

grads=tape.gradient(loss,variables)

variables=self.encoder.trainable\_variables+self.decoder.trainable\_variables

acc=self.accuracy\_fn(y,y\_pred)

loss=self.loss\_fn(y,y\_pred)

y\_pred=self.decoder(decoder\_inputs,encoder\_states,training=True)

encoder\_states=self.encoder(encoder\_inputs,training=True)

with tf.GradientTape() as tape:

encoder\_inputs,decoder\_inputs,y=batch

def train\_step(self,batch):

ax[0].set\_title('Loss Metrics')

ax[1].set\_ylabel('Accuracy')

ax[0].set\_ylabel('Loss')

ax[1].set\_xlabel('Epochs')

ax[0].set\_xlabel('Epochs')

ax[0].plot(history.history['val\_loss'],label='val\_loss',c = 'blue')

ax[0].plot(history.history['loss'],label='loss',c='red')

fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))

Input-16

figure.

accuracy metrics over training epochs. It uses Matplotlib for plotting and shows the resulting

This code creates a figure with two subplots to visualize training and validation loss and

IV.Visualize Metrics:

)

]

tf.keras.callbacks.ModelCheckpoint('ckpt',verbose=1,save\_best\_only=True)

tf.keras.callbacks.TensorBoard(log\_dir='logs'),

callbacks=[

validation\_data=val\_data,

epochs=100,

train\_data,

history=model.fit(

Input- 15

best model during training. The training history is stored in the history variable.

callback for monitoring the training process and the ModelCheckpoint callback to save the

data (train\_data) and validation data (val\_data). Two callbacks are specified: the TensorBoard

In this code, the model.fit function is used to train the model for 100 epochs with training

 Train Model:

encoder\_inputs=tf.keras.Input(shape=(None,))

def build\_inference\_model(self,base\_encoder,base\_decoder):

self.encoder,self.decoder=self.build\_inference\_model(base\_encoder,base\_decoder)

super().\_\_init\_\_(\*args,\*\*kwargs)

def \_\_init\_\_(self,base\_encoder,base\_decoder,\*args,\*\*kwargs):

class ChatBot(tf.keras.models.Model):

VI.Create Inference Model

print('---------------------')

print(j)

for j in i.layers:

print('Encoder layers:' if idx==0 else 'Decoder layers: ')

for idx,i in enumerate(model.layers):

model.save('models',save\_format='tf')

model.load\_weights('ckpt')

V.Save Model

plt.show()

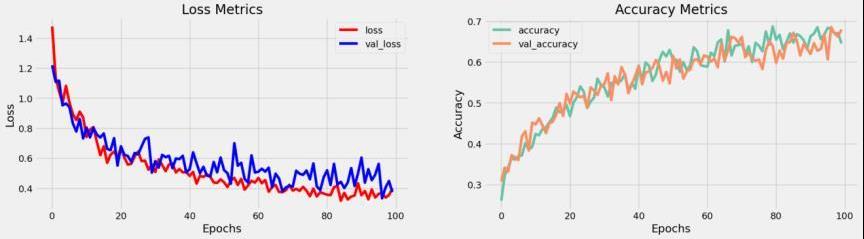
ax[1].legend()

ax[0].legend()

ax[1].plot(history.history['val\_accuracy'],label='val\_accuracy')

ax[1].plot(history.history['accuracy'],label='accuracy')

ax[1].set\_title('Accuracy Metrics')



conditional\_probability = np.asarray(conditional\_probability).astype("float64")

def sample(self,conditional\_probability,temperature=0.5):

return np.exp(z)/sum(np.exp(z))

def softmax(self,z):

self.decoder.summary()

self.encoder.summary()

def summary(self):

return encoder,decoder

)

outputs=[decoder\_outputs,[decoder\_state\_h,decoder\_state\_c]],name='chatbot\_decoder'

inputs=[decoder\_inputs,[decoder\_input\_state\_h,decoder\_input\_state\_c]],

decoder=tf.keras.models.Model(

decoder\_outputs=base\_decoder.layers[-1](x)

ate\_h,decoder\_input\_state\_c])

x,decoder\_state\_h,decoder\_state\_c=base\_decoder.layers[2](x,initial\_state=[decoder\_input\_st

x=base\_encoder.layers[1](x)

x=base\_decoder.layers[0](decoder\_inputs)

decoder\_inputs=tf.keras.Input(shape=(None,))

decoder\_input\_state\_c=tf.keras.Input(shape=(lstm\_cells,))

decoder\_input\_state\_h=tf.keras.Input(shape=(lstm\_cells,))

ate\_c],name='chatbot\_encoder')

encoder=tf.keras.models.Model(inputs=encoder\_inputs,outputs=[encoder\_state\_h,encoder\_st

x,encoder\_state\_h,encoder\_state\_c=base\_encoder.layers[2](x)

x=base\_encoder.layers[1](x)

x=base\_encoder.layers[0](encoder\_inputs)

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(' [0] ','0',text)

text=re.sub(' [9] ','9',text)

text=re.sub(' [8] ','8',text)

text=re.sub(' [7] ','7',text)

text=re.sub(' [6] ','6',text)

text=re.sub(' [5] ','5',text)

text=re.sub(' [4] ','4',text)

text=re.sub(' [3] ','3',text)

text=re.sub(' [2] ','2',text)

text=re.sub(' [1] ','1',text)

text=re.sub(' [.] ','. ',text)

text=re.sub(' - ','-',text.lower())

def postprocess(self,text):

return seq

seq[:,i]=sequences2ids(word).numpy()[0]

for i,word in enumerate(text.split()):

seq=np.zeros((1,max\_sequence\_length),dtype=np.int32)

text=clean\_text(text)

def preprocess(self,text):

return np.argmax(probas)

probas = np.random.multinomial(1, reweighted\_conditional\_probability, 1)

reweighted\_conditional\_probability = self.softmax(conditional\_probability)

conditional\_probability = np.log(conditional\_probability) / temperature

chatbot.summary()

chatbot=ChatBot(model.encoder,model.decoder,name='chatbot')

return self.postprocess(ids2sequences(decoded))

states=new\_states

target\_seq[:,:]=index

target\_seq=np.zeros((1,1))

decoded.append(index)

else:

stop\_condition=True

if word=='<end> ' or len(decoded)>=max\_sequence\_length:

word=ids2sequences([index])

index=self.sample(decoder\_outputs[0,0,:]).item()

#

index=tf.argmax(decoder\_outputs[:,-1,:],axis=-1).numpy().item()

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

while not stop\_condition:

decoded=[]

stop\_condition=False

target\_seq[:,:]=sequences2ids(['<start>']).numpy()[0][0]

target\_seq=np.zeros((1,1))

states=self.encoder(input\_seq,training=False)

input\_seq=self.preprocess(text)

def call(self,text,config=None):

return text

text=re.sub(' [\"] ','\"',text)

text=re.sub(' [\'] ','\'',text)

text=re.sub(' [\*] ','\* ',text)

text=re.sub(' [;] ','; ',text)

text=re.sub(' [:] ',': ',text)

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

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

Non-trainable params: 0

Trainable params: 1,779,083

Total params: 1,779,083

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

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

decoder\_dense (Dense)

(None, None, 2443) 627851

['decoder\_lstm[0][0]']

(None, 256)]

'input\_3[0][0]']

(None, 256),

'input\_2[0][0]',

decoder\_lstm (LSTM)

[(None, None, 256), 525312

['layer\_normalization[1][0]',

input\_3 (InputLayer)

[(None, 256)]

0

[]

input\_2 (InputLayer)

[(None, 256)]

0

[]

alization)

layer\_normalization (LayerNorm (None, None, 256) 512

['decoder\_embedding[0][0]']

decoder\_embedding (Embedding) (None, None, 256)

625408

['input\_4[0][0]']

input\_4 (InputLayer)

[(None, None)]

0

[]

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

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

Layer (type)

Output Shape

Param # Connected to

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

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

Model: "chatbot\_decoder"

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

Non-trainable params: 0

Trainable params: 1,151,232

Total params: 1,151,232

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

(None, 256)]

(None, 256),

encoder\_lstm (LSTM)

[(None, None, 256),

525312

ormalization)

layer\_normalization (LayerN (None, None, 256)

512

g)

encoder\_embedding (Embeddin (None, None, 256)

625408

input\_1 (InputLayer)

[(None, None)]

0

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

Layer (type)

Output Shape

Param #

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

Model: "chatbot\_encoder"

'do yo know me?',

'hi',

print\_conversation([

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

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

print(f'You: {text}')

for text in texts:

def print\_conversation(texts):

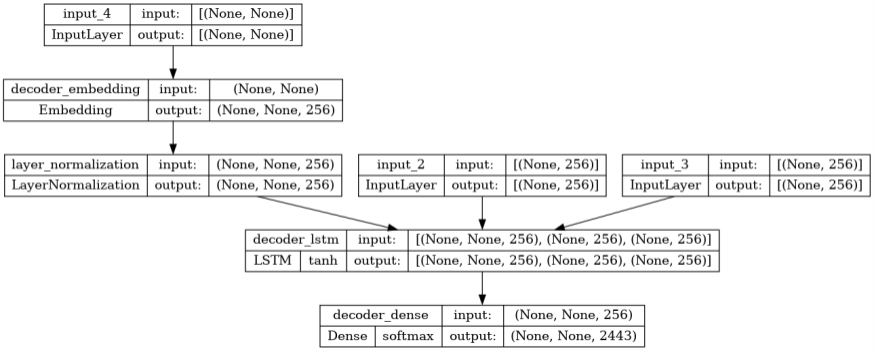
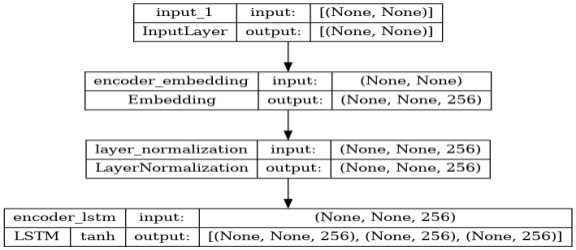
VII.Time to Chat

yer\_activations=True)

tf.keras.utils.plot\_model(chatbot.decoder,to\_file='decoder.png',show\_shapes=True,show\_la

r\_activations=True)

tf.keras.utils.plot\_model(chatbot.encoder,to\_file='encoder.png',show\_shapes=True,show\_laye



You: what is your name?

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

Bot: yes, it's too close to the other.

You: do yo know me?

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

Bot: i have to go to the bathroom.

You: hi

OUTPUT:

'''I like this restaurant because they give you free bread.'''])

'''Tomorrow we'll have rice and fish for lunch.''',

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

'''I need something that's reliable.''',

'''We'll be here forever.''',

'''Wow! that's terrible.''',

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

'''You want forgiveness? Get religion''',

'''I've read all your research on nano-technology ''',

'''You're trash ''',

'''I'm gonna put some dirt in your eye ''',

"Don't ever be in a hurry",

"i'm pretty good. thanks for asking.",

'hi, how are you doing?',

'you are bot?',

'what is your name?',

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

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

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

Bot: no, i'll be my.

You: You want forgiveness? Get religion

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

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

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

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

Bot: the tv news is reporting a bank robbery.

You: You're trash

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

Bot: that's a good idea.

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

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

Bot: it's not a great.

You: Don't ever be in a hurry

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

mind.

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

You: i'm pretty good. thanks for asking.

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

Bot: i'm going to be a teacher.

You: hi, how are you doing?

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

Bot: no, i have. all my life.

You: you are bot?

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

Bot: i have to walk the house.

import re,string

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np



FULL SOURCE CODE FOR THIS PROJECT:

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

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

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

Bot: i'll make a sandwich.

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

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

Bot: what happened?

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

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

Bot: you need a car with low mileage.

You: I need something that's reliable.

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

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

You: We'll be here forever.

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

Bot: never park your car under the house.

You: Wow! that's terrible.

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



def clean\_text(text):

plt.show()

tokens',data=df,kind='kde',fill=True,cmap='YlGnBu')

sns.jointplot(x='question tokens',y='answer

sns.histplot(x=df['answer tokens'],data=df,kde=True,ax=ax[1])

sns.histplot(x=df['question tokens'],data=df,kde=True,ax=ax[0])

sns.set\_palette('Set2')

fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))

plt.style.use('fivethirtyeight')

df['answer tokens']=df['answer'].apply(lambda x:len(x.split()))

df['question tokens']=df['question'].apply(lambda x:len(x.split()))

print(df)

df = pd.DataFrame(data)

}

]

"i've been good. i'm in school right now."

"i've been great. what about you?",

"no problem. so how have you been?",

"i'm pretty good. thanks for asking.",

"i'm fine. how about yourself?",

'answer': [

],

"i've been great. what about you?"

"no problem. so how have you been?",

"i'm pretty good. thanks for asking.",

"i'm fine. how about yourself?",

"hi, how are you doing?",

'question': [

data = {

df['decoder\_targets']=df['answer'].apply(clean\_text)+' <end>'

df['encoder\_inputs']=df['question'].apply(clean\_text)

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

return text

text=re.sub('\t',' ',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('[0]',' 0 ',text)

text=re.sub('[9]',' 9 ',text)

text=re.sub('[8]',' 8 ',text)

text=re.sub('[7]',' 7 ',text)

text=re.sub('[6]',' 6 ',text)

text=re.sub('[5]',' 5 ',text)

text=re.sub('[4]',' 4 ',text)

text=re.sub('[3]',' 3 ',text)

text=re.sub('[2]',' 2 ',text)

text=re.sub('[1]',' 1 ',text)

text=re.sub('[.]',' . ',text)

text=re.sub('-',' ',text.lower())

"embedding\_dim":256,

"lstm\_cells":256,

"batch\_size":149,

"learning\_rate":0.008,

"max\_sequence\_length":30,

"vocab\_size":2500,

params={

target tokens'],axis=1,inplace=True)

df.drop(columns=['question','answer','encoder input tokens','decoder input tokens','decoder

print(f"Max decoder target length: {df['decoder target tokens'].max()}")

print(f"Max decoder input length: {df['decoder input tokens'].max()}")

print(f"Max encoder input length: {df['encoder input tokens'].max()}")

tokens']]['encoder\_inputs'].values.tolist())}")

print(f"After preprocessing: {' '.join(df[df['encoder input tokens'].max()==df['encoder input

plt.show()

tokens',data=df,kind='kde',fill=True,cmap='YlGnBu')

sns.jointplot(x='encoder input tokens',y='decoder target

sns.histplot(x=df['decoder target tokens'],data=df,kde=True,ax=ax[2])

sns.histplot(x=df['decoder input tokens'],data=df,kde=True,ax=ax[1])

sns.histplot(x=df['encoder input tokens'],data=df,kde=True,ax=ax[0])

sns.set\_palette('Set2')

fig,ax=plt.subplots(nrows=1,ncols=3,figsize=(20,5))

plt.style.use('fivethirtyeight')

df['decoder target tokens']=df['decoder\_targets'].apply(lambda x:len(x.split()))

df['decoder input tokens']=df['decoder\_inputs'].apply(lambda x:len(x.split()))

df['encoder input tokens']=df['encoder\_inputs'].apply(lambda x:len(x.split()))

df.head(10)

df['decoder\_inputs']='<start> '+df['answer'].apply(clean\_text)+' <end>'

return decode

decode+=vectorize\_layer.get\_vocabulary()[id]+' '

for id in ids:

ids=[ids]

if type(ids)==int:

decode=''

def ids2sequences(ids):

return vectorize\_layer(sequence)

def sequences2ids(sequence):

print(f'{vectorize\_layer.get\_vocabulary()[:12]}')

print(f'Vocab size: {len(vectorize\_layer.get\_vocabulary())}')

vocab\_size=len(vectorize\_layer.get\_vocabulary())

vectorize\_layer.adapt(df['encoder\_inputs']+' '+df['decoder\_targets']+' <start> <end>')

)

output\_sequence\_length=max\_sequence\_length

output\_mode='int',

standardize=None,

max\_tokens=vocab\_size,

vectorizelayer=TextVectorization(

df.head(10)

max\_sequence\_length=params['max\_sequence\_length']

vocab\_size=params['vocab\_size']

lstm\_cells=params['lstm\_cells']

embedding\_dim=params['embedding\_dim']

batch\_size=params['batch\_size']

learning\_rate=params['learning\_rate']

}

"buffer\_size":10000

self.normalize=LayerNormalization()

)

embeddings\_initializer=tf.keras.initializers.GlorotNormal()

mask\_zero=True,

name='encoder\_embedding',

embedding\_dim,

vocab\_size,

self.embedding=Embedding(

self.embedding\_dim=embedding\_dim

self.vocab\_size=vocab\_size

self.units=units

super().\_\_init\_\_(\*args,\*\*kwargs)

def \_\_init\_\_(self,units,embedding\_dim,vocab\_size,\*args,\*\*kwargs) -> None:

class Encoder(tf.keras.models.Model):

data=data.shuffle(buffer\_size)

data=tf.data.Dataset.from\_tensor\_slices((x,yd,y))

print(f'Decoder target: {y[0][:12]} ...')

decoder is the output of the previous timestep

print(f'Decoder input: {yd[0][:12]} ...') # shifted by one time step of the target as input to

print(f'Encoder input: {x[0][:12]} ...')

print(f'Decoder target shape: {y.shape}')

print(f'Decoder input shape: {yd.shape}')

print(f'Encoder input shape: {x.shape}')

print(f'Question to tokens: {sequences2ids("hi , how are you ?")[:10]}')

print(f'Question sentence: hi , how are you ?')

y=sequences2ids(df['decoder\_targets'])

yd=sequences2ids(df['decoder\_inputs'])

x=sequences2ids(df['encoder\_inputs'])

val\_data=data.skip(int(.9\*len(data))).take(int(.1\*len(data)))

train\_data\_iterator=train\_data.as\_numpy\_iterator()

train\_data=train\_data.prefetch(tf.data.AUTOTUNE)

train\_data=train\_data.batch(batch\_size)

train\_data=train\_data.shuffle(buffer\_size)

train\_data=train\_data.cache()

train\_data=data.take(int(.9\*len(data)))

encoder.call(\_[0])

encoder=Encoder(lstm\_cells,embedding\_dim,vocab\_size,name='encoder')

return encoder\_state\_h,encoder\_state\_c

self.outputs=[encoder\_state\_h,encoder\_state\_c]

encoder\_outputs,encoder\_state\_h,encoder\_state\_c=self.lstm(x)

x=Dropout(.4)(x)

x=self.normalize(x)

x=self.embedding(encoder\_inputs)

self.inputs=encoder\_inputs

def call(self,encoder\_inputs):

)

kernel\_initializer=tf.keras.initializers.GlorotNormal()

name='encoder\_lstm',

return\_sequences=True,

return\_state=True,

dropout=.4,

units,

self.lstm=LSTM(

return\_state=True,

dropout=.4,

units,

self.lstm=LSTM(

self.normalize=LayerNormalization()

)

embeddings\_initializer=tf.keras.initializers.HeNormal()

mask\_zero=True,

name='decoder\_embedding',

embedding\_dim,

vocab\_size,

self.embedding=Embedding(

self.vocab\_size=vocab\_size

self.embedding\_dim=embedding\_dim

self.units=units

super().\_\_init\_\_(\*args,\*\*kwargs)

def \_\_init\_\_(self,units,embedding\_dim,vocab\_size,\*args,\*\*kwargs) -> None:

class Decoder(tf.keras.models.Model):

print(f'Target Output shape (with batches): {\_[2].shape}')

print(f'Decoder Input shape (with batches): {\_[1].shape}')

print(f'Encoder Input shape (with batches): {\_[0].shape}')

print(f'Number of validation data: {len(val\_data)\*batch\_size}')

print(f'Number of validation batches: {len(val\_data)}')

print(f'Number of training data: {len(train\_data)\*batch\_size}')

print(f'Number of train batches: {len(train\_data)}')

\_=train\_data\_iterator.next()

val\_data=val\_data.prefetch(tf.data.AUTOTUNE)

val\_data=val\_data.batch(batch\_size)

def loss\_fn(self,y\_true,y\_pred):

self.decoder=decoder

self.encoder=encoder

super().\_\_init\_\_(\*args,\*\*kwargs)

def \_\_init\_\_(self,encoder,decoder,\*args,\*\*kwargs):

class ChatBotTrainer(tf.keras.models.Model):

decoder(\_[1][:1],encoder(\_[0][:1]))

decoder=Decoder(lstm\_cells,embedding\_dim,vocab\_size,name='decoder')

return self.fc(x)

x=Dropout(.4)(x)

x=self.normalize(x)

x,decoder\_state\_h,decoder\_state\_c=self.lstm(x,initial\_state=encoder\_states)

x=Dropout(.4)(x)

x=self.normalize(x)

x=self.embedding(decoder\_inputs)

def call(self,decoder\_inputs,encoder\_states):

)

kernel\_initializer=tf.keras.initializers.HeNormal()

name='decoder\_dense',

activation='softmax',

vocab\_size,

self.fc=Dense(

)

kernel\_initializer=tf.keras.initializers.HeNormal()

name='decoder\_lstm',

return\_sequences=True,

grads=tape.gradient(loss,variables)

variables=self.encoder.trainable\_variables+self.decoder.trainable\_variables

acc=self.accuracy\_fn(y,y\_pred)

loss=self.loss\_fn(y,y\_pred)

y\_pred=self.decoder(decoder\_inputs,encoder\_states,training=True)

encoder\_states=self.encoder(encoder\_inputs,training=True)

with tf.GradientTape() as tape:

encoder\_inputs,decoder\_inputs,y=batch

def train\_step(self,batch):

return self.decoder(decoder\_inputs,encoder\_states)

encoder\_states=self.encoder(encoder\_inputs)

encoder\_inputs,decoder\_inputs=inputs

def call(self,inputs):

return n\_correct / n\_total

n\_total = tf.keras.backend.sum(mask)

n\_correct = tf.keras.backend.sum(mask \* correct)

mask = tf.cast(tf.greater(y\_true, 0), dtype='float64')

correct = tf.cast(tf.equal(y\_true, pred\_values), dtype='float64')

pred\_values = tf.cast(tf.argmax(y\_pred, axis=-1), dtype='int64')

def accuracy\_fn(self,y\_true,y\_pred):

return tf.reduce\_mean(loss)

loss\*=mask

mask=tf.cast(mask,dtype=loss.dtype)

mask=tf.math.logical\_not(tf.math.equal(y\_true,0))

loss=self.loss(y\_true,y\_pred)

fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))

)

]

tf.keras.callbacks.ModelCheckpoint('ckpt',verbose=1,save\_best\_only=True)

tf.keras.callbacks.TensorBoard(log\_dir='logs'),

callbacks=[

validation\_data=val\_data,

epochs=100,

train\_data,

history=model.fit(

model(\_[:2])

)

weighted\_metrics=['loss','accuracy']

optimizer=tf.keras.optimizers.Adam(learning\_rate=learning\_rate),

loss=tf.keras.losses.SparseCategoricalCrossentropy(),

model.compile(

model=ChatBotTrainer(encoder,decoder,name='chatbot\_trainer')

return metrics

metrics={'loss':loss,'accuracy':acc}

acc=self.accuracy\_fn(y,y\_pred)

loss=self.loss\_fn(y,y\_pred)

y\_pred=self.decoder(decoder\_inputs,encoder\_states,training=True)

encoder\_states=self.encoder(encoder\_inputs,training=True)

encoder\_inputs,decoder\_inputs,y=batch

def test\_step(self,batch):

return metrics

metrics={'loss':loss,'accuracy':acc}

self.optimizer.apply\_gradients(zip(grads,variables))

x=base\_encoder.layers[1](x)

x=base\_encoder.layers[0](encoder\_inputs)

encoder\_inputs=tf.keras.Input(shape=(None,))

def build\_inference\_model(self,base\_encoder,base\_decoder):

self.encoder,self.decoder=self.build\_inference\_model(base\_encoder,base\_decoder)

super().\_\_init\_\_(\*args,\*\*kwargs)

def \_\_init\_\_(self,base\_encoder,base\_decoder,\*args,\*\*kwargs):

class ChatBot(tf.keras.models.Model):

print('---------------------')

print(j)

for j in i.layers:

print('Encoder layers:' if idx==0 else 'Decoder layers: ')

for idx,i in enumerate(model.layers):

model.save('models',save\_format='tf')

model.load\_weights('ckpt')

plt.show()

ax[1].legend()

ax[0].legend()

ax[1].plot(history.history['val\_accuracy'],label='val\_accuracy')

ax[1].plot(history.history['accuracy'],label='accuracy')

ax[1].set\_title('Accuracy Metrics')

ax[0].set\_title('Loss Metrics')

ax[1].set\_ylabel('Accuracy')

ax[0].set\_ylabel('Loss')

ax[1].set\_xlabel('Epochs')

ax[0].set\_xlabel('Epochs')

ax[0].plot(history.history['val\_loss'],label='val\_loss',c = 'blue')

ax[0].plot(history.history['loss'],label='loss',c='red')

conditional\_probability = np.log(conditional\_probability) / temperature

conditional\_probability = np.asarray(conditional\_probability).astype("float64")

def sample(self,conditional\_probability,temperature=0.5):

return np.exp(z)/sum(np.exp(z))

def softmax(self,z):

self.decoder.summary()

self.encoder.summary()

def summary(self):

return encoder,decoder

)

outputs=[decoder\_outputs,[decoder\_state\_h,decoder\_state\_c]],name='chatbot\_decoder'

inputs=[decoder\_inputs,[decoder\_input\_state\_h,decoder\_input\_state\_c]],

decoder=tf.keras.models.Model(

decoder\_outputs=base\_decoder.layers[-1](x)

state\_h,decoder\_input\_state\_c])

x,decoder\_state\_h,decoder\_state\_c=base\_decoder.layers[2](x,initial\_state=[decoder\_input\_

x=base\_encoder.layers[1](x)

x=base\_decoder.layers[0](decoder\_inputs)

decoder\_inputs=tf.keras.Input(shape=(None,))

decoder\_input\_state\_c=tf.keras.Input(shape=(lstm\_cells,))

decoder\_input\_state\_h=tf.keras.Input(shape=(lstm\_cells,))

\_state\_c],name='chatbot\_encoder')

encoder=tf.keras.models.Model(inputs=encoder\_inputs,outputs=[encoder\_state\_h,encoder

x,encoder\_state\_h,encoder\_state\_c=base\_encoder.layers[2](x)

text=re.sub(' [&] ','& ',text)

text=re.sub(' [$] ','$ ',text)

text=re.sub(' [!] ','! ',text)

text=re.sub(' [?] ','? ',text)

text=re.sub(' [,] ',', ',text)

text=re.sub(' [0] ','0',text)

text=re.sub(' [9] ','9',text)

text=re.sub(' [8] ','8',text)

text=re.sub(' [7] ','7',text)

text=re.sub(' [6] ','6',text)

text=re.sub(' [5] ','5',text)

text=re.sub(' [4] ','4',text)

text=re.sub(' [3] ','3',text)

text=re.sub(' [2] ','2',text)

text=re.sub(' [1] ','1',text)

text=re.sub(' [.] ','. ',text)

text=re.sub(' - ','-',text.lower())

def postprocess(self,text):

return seq

seq[:,i]=sequences2ids(word).numpy()[0]

for i,word in enumerate(text.split()):

seq=np.zeros((1,max\_sequence\_length),dtype=np.int32)

text=clean\_text(text)

def preprocess(self,text):

return np.argmax(probas)

probas = np.random.multinomial(1, reweighted\_conditional\_probability, 1)

reweighted\_conditional\_probability = self.softmax(conditional\_probability)

return self.postprocess(ids2sequences(decoded))

states=new\_states

target\_seq[:,:]=index

target\_seq=np.zeros((1,1))

decoded.append(index)

else:

stop\_condition=True

if word=='<end> ' or len(decoded)>=max\_sequence\_length:

word=ids2sequences([index])

index=self.sample(decoder\_outputs[0,0,:]).item()

#

index=tf.argmax(decoder\_outputs[:,-1,:],axis=-1).numpy().item()

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

while not stop\_condition:

decoded=[]

stop\_condition=False

target\_seq[:,:]=sequences2ids(['<start>']).numpy()[0][0]

target\_seq=np.zeros((1,1))

states=self.encoder(input\_seq,training=False)

input\_seq=self.preprocess(text)

def call(self,text,config=None):

return 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)

'''I like this restaurant because they give you free bread.'''])

'''Tomorrow we'll have rice and fish for lunch.''',

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

'''I need something that's reliable.''',

'''We'll be here forever.''',

'''Wow! that's terrible.''',

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

'''You want forgiveness? Get religion''',

'''I've read all your research on nano-technology ''',

'''You're trash ''',

'''I'm gonna put some dirt in your eye ''',

"Don't ever be in a hurry",

"i'm pretty good. thanks for asking.",

'hi, how are you doing?',

'you are bot?',

'what is your name?',

'do yo know me?',

'hi',

print\_conversation([

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

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

print(f'You: {text}')

for text in texts:

def print\_conversation(texts):

yer\_activations=True)

tf.keras.utils.plot\_model(chatbot.decoder,to\_file='decoder.png',show\_shapes=True,show\_la

yer\_activations=True)

tf.keras.utils.plot\_model(chatbot.encoder,to\_file='encoder.png',show\_shapes=True,show\_la

chatbot.summary()

chatbot=ChatBot(model.encoder,model.decoder,name='chatbot')

THANK YOU