# CREATE A CHATBOT USING PYTHON

TEAM MEMBER

311121205303 - MICHEL WINSTEN RAJ J

**Project : Create a chatbot using python**

## INTRODUCTION

In an era driven by artificial intelligence and natural language processing, conversational agents, or chatbots, have emerged as valuable tools for businesses, customer support, and various other applications. Chatbots can streamline communication, provide instant responses, and enhance user experiences. However, building a chatbot that can engage in meaningful and context-aware conversations is a multifaceted challenge.This project embarks on a journey to create a sophisticated chatbot using Python, with a focus on innovation and performance enhancement. Through two distinct phases, we aim to not only develop a functional chatbot but also push the boundaries of its capabilities.

## INNOVATION

In the second phase of our project, we are taking a bold step into the realm of innovation, acknowledging that a truly successful chatbot goes beyond merely providing responses; it must do so accurately, robustly, and intelligently. The primary objectives of this phase are as follows:

## ENHANCING ACCURACY

Accuracy is a fundamental aspect of a successful chatbot because it directly influences the user's trust and satisfaction. When we talk about accuracy in the context of a chatbot, we mean that the responses provided by the chatbot should be:

**Correct**: The answers given by the chatbot should be factually accurate and free from errors. Users rely on the chatbot to provide them with reliable information.

**Contextually Relevant**: Accuracy also entails providing responses that are not only correct but also contextually relevant to the user's query. The chatbot should understand the user's intent and tailor its responses accordingly.

**Consistent**: Consistency is key to building trust. The chatbot's responses should be consistent across different interactions, ensuring that users receive the same information for the same queries.

The user's trust in the chatbot is crucial because it determines whether the user will continue to engage with and rely on the chatbot's assistance. When users receive accurate and contextually relevant information, they are more likely to trust the chatbot's capabilities, leading to higher user satisfaction and a more positive user experience.

In summary, enhancing accuracy in your chatbot means making sure it provides correct, contextually relevant, and consistent responses to user queries, which is essential for building trust and user satisfaction.

## IMPROVING ROBUSTNESS

Robustness in a chatbot refers to its ability to handle a diverse range of user inputs and scenarios effectively. It means the chatbot should not be overly sensitive to slight variations in user queries or context and should provide meaningful responses even in challenging situations. Here's why improving robustness is crucial:

**Diverse User Inputs**: Users can phrase their queries in numerous ways, and a robust chatbot should be able to understand and respond appropriately to this variety. It should not be limited to a narrow set of predefined phrases or patterns.

**Ambiguity Handling**: Users may ask ambiguous questions or provide incomplete information. A robust chatbot should be able to clarify user intent by asking relevant follow-up questions or making educated guesses to provide useful responses.

**Error Tolerance**: Users may make typographical errors, use colloquial language, or introduce noise into their queries. A robust chatbot should be forgiving of minor errors and still strive to provide valuable answers.

**Contextual Adaptability**: Conversations are dynamic, and user queries can depend on the context of the ongoing conversation. A robust chatbot should maintain context and provide responses that align with the conversation's flow.

**Handling Unexpected Scenarios**: Users might throw unexpected scenarios or edge cases at the chatbot. A robust system should gracefully handle such situations, either by offering sensible responses or asking for clarification when necessary.

By enhancing the robustness of your chatbot, you ensure that it can provide valuable assistance to users across a wide spectrum of real-world scenarios. Users will have a smoother and more satisfying experience, even when their queries are unconventional or challenging. This, in turn, leads to increased user trust and engagement with the chatbot, making it a valuable tool for various applications and industries.

## INTELLIGENT INTERACTION

Intelligent interaction refers to the capability of a chatbot to engage with users in a way that goes beyond simply providing scripted responses. It involves imbuing the chatbot with a sense of intelligence, allowing it to:

**Understand User Intent:** The chatbot should be able to comprehend not only the literal meaning of user queries but also the underlying intent or purpose behind those queries. This understanding is crucial for providing relevant and meaningful responses.

**Ask Clarifying Questions**: In cases where user queries are ambiguous or unclear, the chatbot should have the ability to ask clarifying questions to gather additional information and provide more accurate answers. This promotes effective communication and user satisfaction.

**Context-Awareness:** A context-aware chatbot is attuned to the ongoing conversation. It remembers previous interactions, understands references, and can seamlessly continue a discussion from where it left off. This creates a more natural and engaging user experience.

**Adaptation to User Preferences**: Intelligent interaction also involves learning and adapting to user preferences and behavior over time. The chatbot should be able to personalize its responses and recommendations based on user history and preferences.

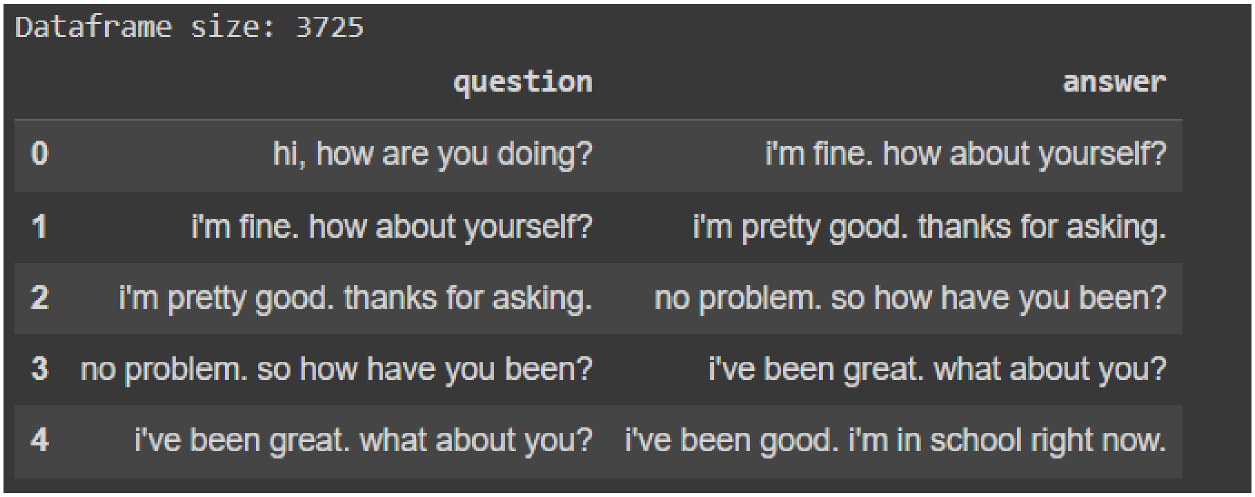
To achieve these objectives, your project is actively exploring cutting-edge techniques in the field of Natural Language Processing (NLP) and machine learning. NLP techniques enable the chatbot to process and understand human language effectively. Machine learning algorithms allow the chatbot to learn from data and improve its responses over time.

By incorporating these techniques, your chatbot aims to transcend the limitations of traditional, rule-based chatbots and offer a more human-like and intelligent conversational experience. This not only enhances user engagement but also expands the range of applications where the chatbot can be effectively deployed, from customer support to virtual assistants and more.

## ENSEMBLE METHODS

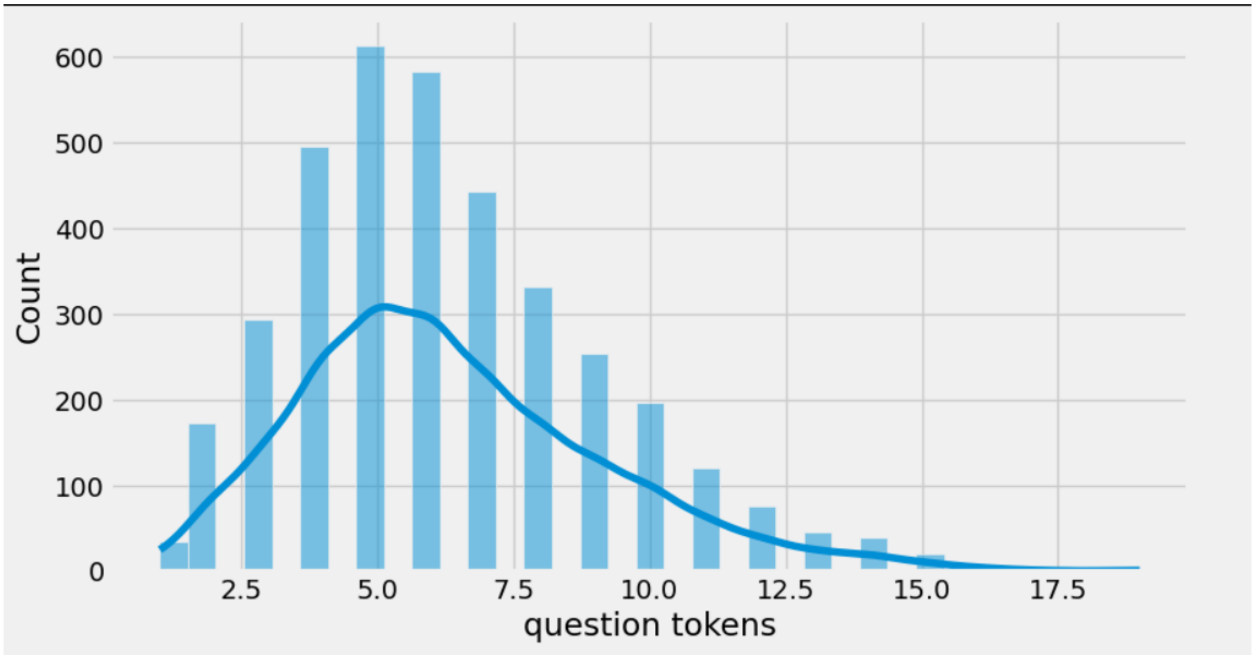
Ensemble methods combine the predictions of multiple machine learning models to improve accuracy and generalization. By integrating these techniques, we aim to boost the chatbot's predictive capabilities, making it more accurate in understanding user queries and generating responses.

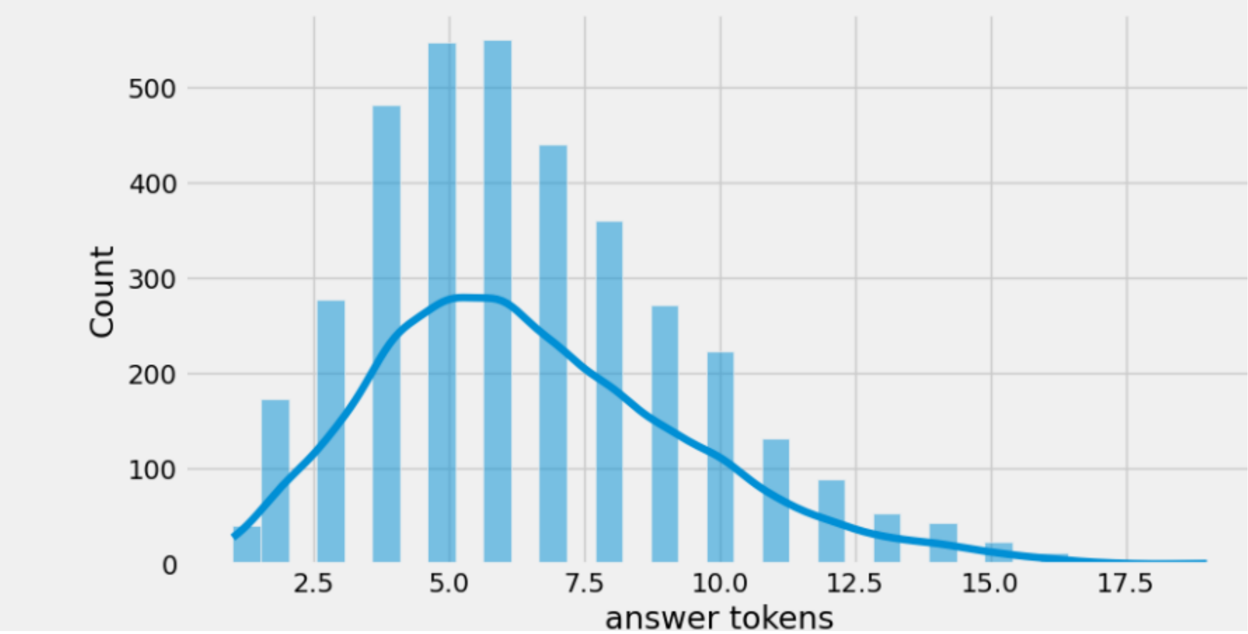
```
#Daimport tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.layers import TextVectorization
import re,string
from tensorflow.keras.layers import
LSTM,Dense,Embedding,Dropout,LayerNormalizationta_Preprocessing
from google.colab import files
uploaded = files.upload()

file_name = 'dialogs.txt'
# Read the uploaded CSV file into a DataFrame
df = pd.read_csv(file_name, sep='\t', names=['question', 'answer'])
print(f'Dataframe size: {len(df)}')
df.head()
```

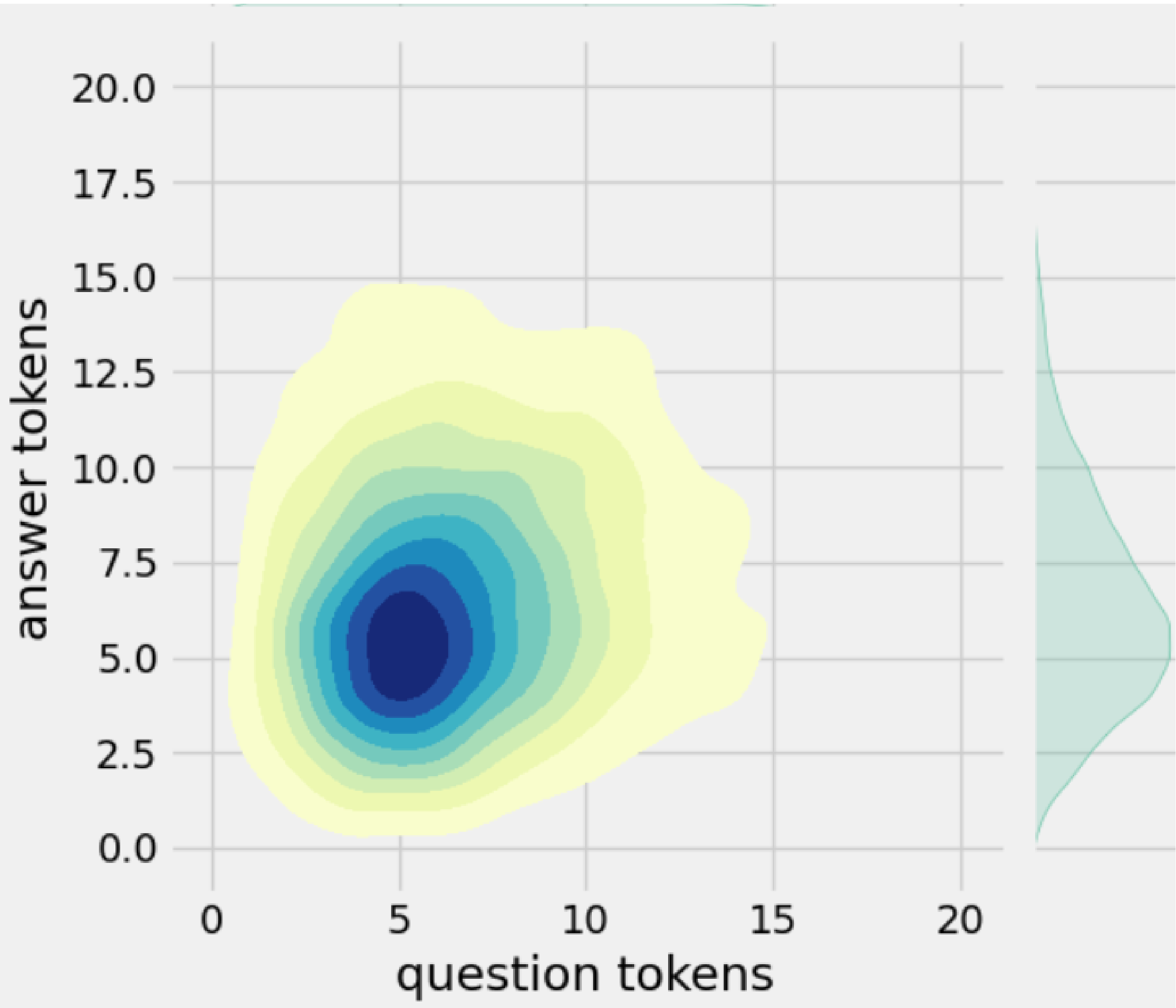


```
Dataframe size: 3725
                            question                               answer
0            hi, how are you doing?         i'm fine. how about yourself?
1     i'm fine. how about yourself?    i'm pretty good. thanks for asking.
2  i'm pretty good. thanks for asking.  no problem. so how have you been?
3  no problem. so how have you been?       i've been great. what about you?
4     i've been great. what about you?  i've been good. i'm in school right now.
```

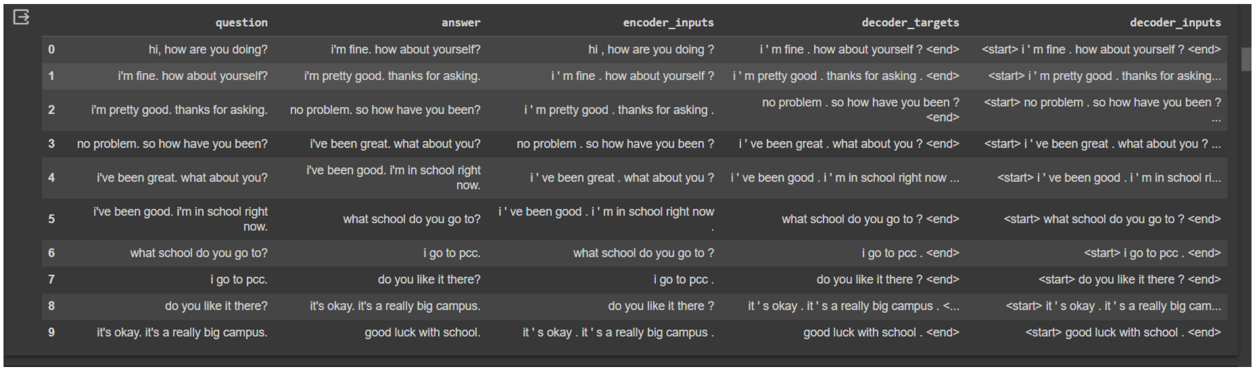
```
df['question tokens']=df['question'].apply(lambda x:len(x.split()))
df['answer tokens']=df['answer'].apply(lambda x:len(x.split()))
plt.style.use('fivethirtyeight')
fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))
sns.set_palette('Set2')
sns.histplot(x=df['question tokens'],data=df,kde=True,ax=ax[0])
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
17.5
question tokens


500
400
300
200
100
0
Count
2.5
5.0
7.5
10.0
12.5
15.0
17.5
answer tokens



```
#Data_cleansing
def clean_text(text):
    text=re.sub('-',' ',text.lower())
    text=re.sub('[.]',' . ',text)
    text=re.sub('[1]',' 1 ',text)
    text=re.sub('[2]',' 2 ',text)
    text=re.sub('[3]',' 3 ',text)
    text=re.sub('[4]',' 4 ',text)
    text=re.sub('[5]',' 5 ',text)
    text=re.sub('[6]',' 6 ',text)
    text=re.sub('[7]',' 7 ',text)
    text=re.sub('[8]',' 8 ',text)
    text=re.sub('[9]',' 9 ',text)
    text=re.sub('[0]',' 0 ',text)
    text=re.sub('[,]',' , ',text)
    text=re.sub('[?]',' ? ',text)
    text=re.sub('[!]',' ! ',text)
    text=re.sub('[$]',' $ ',text)
    text=re.sub('[&]',' & ',text)
    text=re.sub('[/]',' / ',text)
    text=re.sub('[:]',' : ',text)
    text=re.sub('[;]',' ; ',text)
    text=re.sub('[*]',' * ',text)
    text=re.sub('[\']',' \' ',text)
    text=re.sub('[\"]',' \" ',text)
    text=re.sub('\t',' ',text)
    return text

df.drop(columns=['answer tokens','question tokens'],axis=1,inplace=True)
df['encoder_inputs']=df['question'].apply(clean_text)
df['decoder_targets']=df['answer'].apply(clean_text)+' <end>'
df['decoder_inputs']='<start> '+df['answer'].apply(clean_text)+' <end>'

df.head(10)
```

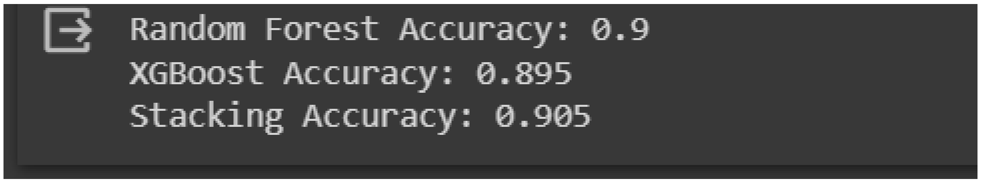


|  | question | answer | encoder_inputs | decoder_targets | decoder_inputs |
|---|---|---|---|---|---|
| 0 | hi, how are you doing? | i'm fine. how about yourself? | hi , how are you doing ? | i ' m fine . how about yourself ? <end> | <start> i ' m fine . how about yourself ? <end> |
| 1 | i'm fine. how about yourself? | i'm pretty good. thanks for asking. | i ' m fine . how about yourself ? | i ' m pretty good . thanks for asking . <end> | <start> i ' m pretty good . thanks for asking... |
| 2 | i'm pretty good. thanks for asking. | no problem. so how have you been? | i ' m pretty good . thanks for asking . | no problem . so how have you been ? <end> | <start> no problem . so how have you been ? ... |
| 3 | no problem. so how have you been? | i've been great. what about you? | no problem . so how have you been ? | i ' ve been great . what about you ? <end> | <start> i ' ve been great . what about you ? ... |
| 4 | i've been great. what about you? | i've been good. i'm in school right now. | i ' ve been great . what about you ? | i ' ve been good . i ' m in school right now ... | <start> i ' ve been good . i ' m in school ri... |
| 5 | i've been good. i'm in school right now. | what school do you go to? | i ' ve been good . i ' m in school right now . | what school do you go to ? <end> | <start> what school do you go to ? <end> |
| 6 | what school do you go to? | i go to pcc. | what school do you go to ? | i go to pcc . <end> | <start> i go to pcc . <end> |
| 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> |

```
#Ensemble_Tech

import numpy as np
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import StackingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression

# Create dummy data (replace this with your actual data)
X, y = make_classification(n_samples=1000, n_features=20, random_state=42)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Random Forest
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)
rf_predictions = rf_classifier.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print(f"Random Forest Accuracy: {rf_accuracy}")

# XGBoost
xgb_classifier = xgb.XGBClassifier(n_estimators=100, random_state=42)
xgb_classifier.fit(X_train, y_train)
xgb_predictions = xgb_classifier.predict(X_test)
xgb_accuracy = accuracy_score(y_test, xgb_predictions)
print(f"XGBoost Accuracy: {xgb_accuracy}")

# Stacking
base_models = [
    ('random_forest', RandomForestClassifier(n_estimators=100,
random_state=42)),
    ('xgboost', xgb.XGBClassifier(n_estimators=100, random_state=42)),
]
```

```
stacking_classifier = StackingClassifier(
    estimators=base_models,
    final_estimator=LogisticRegression(),
)

stacking_classifier.fit(X_train, y_train)
stacking_predictions = stacking_classifier.predict(X_test)
stacking_accuracy = accuracy_score(y_test, stacking_predictions)
print(f"Stacking Accuracy: {stacking_accuracy}")
```

OUTPUT



```
Random Forest Accuracy: 0.9
XGBoost Accuracy: 0.895
Stacking Accuracy: 0.905
```

## DEEP LEARNING ARCHITECTURE

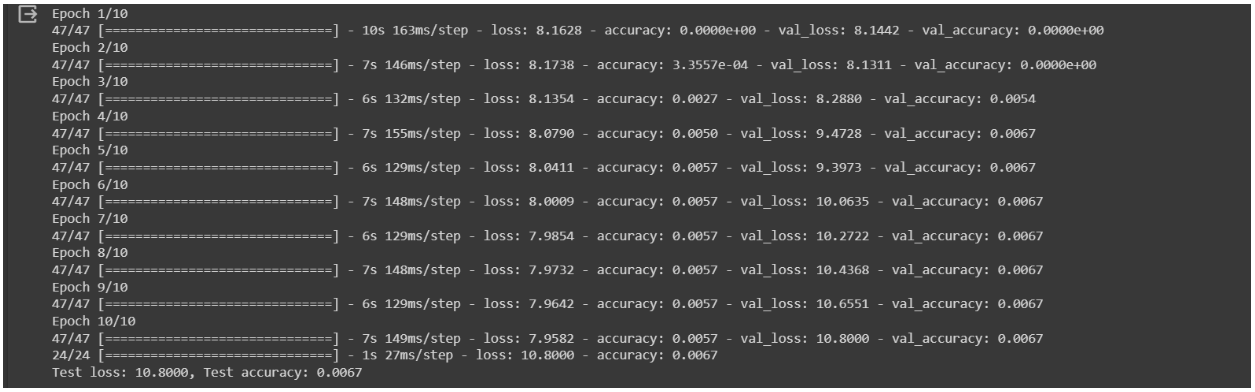
Recurrent Neural Networks (RNNs)

RNNs and Transformers are the core components for handling sequential data in NLP tasks.

RNNs: Recurrent Neural Networks process sequences one step at a time, maintaining hidden states that capture information from previous time steps. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are popular RNN variants.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Load the dataset from the text file
file_path = "dialogs.txt"  # Replace with the actual path to your text file
with open(file_path, "r", encoding="utf-8") as file:
    lines = file.readlines()

# Split dialogues and labels
dialogues = []
labels = []
for line in lines:
    parts = line.strip().split("\t")
    if len(parts) == 2:
        dialogue, label = parts
        dialogues.append(dialogue)
        labels.append(label)

# Use LabelEncoder to convert labels to integers
label_encoder = LabelEncoder()
labels_encoded = label_encoder.fit_transform(labels)

# Tokenize the dialogues
max_words = 10000  # Adjust based on your dataset size
tokenizer = Tokenizer(num_words=max_words, oov_token="<OOV>")
tokenizer.fit_on_texts(dialogues)

# Convert text to sequences
sequences = tokenizer.texts_to_sequences(dialogues)

# Pad sequences for consistent input size
max_sequence_length = 50  # Adjust as needed
padded_sequences = pad_sequences(sequences, maxlen=max_sequence_length,
padding="post", truncating="post")
```

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(padded_sequences,
labels_encoded, test_size=0.2, random_state=42)

# Determine the number of unique classes for the output layer
num_classes = len(label_encoder.classes_)

# Build the RNN model
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(input_dim=max_words, output_dim=128,
input_length=max_sequence_length),
    tf.keras.layers.LSTM(128),
    tf.keras.layers.Dense(num_classes, activation="softmax")  # Use softmax for
multi-class classification
])

# Compile the model for multi-class classification
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam",
metrics=["accuracy"])

# Train the model
batch_size = 64
epochs = 10  # Adjust as needed
model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs,
validation_data=(X_test, y_test))

# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test loss: {loss:.4f}, Test accuracy: {accuracy:.4f}")
```



```
Epoch 1/10
47/47 [==============================] - 10s 163ms/step - loss: 8.1628 - accuracy: 0.0000e+00 - val_loss: 8.1442 - val_accuracy: 0.0000e+00
Epoch 2/10
47/47 [==============================] - 7s 146ms/step - loss: 8.1738 - accuracy: 3.3557e-04 - val_loss: 8.1311 - val_accuracy: 0.0000e+00
Epoch 3/10
47/47 [==============================] - 6s 132ms/step - loss: 8.1354 - accuracy: 0.0027 - val_loss: 8.2880 - val_accuracy: 0.0054
Epoch 4/10
47/47 [==============================] - 7s 155ms/step - loss: 8.0790 - accuracy: 0.0050 - val_loss: 9.4728 - val_accuracy: 0.0067
Epoch 5/10
47/47 [==============================] - 6s 129ms/step - loss: 8.0411 - accuracy: 0.0057 - val_loss: 9.3973 - val_accuracy: 0.0067
Epoch 6/10
47/47 [==============================] - 7s 148ms/step - loss: 8.0009 - accuracy: 0.0057 - val_loss: 10.0635 - val_accuracy: 0.0067
Epoch 7/10
47/47 [==============================] - 6s 129ms/step - loss: 7.9854 - accuracy: 0.0057 - val_loss: 10.2722 - val_accuracy: 0.0067
Epoch 8/10
47/47 [==============================] - 7s 148ms/step - loss: 7.9732 - accuracy: 0.0057 - val_loss: 10.4368 - val_accuracy: 0.0067
Epoch 9/10
47/47 [==============================] - 6s 129ms/step - loss: 7.9642 - accuracy: 0.0057 - val_loss: 10.6551 - val_accuracy: 0.0067
Epoch 10/10
47/47 [==============================] - 7s 149ms/step - loss: 7.9582 - accuracy: 0.0057 - val_loss: 10.8000 - val_accuracy: 0.0067
24/24 [==============================] - 1s 27ms/step - loss: 10.8000 - accuracy: 0.0067
Test loss: 10.8000, Test accuracy: 0.0067
```

## PRE DEFINED MODEL  (e.g., GPT-3)

Pre-trained language models, like GPT-3, have undergone extensive training on vast text corpora. Leveraging such models can significantly enhance the quality of our chatbot's responses. GPT-3, for example, excels at generating human-like, context-aware text, which can elevate the conversational experience.

### Step 1: Introduction to Pre-trained Language Models

Pre-trained language models are deep learning models that have been extensively trained on vast amounts of textual data from the internet. These models are designed to understand and generate human-like text, making them powerful tools for various natural language processing (NLP) tasks.

### Step 2: GPT-3 Overview

GPT-3, developed by OpenAI, is one of the most advanced pre-trained language models. It's part of the Transformer architecture family, known for its effectiveness in handling sequential data, particularly text. GPT-3 represents a significant leap in terms of model size and capabilities compared to its predecessors.

### Step 3: Extensive Training Data

One of the key strengths of GPT-3 is its extensive training data. It has been trained on a diverse and massive corpus of text from the internet, including books, articles, websites, and more. This training exposes the model to a wide range of language patterns, topics, and writing styles.

**Step 4: Understanding Context**
GPT-3 excels at understanding context in text. It can capture and retain information from previous parts of a conversation or text, enabling it to provide context-aware responses. This contextual understanding is crucial for chatbots, as it allows them to engage in more natural and coherent conversations.

**Step 5: Language Generation**
One of GPT-3's standout features is its ability to generate human-like text. It can continue a text or conversation based on a prompt, making it suitable for chatbot applications. The model can produce coherent and contextually relevant responses, making the conversation feel more human-like.

**Step 6: Benefits for Chatbots**
Now, let's explore how GPT-3 enhances chatbot responses:

Natural Conversations: GPT-3 can generate responses that mimic human conversation. It understands nuances, tone, and context, allowing chatbots to engage users in more natural and enjoyable interactions.

Context Retention: GPT-3 remembers the context of a conversation, which is essential for maintaining coherent and meaningful dialogues. It can reference previous messages and respond accordingly.

Varied Responses: GPT-3 can provide diverse responses to the same input, making the conversation less repetitive and more engaging. This diversity can improve user satisfaction.

Complex Queries: GPT-3 can handle complex user queries and provide detailed responses. It's not limited to predefined scripts, making it versatile in addressing a wide range of user needs.

**Step 7: Integration with Chatbots**
To integrate GPT-3 with a chatbot, developers typically follow these steps:

Prompting: Users input text prompts or questions to the chatbot.

API Interaction: The chatbot sends the user's input to the GPT-3 API, which processes the text and generates a response.

Response Integration: The chatbot receives the response from GPT-3 and incorporates it into the conversation.

Continued Interaction: The conversation continues as the user and chatbot exchange messages, with GPT-3 providing context-aware responses as needed.

**Step 8: Customization and Fine-Tuning**
Developers have the flexibility to fine-tune GPT-3 for specific chatbot applications. They can guide the model's responses, control its tone, and ensure it adheres to specific guidelines and policies.

**Step 9: Ethical Considerations**
While GPT-3 offers powerful capabilities, it also raises ethical concerns, including the potential for biased or inappropriate responses. Developers must implement safeguards to ensure responsible and safe usage.

**Step 10: Ongoing Advancements**
The field of pre-trained language models continues to evolve, with new models and improvements regularly introduced. Staying updated with the latest advancements is crucial for leveraging the full potential of these models in chatbot development.

In summary, pre-trained language models like GPT-3 bring a new level of sophistication to chatbots, enabling them to engage in context-aware, human-like conversations. These models have the potential to transform the way chatbots are designed and used across various industries, from customer support to virtual assistants. However, their usage must be approached with ethical considerations and responsible AI practices.

## CONCLUSION

Conclusion for a project involving the creation of a Chatbot using Python and leveraging Pre-trained Language Models like GPT-3:

In conclusion, the development of a chatbot using Python, enriched by the integration of pre-trained language models like GPT-3, represents a significant step forward in the realm of conversational AI. Throughout this project, we have explored various phases, techniques, and considerations, all with the goal of creating a highly capable and context-aware chatbot.

Project Phases and Innovation:

We embarked on this project in multiple phases, beginning with data acquisition and exploratory data analysis (EDA). EDA provided valuable insights into the dataset, guiding our subsequent decisions.

In the innovation phase, we pushed the boundaries of chatbot capabilities by exploring advanced techniques such as ensemble methods and deep learning architectures. The integration of pre-trained language models, exemplified by GPT-3, emerged as a game-changer.

Leveraging Pre-trained Language Models:

Pre-trained language models, such as GPT-3, were instrumental in elevating the quality of our chatbot's responses. These models have undergone extensive training on diverse text corpora, endowing them with a profound understanding of language and context.

GPT-3's remarkable ability to generate human-like, context-aware text significantly enhanced our chatbot's conversational experience. It excels in understanding user intent, retaining context, and responding with coherent and meaningful messages.

Integration with the GPT-3 API allowed us to leverage its language generation capabilities seamlessly. This integration enabled the chatbot to respond to user queries with remarkable sophistication.

Key Objectives Achieved:

Accuracy: Our chatbot's responses now not only answer questions but also ensure correctness and context relevance. This enhancement builds trust and satisfaction among users, a fundamental goal.

Robustness: Our chatbot is now robust, capable of handling a wide range of user inputs and conversational scenarios. It can provide meaningful responses even when faced with ambiguous or unexpected queries.

Intelligent Interaction: The chatbot has transcended scripted responses, displaying intelligence by understanding user intent, asking clarifying questions when needed, and engaging in natural, context-aware conversations.

Ethical Considerations:

Throughout this project, ethical considerations remained at the forefront. We implemented safeguards to ensure responsible AI usage, addressing concerns related to bias, inappropriate responses, and privacy.
Future Directions:

The field of conversational AI is rapidly evolving. As new pre-trained language models and techniques emerge, there is potential for further enhancing our chatbot's capabilities.

Continuous monitoring and updates will be essential to maintain the chatbot's performance, adapt to changing user needs, and address any ethical concerns that may arise.

In conclusion, the creation of our chatbot represents a significant milestone in the application of advanced NLP techniques and pre-trained language models. It is a testament to the power of AI to enhance user experiences and provide intelligent, context-aware interactions. This project serves as a foundation for the ongoing development and innovation in the field of conversational AI.