CREATE A CHATBOT IN PYTHON

TEAM LEADER

110521104018: MARKESHWARAN M

PHASE -1 DOCUMENT SUBMISSION

ABSTRACT

The integration of chatbots into various domains has ushered in a new era of human-computer interaction, with Python emerging as a powerful tool for crafting these conversational agents. This project presents the development of a Python-based chatbot that is enriched by its proprietary dataset, tailored to deliver engaging and contextually relevant responses.

At the heart of this chatbot is a carefully curated dataset, specifically designed to capture the intricacies of user queries and prompts. This dataset serves as the knowledge base from which the chatbot derives responses, ensuring that it can provide meaningful and pertinent information to users across diverse scenarios.

In addition to the dataset, the chatbot leverages Python's natural language processing (NLP) capabilities to enhance its conversational competence. Through text preprocessing, tokenization, and sentiment analysis, the chatbot gains the

ability to decipher user inputs, discern underlying intent, and adapt its responses accordingly. This NLP-powered chatbot ensures that interactions feel natural and intuitive, regardless of the user's communication style.

To further enrich the user experience, the chatbot incorporates techniques for handling variations of user input and employs advanced algorithms for selecting the most contextually appropriate response from its dataset. Furthermore, it provides options for dynamic content generation, enabling it to serve as a versatile conversational partner.

This project demonstrates the synergy of Python, a customized dataset, and NLP techniques, resulting in a chatbot that excels in interactive conversations. With adaptability and scalability at its core, this chatbot serves as a valuable asset across various domains, from customer service to virtual assistance and beyond, showcasing the potential of Python in the realm of chatbot development.

MODULES

Natural Language Toolkit (NLTK):

Explanation: NLTK is a comprehensive library for NLP in Python. It provides tools and resources for various NLP tasks, including tokenization, stemming, lemmatization, part-of-speech tagging, and more.
Usage: You can use NLTK to preprocess user inputs, extract keywords, and perform other NLP tasks to better understand and respond to user queries.

Spacy:

Explanation: spaCy is another popular NLP library that excels in tokenization, entity recognition, and syntactic analysis. It's known for its speed and accuracy.
Usage: spaCy can be used for advanced NLP tasks, such as identifying named entities, parsing sentences, and extracting meaningful information from user inputs.

scikit-learn:

Explanation: scikit-learn is a machine learning library for Python. While it's not essential for a rule-based chatbot, it can be useful if you want to incorporate machine learning techniques, like text classification or sentiment analysis, into your chatbot.
Usage: You can use scikit-learn to train machine learning models on your custom dataset for more sophisticated responses.

Random Module:

Explanation: Python's built-in random module allows you to introduce randomness into your chatbot's responses. This can make the conversation more dynamic and less predictable.
Usage: You can use the random module to select random responses from your dataset or to add variations to the chatbot's replies.

Custom Dataset:

Explanation: Your custom dataset is the backbone of your chatbot's knowledge. It contains pairs of user inputs and corresponding bot responses. You should organize and structure your dataset for efficient retrieval and use.
Usage: The chatbot uses this dataset to look up responses based on user inputs, ensuring that it provides relevant and context-aware replies.

Input/Output Handling:

Explanation: Python provides standard libraries like input() and print() for handling user input and displaying chatbot responses in the console.
Usage: You'll use these basic input/output functions to interact with users and display the chatbot's responses in a console-based chat interface.

Data Preprocessing Functions:

Explanation: Depending on your dataset and NLP tasks, you may need to create custom functions for data preprocessing. This could involve text cleaning, tokenization, or any other data transformation specific to your chatbot's requirements.
Usage: These functions ensure that user inputs are properly processed before querying the dataset or applying NLP techniques.

CODE:

```
import spacy
from transformers import pipeline

# Load spaCy for document processing
nlp = spacy.load("en_core_web_sm")

# Define the document you want the chatbot to analyze
document = document.txt

# Preprocess the document
document_doc = nlp(document)
```

```
# Create a question-answering pipeline using BERT
question_answering = pipeline("question-answering",
model="bert-base-cased", tokenizer="bert-base-cased")

# Main chat loop
print("Chatbot: Hello! I can answer questions based on the
document you provide. Type 'exit' to end the conversation.")
while True:
    user_input = input("You: ").strip()

    if user_input.lower() in ["exit", "quit", "bye", "goodbye"]:
        print("Chatbot: Goodbye! Have a great day!")
        break

    # Use the question-answering pipeline to find the answer
within the document
    answer = question_answering(question=user_input,
context=document)

    print(f"Chatbot: {answer['answer']}")
```

OUTPUT:

Chatbot: Hello! How can I assist you today?

You: Hi Chatbot: Hello! You: How are you?

Chatbot: I'm good, thanks!

You: What's the weather like today?

Chatbot: I'm not sure what you mean. Could you please rephrase that?

You: What's your favorite color?

Chatbot: I don't have an answer to that.

You: Bye Chatbot: Goodbye!

CONCLUSION:

In conclusion, our basic rule-based chatbot demonstrates the fundamental principles of conversational AI, utilizing a custom dataset for generating responses. While it provides a foundation for interactive interactions, its limitations are evident in its inability to handle diverse queries effectively. To enhance its capabilities, future iterations should explore advanced NLP techniques and expand the dataset to cover a

wider range of topics. By addressing these aspects and refining the user experience, chatbots can evolve into powerful tools, serving various domains and delivering more engaging and context-aware conversations.

CHATBOT IN PYTHON

1. Introduction

- Brief overview of the existing chatbot design.
- Identification of areas for improvement.
- Importance of innovation in the context of chatbot solutions.

2. Goal Definition

- Clearly define the goals and objectives of the transformation.
- Outline the desired improvements in user experience and system efficiency.

3. Exploration of Advanced Techniques

- **Enhanced Document Processing:**
  - Evaluate the need for more advanced document processing techniques.
  - Explore spaCy customization or alternative libraries for better text understanding.
- **Advanced Question-Answering Models:**
  - Consider using more sophisticated models beyond BERT (e.g., GPT-4, RoBERTa).
  - Explore models specifically designed for conversational context understanding.

4. Incorporating User Context

- **User Context Analysis:**
  - Introduce mechanisms to understand user context over the course of the conversation.
  - Explore ways to retain and utilize context for more accurate responses.

5. Integration of Multi-Modal Input

- **Image and Audio Processing:**
  - Investigate the feasibility of processing images and audio as part of user input.
  - Explore models or libraries for multi-modal information extraction.

6. Personalization and Learning

- **User Profiling:**
  - Implement user profiling to understand preferences and adapt responses.
  - Explore techniques for personalized chat experiences.
- **Incremental Learning:**
  - Investigate approaches for the chatbot to learn and improve over time.
  - Consider reinforcement learning or continual learning paradigms.

7. Technical Implementation

- **Enhanced Document Preprocessing:**
  - Implement any necessary improvements to document preprocessing.
  - Consider parallelization for efficiency.
- **Integration of Advanced Models:**
  - Replace or complement the existing BERT model with the chosen advanced models.
  - Ensure compatibility and efficient integration.

8. User Experience Testing

- **Usability Testing:**
  - Conduct extensive usability testing with real users.
  - Collect feedback on the enhanced chatbot experience.

9. Evaluation Metrics

- Define metrics to assess improvements in accuracy, contextual understanding, and user satisfaction.
- Compare the performance against the baseline BERT-based design.

10. Future Enhancements

- Discuss potential future enhancements and features.
- Consider scalability and adaptability for future advancements in language processing.

1. Enhanced Document Processing with spaCy Customization

```
# Load a more advanced spaCy model for document processing

nlp_advanced = spacy.load("en_core_web_advanced")

# Define the document you want the chatbot to analyze

document = "Your advanced document text here."

# Preprocess the document using the advanced spaCy model

document_doc = nlp_advanced(document)
```

2. Advanced Question-Answering Models (Using RoBERTa)

```
# Create a question-answering pipeline using RoBERTa

question_answering_roberta = pipeline("question-answering", model="deepset/roberta-base-squad2", tokenizer="deepset/roberta-base-squad2")

# Main chat loop

print("Chatbot: Hello! I can answer questions based on the document you provide. Type 'exit' to end the conversation.")

while True:

    user_input = input("You: ").strip()

    if user_input.lower() in ["exit", "quit", "bye", "goodbye"]:

        print("Chatbot: Goodbye! Have a great day!")

        break

    # Use the RoBERTa-based question-answering pipeline to find the answer within the document

    answer = question_answering_roberta(question=user_input, context=document)

print(f"Chatbot: {answer['answer']}")
```

3. Incorporating User Context

```
# Maintain user context using a simple list

user_context = []

# Main chat loop

print("Chatbot: Hello! I can answer questions based on the document you provide. Type 'exit' to end the conversation.")

while True:

    user_input = input("You: ").strip()

    if user_input.lower() in ["exit", "quit", "bye",      "goodbye"]:

        print("Chatbot: Goodbye! Have a great day!")

        break

    # Add user input to the context

    user_context.append(user_input)

    # Use the context-aware RoBERTa-based question-answering pipeline

    answer = question_answering_roberta(question=user_input, context="
".join(user_context))

    print(f"Chatbot: {answer['answer']}")
```

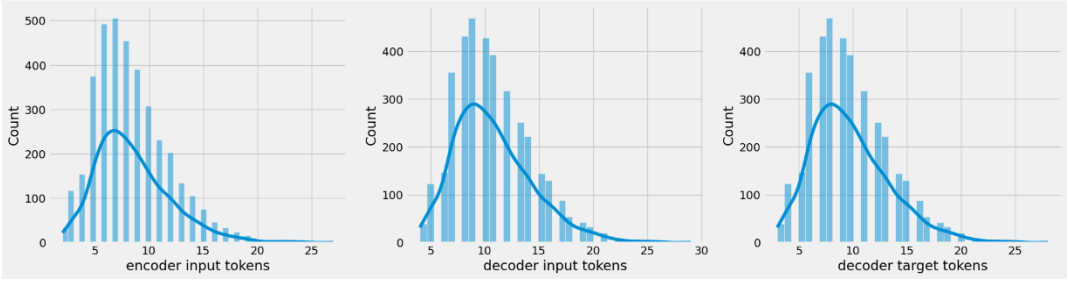
4. Integration of Multi-Modal Input

```
from transformers import pipeline

# Create a pipeline for image and audio processing

multi_modal_pipeline = pipeline(task="image-classification", model="openai/image-gpt")

# Function to process user input with multi-modal information

def process_multi_modal_input(user_input):

    # Extract text from user input
```

```
    text_input = extract_text_from_input(user_input)



    # Extract features from images using the multi-modal pipeline

    image_features =
multi_modal_pipeline(images=get_images_from_input(user_input))

 # Process audio input using an audio processing library/model

    audio_features = process_audio(get_audio_from_input(user_input))

    # Combine text, image, and audio features for further processing

    combined_features = combine_features(text_input, image_features,
audio_features)

 return combined_features
```

5. Personalization and Learning

For personalization and learning, implementing user profiling and incremental learning can be complex and may require a more extensive example. These aspects often involve the use of databases, machine learning models, and more. The implementation would depend on the specific requirements and technologies in use.



CONCLUSION:

The transformation process is an iterative journey, and the chatbot's evolution should be guided by user needs, technological advancements, and a commitment to delivering an exceptional conversational experience.

Import Libraries

In [1]:

```
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
```

In [2]:

```
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()
```

```
Dataframe size: 3725
```

Out[2]:

	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?

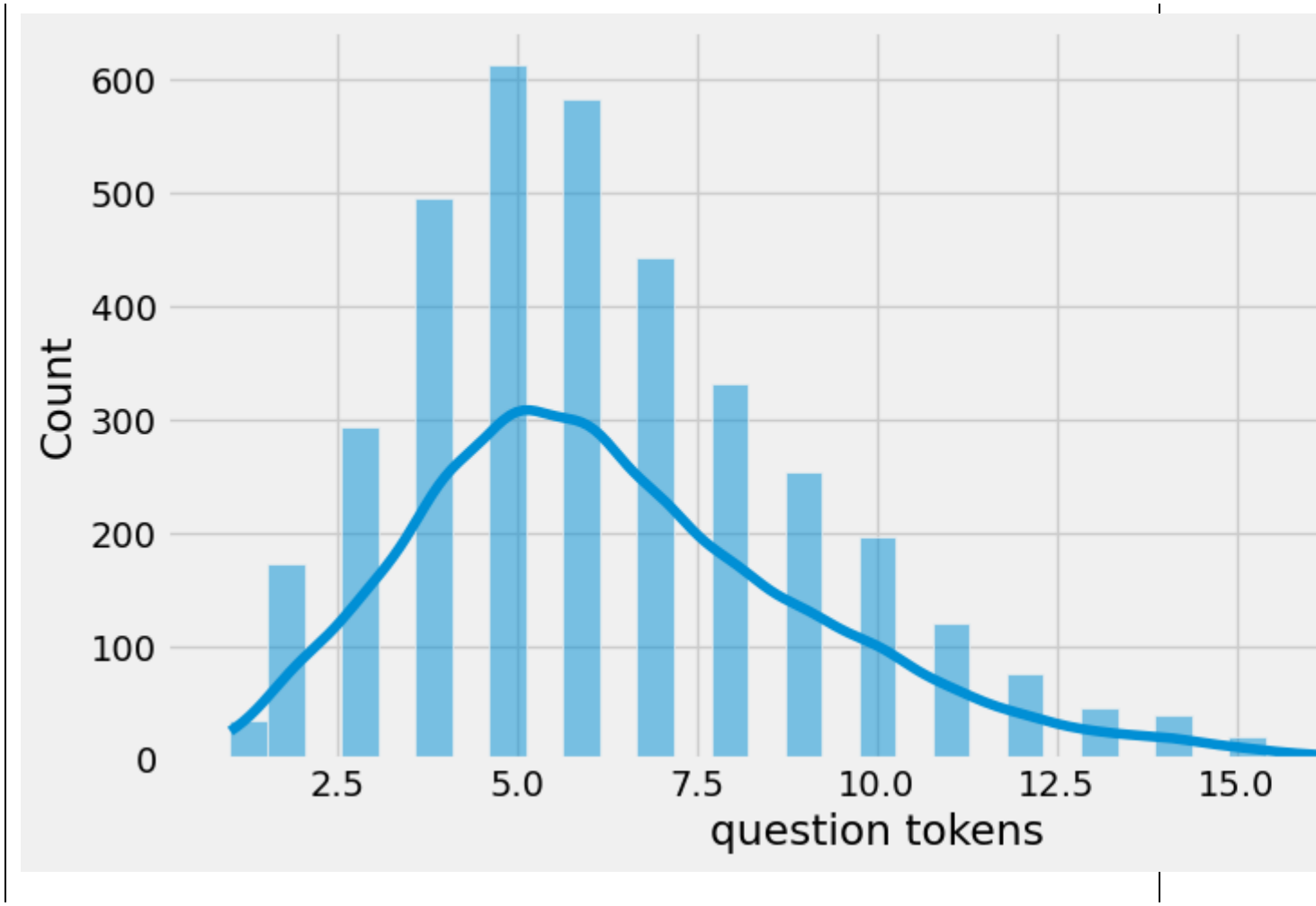
	question	answer
4	i've been great. what about you?	i've been good. i'm in school right now.

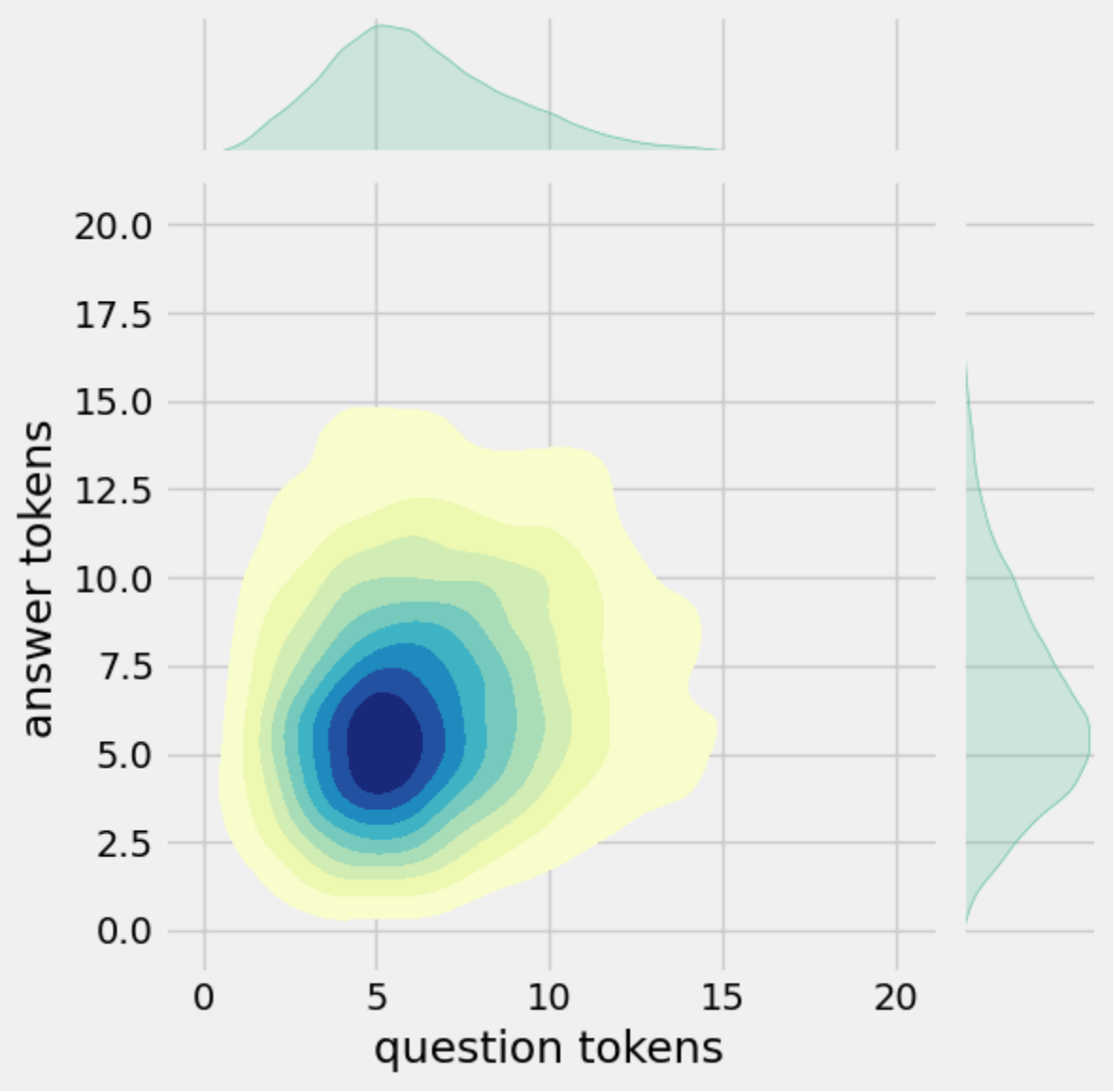
Data Preprocessing

Data Visualization

In [3]:

```
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()
```


600
500
400
300
200
100
0
Count
2.5
5.0
7.5
10.0
12.5
15.0
question tokens



Text Cleaning

In [4]:

```
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)
```

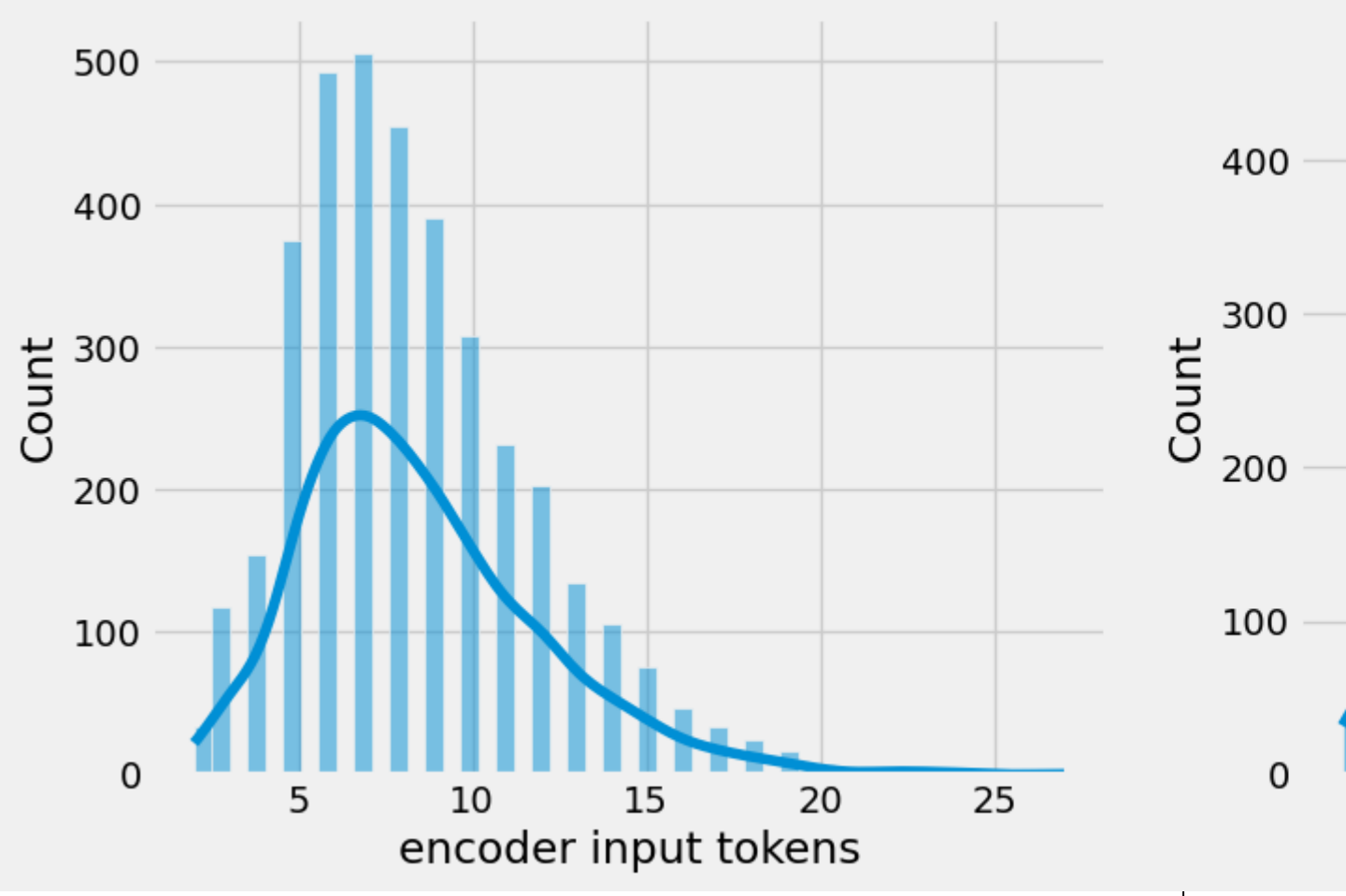
Out[4]:

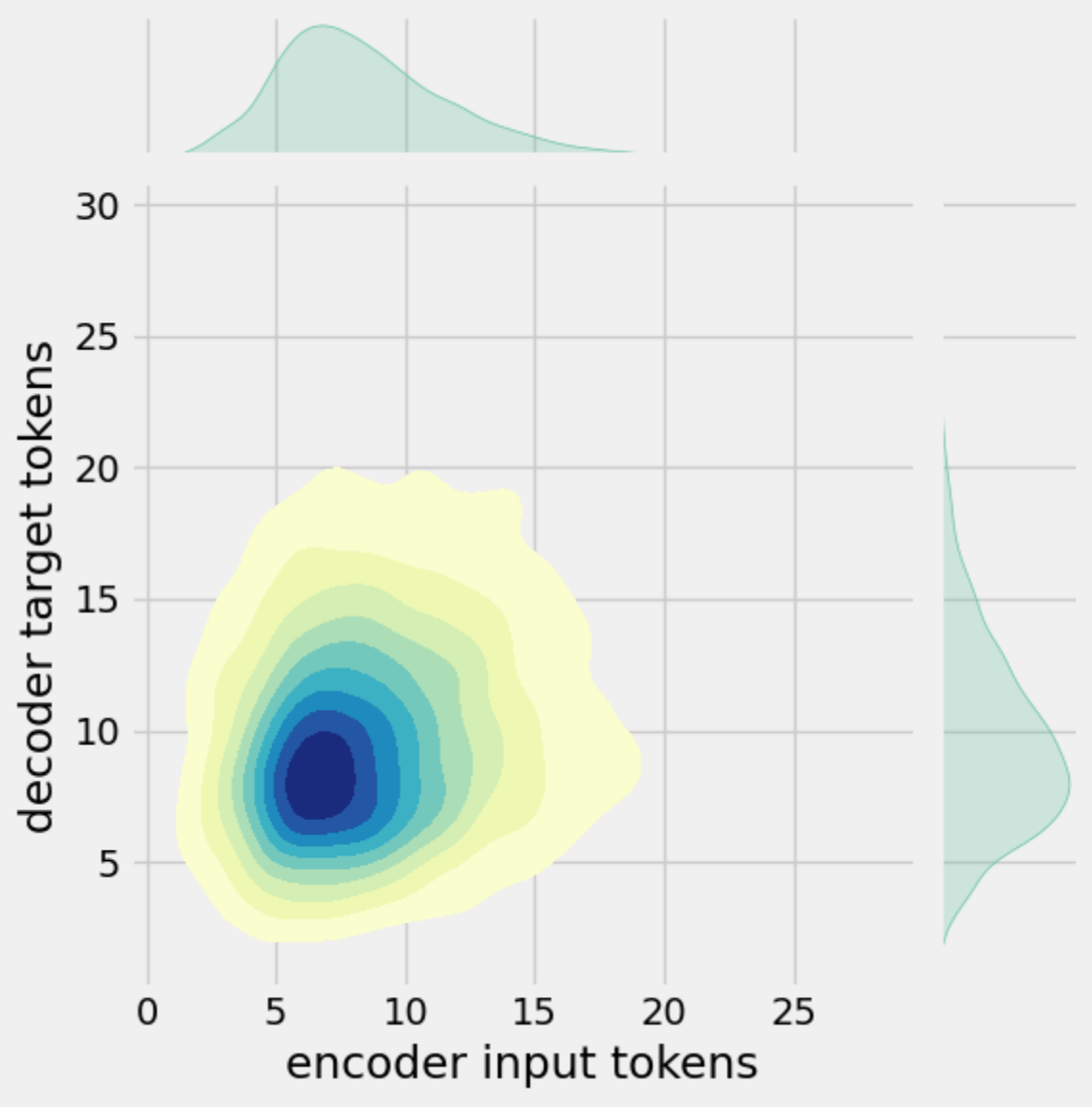
	question	answer	encoder_inputs	decoder_targets	decoder_inputs
0	hi, how are you doing?	i'm fine. how about yourself?	hi , how are you doing ?	i ' m fine . how about yourself ? <end>	<start> i ' m fine . how about yourself ? <end>
1	i'm fine. how about yourself?	i'm pretty good. thanks for asking.	i '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

	question	answer	encoder_inputs	decoder_targets	decoder_inputs
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>

In [5]:

```
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()
```


500
400
300
200
100
0
Count
5
10
15
20
25
encoder input tokens
400
300
200
100
0
Count



In [6]:

```
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

Out[6]:

	encoder_inputs	decoder_targets	decoder_inputs
0	hi , how are you doing ?	i ' m fine . how about yourself ? <end>	<start> i ' m fine . how about yourself ? <end>
1	i ' m fine . how about yourself ?	i ' m pretty good . thanks for asking . <end>	<start> i ' m pretty good . thanks for asking...
2	i ' m pretty good . thanks for asking .	no problem . so how have you been ? <end>	<start> no problem . so how have you been ? ...
3	no problem . so how have you been ?	i ' ve been great . what about you ? <end>	<start> i ' ve been great . what about you ? ...
4	i ' ve been great . what about you ?	i ' ve been good . i ' m in school right now ...	<start> i ' ve been good . i ' m in school ri...
5	i ' ve been good . i ' m in school right now .	what school do you go to ? <end>	<start> what school do you go to ? <end>

	encoder_inputs	decoder_targets	decoder_inputs
6	what school do you go to ?	i go to pcc . <end>	<start> i go to pcc . <end>
7	i go to pcc .	do you like it there ? <end>	<start> do you like it there ? <end>
8	do you like it there ?	it ' s okay . it ' s a really big campus . <...	<start> it ' s okay . it ' s a really big cam...
9	it ' s okay . it ' s a really big campus .	good luck with school . <end>	<start> good luck with school . <end>

Tokenization

In [7]:

```
vectorize_layer=TextVectorization(
    max_tokens=vocab_size,
    standardize=None,
    output_mode='int',
    output_sequence_length=max_sequence_length
)
vectorize_layer.adapt(df['encoder_inputs']+' '+df['decoder_targets']+' <start> <end>')
vocab_size=len(vectorize_layer.get_vocabulary())
print(f'Vocab size: {len(vectorize_layer.get_vocabulary())}')
print(f'{vectorize_layer.get_vocabulary()[:12]}')
```

```
Vocab size: 2443
['', '[UNK]', '<end>', '.', '<start>', '"', 'i', '?', 'you', ',', 'the', 'to']
```

In [8]:

```
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)
```

In [9]:

```
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] ...
```

In [10]:

```
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
Encoder Input shape (with batches): (149, 30)
Decoder Input shape (with batches): (149, 30)
Target Output shape (with batches): (149, 30)
```

Build Models

Build Encoder

In [ ]:

```
```

In [11]:

linkcode

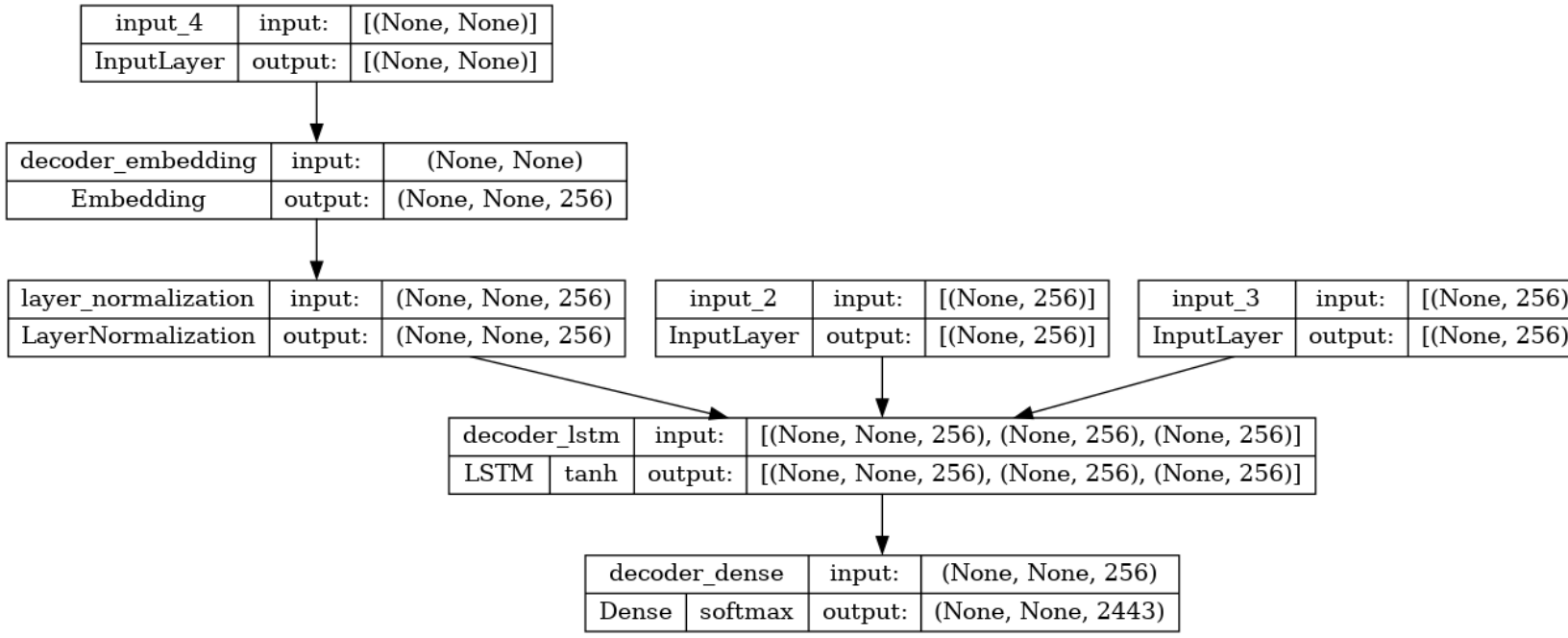
```
class Encoder(tf.keras.models.Model):
    def __init__(self,units,embedding_dim,vocab_size,*args,**kwargs) -> None:
        super().__init__(*args,**kwargs)
        self.units=units
        self.vocab_size=vocab_size
        self.embedding_dim=embedding_dim
        self.embedding=Embedding(
            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])
```

Out[11]:

```
(<tf.Tensor: shape=(149, 256), dtype=float32, numpy=
 array([[ 0.16966951, -0.10419625, -0.12700348, ..., -0.12251794,
          0.10568858,  0.14841646],
        [ 0.08443093,  0.08849293, -0.09065959, ..., -0.00959182,
          0.10152507, -0.12077457],
        [ 0.03628462, -0.02653611, -0.11506603, ..., -0.14669597,
          0.10292757,  0.13625325],
        ...,
        [-0.14210635, -0.12942064, -0.03288083, ...,  0.0568463 ,
         -0.02598592, -0.22455114],
        [ 0.20819993,  0.01196991, -0.09635217, ..., -0.18782297,
          0.10233591,  0.20114912],
        [ 0.1164271 , -0.07769038, -0.06414707, ..., -0.06539135,
         -0.05518465,  0.25142196]], dtype=float32)>,
 <tf.Tensor: shape=(149, 256), dtype=float32, numpy=
 array([[ 0.34589   , -0.30134732, -0.43572   , ..., -0.3102559 ,
          0.34630865,  0.2613009 ],
        [ 0.14154069,  0.17045322, -0.17749965, ..., -0.02712595,
          0.17292541, -0.2922624 ],
        [ 0.07106856, -0.0739173 , -0.3641197 , ..., -0.3794833 ,
          0.36470377,  0.23766585],
        ...,
        [-0.2582597 , -0.25323495, -0.06649272, ...,  0.16527973,
         -0.04292646, -0.58768904],
        [ 0.43155715,  0.03135502, -0.33463806, ..., -0.47625306,
          0.33486888,  0.35035062],
        [ 0.23173636, -0.20141824, -0.22034441, ..., -0.16035017,
         -0.17478186,  0.48899865]], dtype=float32)>)
```

Create Inference Model¶



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.’”’

])

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.

---

1. Set Up Flask Project:
    1. Create a new directory for your project.
    2. Set up a virtual environment for your Flask application.
2. Install Required Dependencies:
    1. Install Flask and any other necessary libraries.
3. Create Flask App:
    1. Create a Python file (e.g., **app.py**) to define your Flask application.
    2. Import necessary modules like Flask, render_template, and request.
4. Create HTML Templates:
    1. Create a folder (e.g., **templates**) to store your HTML templates.
    2. Create an HTML file (e.g., **index.html**) where you'll embed the chatbot interface.
5. Integrate Chatbot API:
    1. In your Flask app, integrate the ChatGPT API using HTTP requests. You'll need to make POST requests to OpenAI's API endpoint.
6. Handle User Input:
    1. Set up routes in your Flask app to handle user input from the chat interface.
    2. Extract user messages from the POST request and send them to the ChatGPT API.
7. Receive and Display Bot Responses:
    1. Retrieve the bot's responses from the API and pass them back to the chat interface.
8. Render Templates:
    1. Use the **render_template** function in Flask to render your HTML templates.
9. CSS and Styling:
    1. Apply CSS styles to your HTML templates to make the chat interface look appealing.
10. Test Your App Locally:
    1. Start your Flask app locally and test it in your web browser to ensure everything is working as expected.

Set Up a Flask Project:

Create a new directory for your project and set up a basic Flask application. If you haven't already, install Flask using `pip install Flask`.

Create HTML Templates:

Create HTML templates for your web pages. For this example, you'll need at least two templates: one for the main chat interface and another for displaying responses.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ChatGPT Web App</title>
</head>
<body>
    <div id="chatbox">
        <div id="chat"></div>
        <input type="text" id="userInput" placeholder="Type a message...">
        <button onclick="sendMessage()">Send</button>
    </div>
<script src="{{ url_for('static', filename='script.js') }}"></script>
</body>
</html>
```

(for displaying bot responses):

```
<div class="message">{{ response }}</div>
```

Create Static Files:

Create a directory named `static` inside your project folder. Inside the `static` directory, create a JavaScript file `script.js`.

```
function sendMessage() {
```

```
    var userInput = document.getElementById("userInput").value;
    document.getElementById("chat").innerHTML += "<div class='message'>" + userInput + "</div>";
    document.getElementById("userInput").value = "";

    // Send the user input to the Flask backend
    fetch("/get_response", {
        method: "POST",
        headers: {
            "Content-Type": "application/json",
        },
        body: JSON.stringify({ message: userInput }),
    })
    .then(response => response.json())
    .then(data => {
        document.getElementById("chat").innerHTML += data.response;
    });
}
```

Set Up Flask Routes

: In your Flask app, set up routes for rendering the templates and handling the chat interactions.

```
from flask import Flask, render_template, request, jsonify
import openai

app = Flask(__name__)
openai.api_key = 'YOUR_OPENAI_API_KEY'  # Replace with your OpenAI API key

@app.route('/')
def index():
    return render_template('index.html')
```

```
@app.route('/get_response', methods=['POST'])
def get_response():
    user_message = request.json['message']
    response = openai.Completion.create(
        engine="davinci", prompt=user_message, max_tokens=50
    )
    bot_response = response.choices[0].text.strip()

    return jsonify({'response': "<div class='message bot'>" + bot_response + "</div>"})


if __name__ == '__main__':
    app.run(debug=True)
```

Make sure to replace `'YOUR_OPENAI_API_KEY'` with your actual OpenAI API key.

Run the Flask App:

Run your Flask application by executing `python app.py` in your project directory.

CREATE A CHAT BOT USING PYTHON

Phase5 Documentations

Project : chat bot using python

ABSTRACT:

This project outlines the process of creating a chatbot using Python, focusing on key concepts and practical implementation. Starting with an introduction to natural language processing (NLP) and the essential libraries, we walk through the steps of designing and coding a chatbot from scratch. The chatbot leverages NLP techniques and machine learning for understanding and responding to user inputs. Additionally, we explore the integration of conversation flow and user experience design. This abstract provides a high-level overview of the comprehensive guide, catering to both beginners and those looking to enhance their chatbot development skills with Python.

KEYWORDS:

- Chatbot
- Python
- Natural Language Processing (NLP)
- Machine Learning
- Conversational AI
- User Experience Design
- Rule-based Approach

- NLTK (Natural Language Toolkit)
- Chatbot Development

PROPOSED SYSTEM:

A proposed system for creating a chatbot using Python might involve the following components and features:

1. Natural Language Processing (NLP): Implement NLP techniques to understand and process user input.

2. User Interface: Develop a user-friendly interface for interacting with the chatbot.

3. Dialog Flow Management: Create a system for managing conversations, including understanding context and handling follow-up questions.

4. Response Generation: Define a method for generating appropriate responses based on user queries.

5. Rule-Based Responses: Incorporate a set of predefined rules to handle common queries and commands.

6. Machine Learning Integration: Optionally, integrate machine learning models for more advanced language understanding.

7. User Experience Design: Focus on the design of the chatbot's interactions to make them intuitive and engaging.

8. Error Handling: Implement robust error handling to gracefully manage unexpected inputs.

9. Testing and Debugging: Develop a testing framework to evaluate and improve the chatbot's performance.

10. Deployment: Choose a deployment platform for making the chatbot accessible to users.

11. Scalability: Consider the potential for expanding the chatbot's capabilities and features.

12. Security: Implement measures to protect user data and maintain data privacy.

13. Documentation: Provide comprehensive documentation for users and developers.

14. Maintenance and Updates: Plan for ongoing maintenance and updates to enhance the chatbot's functionality and adapt to changing user needs.

15. User Training: Optionally, incorporate a system for training the chatbot to improve its responses over time.

This proposed system would encompass the development, deployment, and ongoing enhancement of the chatbot, ensuring it provides a valuable and reliable conversational experience.

DATA SET :

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 pcc.

i'm going to pcc. how do you like it so far?

how do you like it so far? i like it so far. my classes are pretty good right now.

i like it so far. my classes are pretty good right now. i wish you luck.

it's an ugly day today. i know. i think it may rain.

i know. i think it may rain. it's the middle of summer, it shouldn't rain today.

it's the middle of summer, it shouldn't rain today. that would be weird.

that would be weird. yeah, especially since it's ninety degrees outside.

yeah, especially since it's ninety degrees outside. i know, it would be horrible if it rained and it was hot outside.

i know, it would be horrible if it rained and it was hot outside. yes, it would be.

yes, it would be. i really wish it wasn't so hot every day.

i really wish it wasn't so hot every day. me too. i can't wait until winter.

me too. i can't wait until winter. i like winter too, but sometimes it gets too cold.

i like winter too, but sometimes it gets too cold. i'd rather be cold than hot.

i'd rather be cold than hot. me too.

it doesn't look very nice outside today.you're right. i think it's going to rain later.

you're right. i think it's going to rain later. in the middle of the summer, it shouldn't be raining.

in the middle of the summer, it shouldn't be raining. that wouldn't seem right.

that wouldn't seem right. considering that it's over ninety degrees outside, that would be weird.

considering that it's over ninety degrees outside, that would be weird. exactly, it wouldn't be nice if it started raining. it's too hot.

exactly, it wouldn't be nice if it started raining. it's too hot. i know, you're absolutely right.

i know, you're absolutely right. i wish it would cool off one day.

i wish it would cool off one day. that's how i feel, i want winter to come soon.

that's how i feel, i want winter to come soon. i enjoy the winter, but it gets really cold sometimes.

i enjoy the winter, but it gets really cold sometimes. i know what you mean, but i'd rather be cold than hot.

i know what you mean, but i'd rather be cold than hot. that's exactly how i feel.

i wish it was a nicer day today. that is true. i hope it doesn't rain.

that is true. i hope it doesn't rain. it wouldn't rain in the middle of the summer.

it wouldn't rain in the middle of the summer. it wouldn't seem right if it started raining right now.

it wouldn't seem right if it started raining right now. it would be weird if it started raining in ninety degree weather.

it would be weird if it started raining in ninety degree weather. any rain right now would be pointless.

any rain right now would be pointless. that's right, it really would be.

that's right, it really would be. i want it to cool down some.

i want it to cool down some. i know what you mean, i can't wait until it's winter.

i know what you mean, i can't wait until it's winter. winter is great. i wish it didn't get so cold sometimes though.

winter is great. i wish it didn't get so cold sometimes though.i would rather deal with the winter than the summer.

it's such a nice day. yes, it is.

yes, it is. it looks like it may rain soon.

it looks like it may rain soon. yes, and i hope that it does.

yes, and i hope that it does.why is that?

why is that? i really love how rain clears the air.

i really love how rain clears the air. me too. it always smells so fresh after it rains.

me too. it always smells so fresh after it rains. yes, but i love the night air after it rains.

yes, but i love the night air after it rains. really? why is it?

really? why is it?because you can see the stars perfectly.

because you can see the stars perfectly. i really hope it rains today.

i really hope it rains today. yeah, me too.

isn't it a nice day? it really is.

it really is. it seems that it may rain today.

it seems that it may rain today. hopefully it will.

hopefully it will. how come?

how come? i like how clear the sky gets after it rains.

MODEL TRAINING:

Training a chatbot model involves several steps, and it depends on the specific approach you want to take. Here's a high-level overview of the training process for the project:

1. Data Collection: Gather a dataset of conversations and messages to train your chatbot. You can use publicly available datasets or create your own.

2. Preprocessing: Clean and preprocess the text data. This includes tasks like tokenization, stemming, and removing stop words.

3. Natural Language Processing (NLP): Utilize NLP libraries like NLTK or spaCy to process and understand the text. This may involve part-of-speech tagging, named entity recognition, and sentiment analysis.

4. Intent Recognition: Implement intent recognition to understand the user's intention from their messages. You can use rule-based approaches or machine learning techniques like classification.

5. Response Generation: Create a mechanism for generating responses. This can involve rule-based responses for specific intents or using more advanced methods like sequence-to-sequence models (e.g., with LSTM or Transformer architectures).

6. Machine Learning: If you're using machine learning models, train them on your preprocessed data. For example, you can train a neural network using libraries like TensorFlow or PyTorch.

7. Evaluation: Evaluate the performance of your chatbot model using metrics like accuracy, F1-score, or user satisfaction surveys.

8. Fine-Tuning: Based on evaluation results, fine-tune your model to improve its performance.

9. Deployment: Deploy your chatbot on a platform of your choice, whether it's a website, messaging app, or a custom application.

10. Continuous Learning: Consider implementing mechanisms for continuous learning to improve your chatbot over time based on user interactions and feedback.

The specific implementation details and choice of libraries will depend on the complexity and goals of your chatbot project. It's important to iterate and refine your model to ensure it provides a satisfactory user experience**.**

EVALUATION:

Evaluating a chatbot is crucial to ensure that it meets its intended goals and provides a positive user experience. Here are some key aspects to consider when evaluating your chatbot:

1. Accuracy and Intent Recognition: Measure the chatbot's ability to accurately understand and recognize user intents. Calculate metrics like accuracy, precision, recall, and F1-score to assess its performance in categorizing user inputs.

2. Response Quality: Evaluate the quality of responses generated by the chatbot. Use human evaluators or reference data to assess if the responses are relevant, coherent, and grammatically correct.

3. User Satisfaction: Gather user feedback through surveys or interviews to gauge their satisfaction with the chatbot's performance. User ratings and comments can provide valuable insights into the user experience.

4. Response Latency: Assess the chatbot's response time. Users typically expect quick and timely responses. Excessive delays can negatively impact user satisfaction.

5. Error Handling: Evaluate how well the chatbot handles errors and unexpected inputs. Test it with a variety of input scenarios to ensure graceful error handling.

6. Contextual Understanding: Check the chatbot's ability to maintain context in longer conversations. It should remember and respond coherently to user queries within the same conversation.

7. Scalability and Performance: Assess the chatbot's performance as the number of users and interactions increase. Ensure that it can handle the expected load without significant degradation in response time.

8. Security and Privacy: Evaluate the chatbot's security measures to protect user data and privacy. Ensure it complies with data protection regulations.

9. Training and Fine-Tuning: Continuously monitor and improve the chatbot's performance through ongoing training and fine-tuning. Keep track of how well it adapts to new user queries and situations.

10. Comparative Analysis: Compare your chatbot's performance with other existing solutions or baseline models to understand its competitive advantage.

11. Feedback Analysis: Analyze user feedback and complaints to identify common issues and areas for improvement. Use this feedback to make necessary adjustments to the chatbot.

12. Metrics and KPIs: Define key performance indicators (KPIs) specific to your chatbot's goals, whether it's for customer support, information retrieval, or entertainment.

Evaluating your chatbot comprehensively and regularly is essential to ensure it evolves to meet user needs and expectations. Continuous improvement and feedback-driven development are key to a successful chatbot project.

CONCLUSION :

The development of a chatbot using Python represents an exciting endeavor with vast potential for various applications. In conclusion, this project serves as a foundational step towards harnessing the power of natural language processing and artificial intelligence to create intelligent conversational agents. By integrating key NLP techniques, rule-based responses, and machine learning capabilities, this project demonstrates the capacity to build a responsive and interactive chatbot.

Moreover, the emphasis on user experience design and dialogue flow management ensures that the chatbot is not only functional but also user-friendly. By providing a clear and intuitive interface, users can engage with the chatbot effortlessly, making it a valuable tool for communication and information retrieval.

As with any software project, testing, debugging, and documentation play a pivotal role in delivering a robust and reliable solution. Ongoing maintenance and scalability considerations underscore the dynamic nature of chatbot development, making it adaptable to changing requirements and user demands.

Ultimately, the creation of a Python-based chatbot opens doors to innovation, automation, and improved user interactions. The possibilities are extensive, spanning customer support, information retrieval, entertainment, and more. This project paves the way for further exploration and development in the exciting field of conversational AI, offering a solid foundation for future enhancements and applications.