**CREATE AN CHATBOT BY USING PYTHON**

**PHASE 3:PROJECT SUBMISSION**

**ABSTRACT:**

In today's digital age, chatbots have emerged as an innovative and efficient means of engaging with users, automating tasks, and delivering personalized experiences. This project aims to create a Python-based chatbot that leverages cutting-edge technologies to provide a seamless conversational interface

**Objectives:**

**Chatbot's Purpose:** We will begin by identifying the specific use case and objectives for the chatbot. Whether it's assisting customers, providing information, or automating tasks, a clear understanding of the chatbot's purpose is essential.

**Select Appropriate Technologies:** The project will evaluate and choose the most suitable Python libraries, frameworks, and tools for NLP and chatbot development, such as NLTK, spaCy, and the Transformers library.

**Importing libraries:**

*#model*

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

*#nlp processing*

import unicodedata

import re

import numpy as np

import warnings

warnings.filterwarnings('ignore')

**Data Preprocessing:** To train the chatbot's language model, a dataset will be collected and preprocessed, ensuring high-quality input data for model training.

**The basic text processing in NLP are:**

1. Sentence Segmentation
2. Normalization
3. Tokenization

**Segmentation:**

Segmentation is the process of dividing a larger and more diverse group, market, dataset, or entity into smaller, distinct, and more homogenous segments or subgroups based on shared characteristics, attributes, or criteria.

*#reading data*

data=open('/kaggle/input/simple-dialogs-for-chatbot/dialogs.txt','r').read()

*#paried list of question and corresponding answer*

QA\_list=[QA.split('**\t**') for QA **in** data.split('**\n**')]

print(QA\_list[:5])

questions=[row[0] for row **in** QA\_list]

answers=[row[1] for row **in** QA\_list]

print(questions[0:5])

print(answers[0:5])

**Normalization:**

Normalization is a process used in various fields, such as mathematics, statistics, and databases, to standardize or scale data, making it more manageable, comparable, or suitable for specific analysis or processing.

def remove\_diacritic(text):

return ''.join(char for char **in** unicodedata.normalize('NFD',text)

if unicodedata.category(char) !='Mn')

def preprocessing(text):

*#Case folding and removing extra whitespaces*

text=remove\_diacritic(text.lower().strip())

*#Ensuring punctuation marks to be treated as tokens*

text=re.sub(r"([?.!,¿])", r" \1 ", text)

*#Removing redundant spaces*

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

*#Removing non alphabetic characters*

text=re.sub(r"[^a-zA-Z?.!,¿]+", " ", text)

text=text.strip()

*#Indicating the start and end of each sentence*

text='<start> ' + text + ' <end>'

return text

preprocessed\_questions=[preprocessing(sen) for sen **in** questions]

preprocessed\_answers=[preprocessing(sen) for sen **in** answers]

print(preprocessed\_questions[0])

print(preprocessed\_answers[0])

**Tokenization:**

These tokens serve as the fundamental building blocks for natural language processing (NLP) and text analysis tasks. Tokenization is a crucial step in many NLP applications, including text classification, language modeling, and information  retrieval.

def tokenize(lang):

lang\_tokenizer = tf.keras.preprocessing.text.Tokenizer(

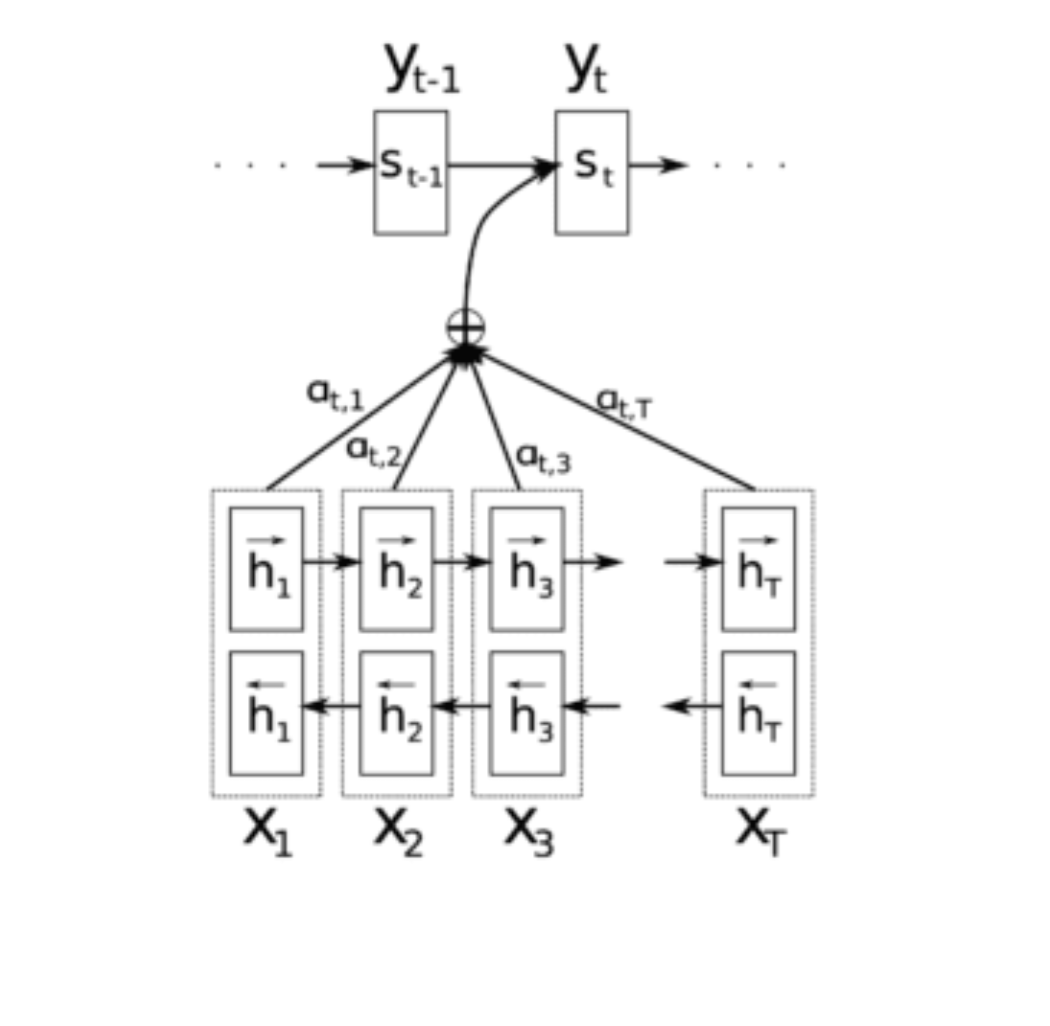
filters='')

*#build vocabulary on unique words*

lang\_tokenizer.fit\_on\_texts(lang)

**r**eturn lang\_tokenizer

**Building Model Architecture**



**Encoder:**

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

def \_\_init\_\_(self, vocab\_size, embedding\_dim, enc\_units, batch\_sz):

super(Encoder, self).\_\_init\_\_()

self.batch\_sz = batch\_sz

self.enc\_units = enc\_units

self.embedding = tf.keras.layers.Embedding(vocab\_size, embedding\_dim)

self.gru = tf.keras.layers.GRU(self.enc\_units,

return\_sequences=True,

return\_state=True,

recurrent\_initializer='glorot\_uniform')

def call(self, x, hidden):

x = self.embedding(x)

output, state = self.gru(x, initial\_state = hidden)

return output, state

def initialize\_hidden\_state(self):

return tf.zeros((self.batch\_sz, self.enc\_units))

encoder = Encoder(vocab\_inp\_size, embedding\_dim, units, BATCH\_SIZE)

*# sample input*

sample\_hidden = encoder.initialize\_hidden\_state()

sample\_output, sample\_hidden = encoder(example\_input\_batch, sample\_hidden)

print ('Encoder output shape: (batch size, sequence length, units) **{}**'.format(sample\_output.shape))

print ('Encoder Hidden state shape: (batch size, units) **{}**'.format(sample\_hidden.shape))

**Encoder output shape: (batch size, sequence length, units) (64, 24, 1024)**

**Encoder Hidden state shape: (batch size, units) (64, 1024)**

**Decoder:**

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

def \_\_init\_\_(self, vocab\_size, embedding\_dim, dec\_units, batch\_sz):

super(Decoder, self).\_\_init\_\_()

self.batch\_sz = batch\_sz

self.dec\_units = dec\_units

self.embedding = tf.keras.layers.Embedding(vocab\_size, embedding\_dim)

self.gru = tf.keras.layers.GRU(self.dec\_units,

return\_sequences=True,

return\_state=True,

recurrent\_initializer='glorot\_uniform')

self.fc = tf.keras.layers.Dense(vocab\_size)

*# used for attention*

self.attention = BahdanauAttention(self.dec\_units)

def call(self, x, hidden, enc\_output):

*# enc\_output shape == (batch\_size, max\_length, hidden\_size)*

context\_vector, attention\_weights = self.attention(hidden, enc\_output)

*# x shape after passing through embedding == (batch\_size, 1, embedding\_dim)*

x = self.embedding(x)

*# x shape after concatenation == (batch\_size, 1, embedding\_dim + hidden\_size)*

x = tf.concat([tf.expand\_dims(context\_vector, 1), x], axis=-1)

*# passing the concatenated vector to the GRU*

output, state = self.gru(x)

*# output shape == (batch\_size \* 1, hidden\_size)*

output = tf.reshape(output, (-1, output.shape[2]))

*# output shape == (batch\_size, vocab)*

x = self.fc(output)

return x, state, attention\_weights

decoder = Decoder(vocab\_tar\_size, embedding\_dim, units, BATCH\_SIZE)

sample\_decoder\_output, \_, \_ = decoder(tf.random.uniform((BATCH\_SIZE, 1)),

sample\_hidden, sample\_output)

print ('Decoder output shape: (batch\_size, vocab size) **{}**'.format(sample\_decoder\_output.shape))

**Decoder output shape: (batch\_size, vocab size) (64, 2349)**

**Model Development:** Leveraging state-of-the-art NLP models, the chatbot will be designed to understand and generate human-like text responses, allowing for natural and context-aware conversations.  
  
**User Interface:** A user-friendly interface will be created to enable users to interact with the chatbot seamlessly. This can be through a web application, a messaging platform, or other channels as per the project's requirements.  
  
**Integration and Testing:** The chatbot will be integrated into the target platform and rigorously tested to ensure its functionality, accuracy, and user-friendliness. User feedback will be used to fine-tune and improve the chatbot's performance.  
  
**Scalability and Maintenance:** The project will consider the scalability of the chatbot, accommodating increasing user interactions. Additionally, a maintenance plan will be established to keep the chatbot up-to-date and effective.

**Innovation and Future Development:**The chatbot development will stay innovative by exploring emerging technologies and trends in NLP and conversational AI, ensuring it remains a cutting-edge solution for users.

**Conculsion:**

 This project will create an innovative Python-based chatbot with a clear focus on user engagement, automation, and providing personalized experiences. It aims to leverage the latest advancements in NLP to build a robust and efficient conversational agent that can adapt to various domains and use cases.