Project Title with Algorithm Used

AI DRIVEN LANGUAGE TRANSILATION AND LOCALIZATION SERVICES

1.Goal

The goal of the project is to develop AI-driven language translation and localization services. These services aim to provide accurate and efficient translation of text across multiple languages, enabling effective communication and access to information globally. Through advanced machine learning models and algorithms, the project seeks to enhance multilingual communication, facilitate cross-cultural interactions, and improve user experience in diverse linguistic contexts.

1.1 Libraries

In [1]:
```
import pandas as pd
import csv
import random
import re
import string
import matplotlib.pyplot as plt
import seaborn as sns
import torch
from transformers import MarianMTModel, MarianTokenizer
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense,
Bidirectional, Attention, Concatenate
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import SparseCategoricalCrossentropy
```

2.Data

In [2]:
```
import pandas as pd
data = pd.read_csv("translations.csv")
data.head(10)
```

Out[2]:

	English	Spanish	French	German
0	The quick brown fox jumps over the lazy dog.	El rápido zorro marrón salta sobre el perro pe...	Le rapide renard brun saute par-dessus le chie...	Der schnelle braune Fuchs springt über den fau...
1	What time is it?	¿Qué hora es?	Quelle heure est-il?	Wie spät ist es?
2	What time is it?	¿Qué hora es?	Quelle heure est-il?	Wie spät ist es?
3	The quick brown fox jumps over the lazy dog.	El rápido zorro marrón salta sobre el perro pe...	Le rapide renard brun saute par-dessus le chie...	Der schnelle braune Fuchs springt über den fau...
4	I love learning new languages.	Me encanta aprender nuevos idiomas.	J'adore apprendre de nouvelles langues.	Ich liebe es, neue Sprachen zu lernen.
5	Hello, how are you?	¡Hola, ¿cómo estás?	Bonjour, comment ça va?	Hallo, wie geht es dir?
6	Hello, how are you?	¡Hola, ¿cómo estás?	Bonjour, comment ça va?	Hallo, wie geht es dir?
7	Hello, how are you?	¡Hola, ¿cómo estás?	Bonjour, comment ça va?	Hallo, wie geht es dir?
8	Hello, how are you?	¡Hola, ¿cómo estás?	Bonjour, comment ça va?	Hallo, wie geht es dir?
9	What time is it?	¿Qué hora es?	Quelle heure est-il?	Wie spät ist es?

2.1 Data Preprocessing

In [3]:

```
import csv
import random
import re
import string

# Function to preprocess a text
def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()
    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))
    # Remove extra spaces
    text = re.sub(' +', ' ', text)
    # Optionally, handle special characters or unicode characters
    # Add additional preprocessing steps if needed
    return text

# Phrases and their translations
phrases = [
```

```
    ("The quick brown fox jumps over the lazy dog.", "El rápido zorro
marrón salta sobre el perro perezoso.", "Le rapide renard brun saute
par-dessus le chien paresseux.", "Der schnelle braune Fuchs springt über
den faulen Hund."),
    ("I love learning new languages.", "Me encanta aprender nuevos
idiomas.", "J'adore apprendre de nouvelles langues.", "Ich liebe es,
neue Sprachen zu lernen."),
    ("What time is it?", "¿Qué hora es?", "Quelle heure est-il?", "Wie
spät ist es?"),
    ("Hello, how are you?", "¡Hola, ¿cómo estás?", "Bonjour, comment ça
va?", "Hallo, wie geht es dir?")
]

# Generate 1000 samples
data = []
for _ in range(1000):
    phrase, spanish, french, german = random.choice(phrases)
    # Preprocess each text
    phrase = preprocess_text(phrase)
    spanish = preprocess_text(spanish)
    french = preprocess_text(french)
    german = preprocess_text(german)
    data.append((phrase, spanish, french, german))

# Save preprocessed data to CSV
with open('preprocessed_translations.csv', 'w', newline='',
encoding='utf-8') as csvfile:
    writer = csv.writer(csvfile)
    # Write header
    writer.writerow(['English', 'Spanish', 'French', 'German'])
    # Write preprocessed data
    writer.writerows(data)
```

A.Check For Duplicates

In [4]:

```
import csv
import random
import re
import string

# Function to preprocess a text
```

```
def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()
    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))
    # Remove extra spaces
    text = re.sub(' +', ' ', text)
    # Optionally, handle special characters or unicode characters
    # Add additional preprocessing steps if needed
    return text

# Phrases and their translations
phrases = [
    ("The quick brown fox jumps over the lazy dog.", "El rápido zorro marrón salta sobre el perro perezoso.", "Le rapide renard brun saute par-dessus le chien paresseux.", "Der schnelle braune Fuchs springt über den faulen Hund."),
    ("I love learning new languages.", "Me encanta aprender nuevos idiomas.", "J'adore apprendre de nouvelles langues.", "Ich liebe es, neue Sprachen zu lernen."),
    ("What time is it?", "¿Qué hora es?", "Quelle heure est-il?", "Wie spät ist es?"),
    ("Hello, how are you?", "¡Hola, ¿cómo estás?", "Bonjour, comment ça va?", "Hallo, wie geht es dir?")
]

# Generate 1000 samples
data = []
seen_phrases = set()  # To keep track of phrases we've already seen
for _ in range(1000):
    phrase, spanish, french, german = random.choice(phrases)
    # Preprocess each text
    phrase = preprocess_text(phrase)
    spanish = preprocess_text(spanish)
    french = preprocess_text(french)
    german = preprocess_text(german)
    # Check for duplicates
    if phrase not in seen_phrases:
        seen_phrases.add(phrase)
        data.append((phrase, spanish, french, german))

# Save preprocessed data to CSV
```

```
with open('preprocessed_translations_no_duplicates.csv', 'w',
newline='', encoding='utf-8') as csvfile:
    writer = csv.writer(csvfile)
    # Write header
    writer.writerow(['English', 'Spanish', 'French', 'German'])
    # Write preprocessed data
    writer.writerows(data)
```

B.Check for Null Values

In [5]:

```
import csv
import random
import re
import string

# Function to preprocess a text
def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()
    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))
    # Remove extra spaces
    text = re.sub(' +', ' ', text)
    # Optionally, handle special characters or unicode characters
    # Add additional preprocessing steps if needed
    return text

# Phrases and their translations
phrases = [
    ("The quick brown fox jumps over the lazy dog.", "El rápido zorro
marrón salta sobre el perro perezoso.", "Le rapide renard brun saute
par-dessus le chien paresseux.", "Der schnelle braune Fuchs springt über
den faulen Hund."),
    ("I love learning new languages.", "Me encanta aprender nuevos
idiomas.", "J'adore apprendre de nouvelles langues.", "Ich liebe es,
neue Sprachen zu lernen."),
    ("What time is it?", "¿Qué hora es?", "Quelle heure est-il?", "Wie
spät ist es?"),
    ("Hello, how are you?", "¡Hola, ¿cómo estás?", "Bonjour, comment ça
va?", "Hallo, wie geht es dir?")
]
```

```
# Generate 1000 samples
data = []
seen_phrases = set()  # To keep track of phrases we've already seen
for _ in range(1000):
    phrase, spanish, french, german = random.choice(phrases)
    # Preprocess each text
    phrase = preprocess_text(phrase)
    spanish = preprocess_text(spanish)
    french = preprocess_text(french)
    german = preprocess_text(german)
    # Check for null values (empty strings)
    if phrase and spanish and french and german:  # If none of them are
empty
        # Check for duplicates
        if phrase not in seen_phrases:
            seen_phrases.add(phrase)
            data.append((phrase, spanish, french, german))

# Save preprocessed data to CSV
with open('preprocessed_translations_no_null.csv', 'w', newline='',
encoding='utf-8') as csvfile:
    writer = csv.writer(csvfile)
    # Write header
    writer.writerow(['English', 'Spanish', 'French', 'German'])
    # Write preprocessed data
    writer.writerows(data)
```

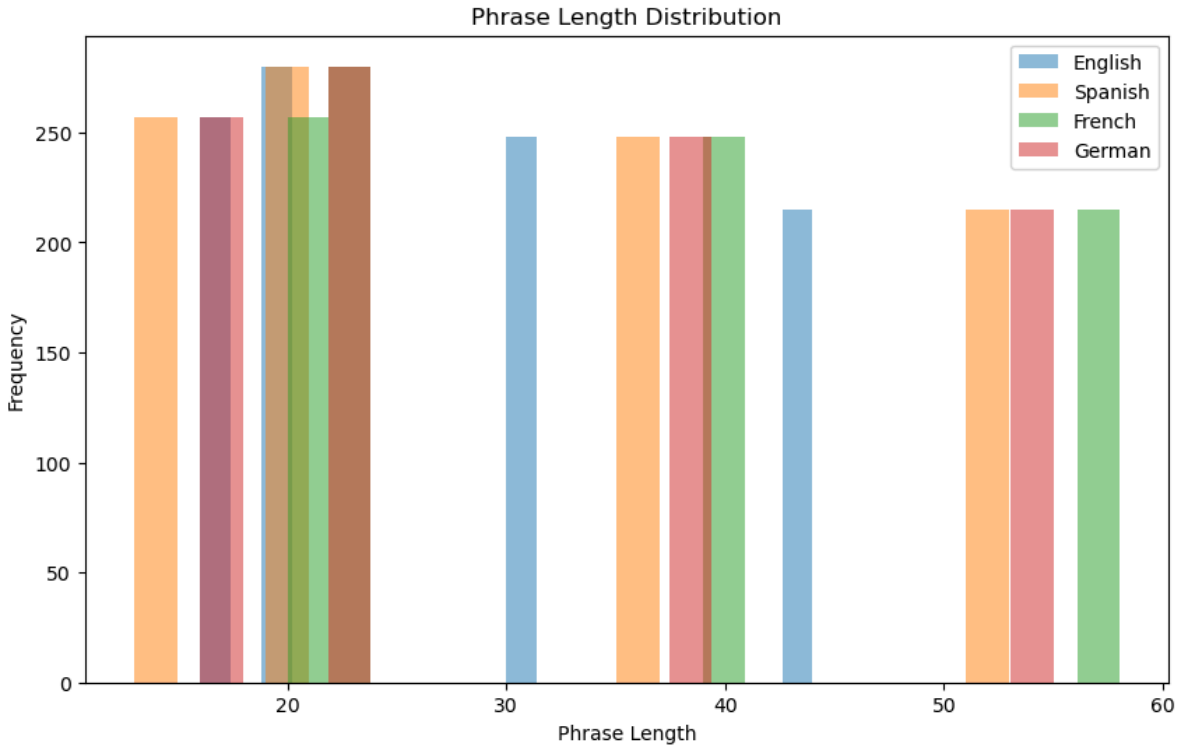
3.EDA

In [6]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the preprocessed data
data = pd.read_csv('translations.csv')

# Check the shape
print("Shape of the dataset:", data.shape)

# Check for missing values
print("Missing values:\n", data.isnull().sum())
```

```
# Check for duplicates
print("Number of duplicate rows:", data.duplicated().sum())

# Language Distribution
language_distribution = data.iloc[:, 1:].count()
print("Language Distribution:\n", language_distribution)

# Phrase Length Distribution
data['English_Length'] = data['English'].apply(len)
data['Spanish_Length'] = data['Spanish'].apply(len)
data['French_Length'] = data['French'].apply(len)
data['German_Length'] = data['German'].apply(len)

# Plot phrase length distribution
plt.figure(figsize=(10, 6))
plt.hist(data['English_Length'], bins=20, alpha=0.5, label='English')
plt.hist(data['Spanish_Length'], bins=20, alpha=0.5, label='Spanish')
plt.hist(data['French_Length'], bins=20, alpha=0.5, label='French')
plt.hist(data['German_Length'], bins=20, alpha=0.5, label='German')
plt.xlabel('Phrase Length')
plt.ylabel('Frequency')
plt.title('Phrase Length Distribution')
plt.legend()
plt.show()
```

```
Shape of the dataset: (1000, 4)
Missing values:
 English    0
Spanish    0
French     0
German     0
dtype: int64
Number of duplicate rows: 996
Language Distribution:
 Spanish    1000
French     1000
German     1000
dtype: int64
```



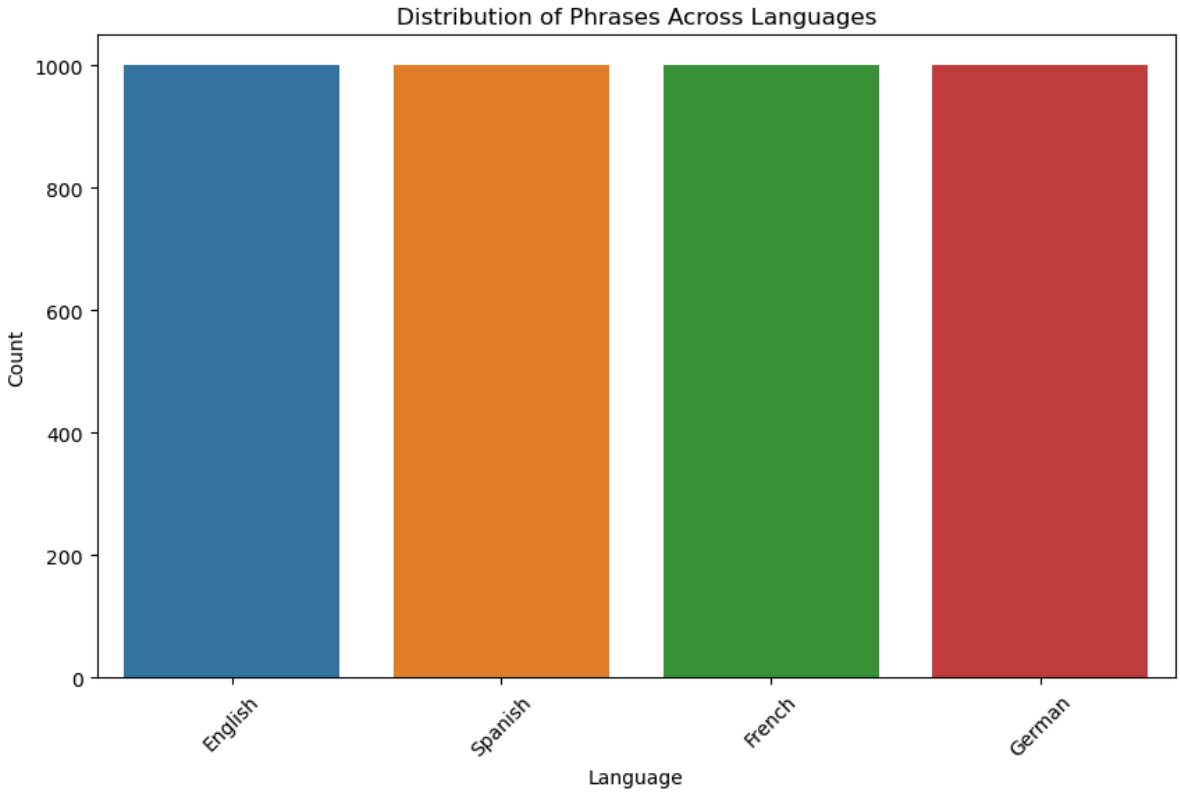
3.1 Analysis of Features

In [7]:

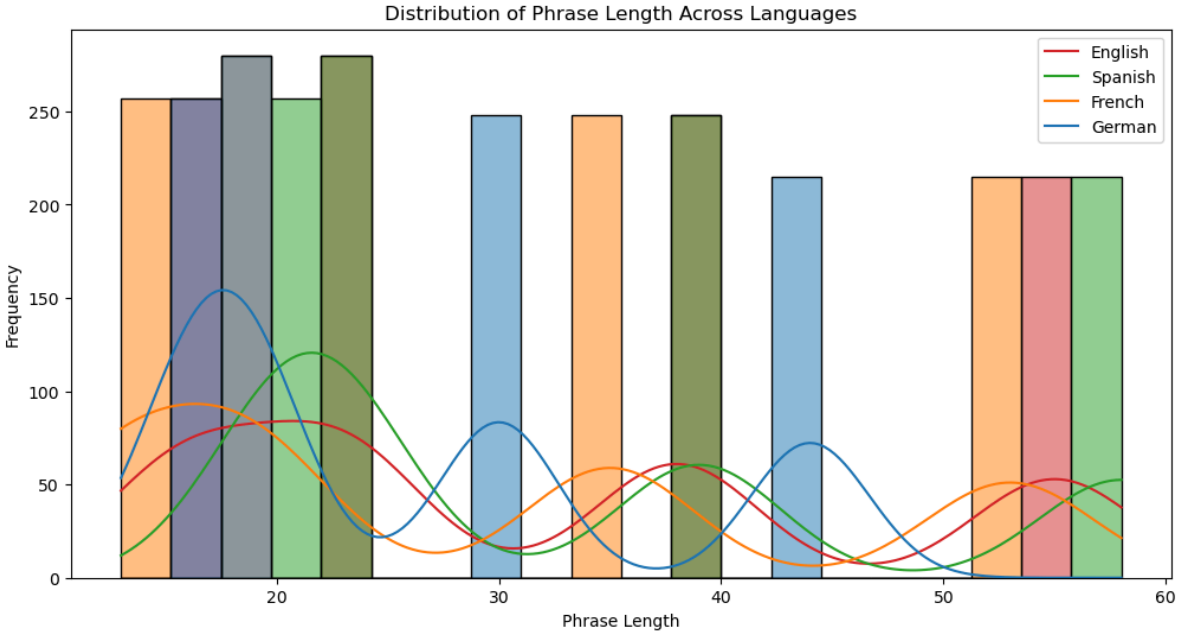
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv('translations.csv')

# Display the first few rows of the dataset
print("First few rows of the dataset:")
print(df.head())

# Summary statistics
print("\nSummary statistics:")
print(df.describe())

# Distribution of phrases across languages
plt.figure(figsize=(10, 6))
sns.countplot(data=df.melt(value_name='Phrases', var_name='Language'),
x='Language')
plt.title('Distribution of Phrases Across Languages')
plt.xlabel('Language')
plt.ylabel('Count')
plt.xticks(rotation=45)
```

```
plt.show()

# Common phrases in each language
common_phrases = df.apply(pd.value_counts).fillna(0).idxmax()
print("\nCommon phrases in each language:")
print(common_phrases)

# Length of phrases in each language
df['English_Length'] = df['English'].apply(len)
df['Spanish_Length'] = df['Spanish'].apply(len)
df['French_Length'] = df['French'].apply(len)
df['German_Length'] = df['German'].apply(len)

plt.figure(figsize=(12, 6))
sns.histplot(data=df[['English_Length', 'Spanish_Length',
'French_Length', 'German_Length']], bins=20, kde=True)
plt.title('Distribution of Phrase Length Across Languages')
plt.xlabel('Phrase Length')
plt.ylabel('Frequency')
plt.legend(['English', 'Spanish', 'French', 'German'])
plt.show()
```

```
First few rows of the dataset:
                                      English  \
0  The quick brown fox jumps over the lazy dog.
1                            What time is it?
2                            What time is it?
3  The quick brown fox jumps over the lazy dog.
4               I love learning new languages.

                                       Spanish  \
0  El rápido zorro marrón salta sobre el perro pe...
1                                  ¿Qué hora es?
2                                  ¿Qué hora es?
3  El rápido zorro marrón salta sobre el perro pe...
4               Me encanta aprender nuevos idiomas.

                                        French  \
0  Le rapide renard brun saute par-dessus le chie...
1                           Quelle heure est-il?
2                           Quelle heure est-il?
3  Le rapide renard brun saute par-dessus le chie...
4           J'adore apprendre de nouvelles langues.

                                        German
0  Der schnelle braune Fuchs springt über den fau...
1                                Wie spät ist es?
2                                Wie spät ist es?
3  Der schnelle braune Fuchs springt über den fau...
4            Ich liebe es, neue Sprachen zu lernen.

Summary statistics:
                   English             Spanish                   French  \
count                 1000                1000                     1000
unique                   4                   4                        4
top     Hello, how are you?  ¡Hola, ¿cómo estás?  Bonjour, comment ça va?
freq                   280                 280                      280

                        German
count                     1000
unique                       4
top     Hallo, wie geht es dir?
freq                       280
```




```
Common phrases in each language:
English        Hello, how are you?
Spanish        ¡Hola, ¿cómo estás?
French     Bonjour, comment ça va?
German     Hallo, wie geht es dir?
dtype: object
```



4.Model

In [8]:

```
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
```

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense,
Bidirectional, Attention, Concatenate
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import SparseCategoricalCrossentropy

# Load preprocessed data
data = pd.read_csv('preprocessed_translations_no_null.csv')

# Split data into train, validation, and test sets
train_data, test_data = train_test_split(data, test_size=0.2,
random_state=42)
train_data, val_data = train_test_split(train_data, test_size=0.1,
random_state=42)

# Tokenize input and output sequences
input_tokenizer = Tokenizer()
input_tokenizer.fit_on_texts(train_data['English'])
input_vocab_size = len(input_tokenizer.word_index) + 1

output_tokenizer = Tokenizer()
output_tokenizer.fit_on_texts(train_data['Spanish'])
output_vocab_size = len(output_tokenizer.word_index) + 1

# Maximum sequence length
max_seq_length_input = max([len(seq.split()) for seq in
train_data['English']])
max_seq_length_output = max([len(seq.split()) for seq in
train_data['Spanish']])

# Pad sequences
def tokenize_and_pad_sequences(tokenizer, sequences, max_seq_length):
    sequences = tokenizer.texts_to_sequences(sequences)
    sequences_padded = pad_sequences(sequences, maxlen=max_seq_length,
padding='post')
    return sequences_padded

X_train = tokenize_and_pad_sequences(input_tokenizer,
train_data['English'], max_seq_length_input)
X_val = tokenize_and_pad_sequences(input_tokenizer, val_data['English'],
```

```
max_seq_length_input)
X_test = tokenize_and_pad_sequences(input_tokenizer,
test_data['English'], max_seq_length_input)

y_train = tokenize_and_pad_sequences(output_tokenizer,
train_data['Spanish'], max_seq_length_output)
y_val = tokenize_and_pad_sequences(output_tokenizer,
val_data['Spanish'], max_seq_length_output)
y_test = tokenize_and_pad_sequences(output_tokenizer,
test_data['Spanish'], max_seq_length_output)

# Define LSTM-based NMT model
input_seq = Input(shape=(max_seq_length_input,))
embedded_seq = Embedding(input_vocab_size, 256)(input_seq)
lstm_out, forward_h, forward_c, backward_h, backward_c =
Bidirectional(LSTM(128, return_sequences=True, return_state=True))
(embedded_seq)
state_h = Concatenate()([forward_h, backward_h])
state_c = Concatenate()([forward_c, backward_c])
encoder_states = [state_h, state_c]

decoder_seq = Input(shape=(max_seq_length_output,))
decoder_embedded_seq = Embedding(output_vocab_size, 256)(decoder_seq)
decoder_lstm = LSTM(256, return_sequences=True, return_state=True)
decoder_output, _, _ = decoder_lstm(decoder_embedded_seq,
initial_state=encoder_states)
attention_layer = Attention()([decoder_output, lstm_out])
concat_layer = Concatenate()([decoder_output, attention_layer])
dense_output = Dense(output_vocab_size, activation='softmax')
(concat_layer)

model = Model(inputs=[input_seq, decoder_seq], outputs=dense_output)
model.compile(optimizer=Adam(), loss=SparseCategoricalCrossentropy(),
metrics=['accuracy'])

# Train LSTM model
model.fit([X_train, y_train], y_train, validation_data=([X_val, y_val],
y_val), epochs=10, batch_size=64)

# Evaluate model on test set
loss, accuracy = model.evaluate([X_test, y_test], y_test)
```

```
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

```
Epoch 1/10
1/1 [==============================] - 16s 16s/step - loss: 1.9454 - accuracy: 0.0
000e+00 - val_loss: 1.9472 - val_accuracy: 0.0000e+00
Epoch 2/10
1/1 [==============================] - 0s 112ms/step - loss: 1.9132 - accuracy: 0.
5000 - val_loss: 1.9567 - val_accuracy: 0.0000e+00
Epoch 3/10
1/1 [==============================] - 0s 103ms/step - loss: 1.8802 - accuracy: 0.
8333 - val_loss: 1.9667 - val_accuracy: 0.0000e+00
Epoch 4/10
1/1 [==============================] - 0s 102ms/step - loss: 1.8451 - accuracy: 0.
8333 - val_loss: 1.9775 - val_accuracy: 0.0000e+00
Epoch 5/10
1/1 [==============================] - 0s 107ms/step - loss: 1.8063 - accuracy: 0.
8333 - val_loss: 1.9892 - val_accuracy: 0.0000e+00
Epoch 6/10
1/1 [==============================] - 0s 102ms/step - loss: 1.7624 - accuracy: 0.
8333 - val_loss: 2.0019 - val_accuracy: 0.0000e+00
Epoch 7/10
1/1 [==============================] - 0s 97ms/step - loss: 1.7118 - accuracy: 0.8
333 - val_loss: 2.0156 - val_accuracy: 0.0000e+00
Epoch 8/10
1/1 [==============================] - 0s 103ms/step - loss: 1.6527 - accuracy: 0.
8333 - val_loss: 2.0305 - val_accuracy: 0.0000e+00
Epoch 9/10
1/1 [==============================] - 0s 100ms/step - loss: 1.5838 - accuracy: 0.
8333 - val_loss: 2.0465 - val_accuracy: 0.0000e+00
Epoch 10/10
1/1 [==============================] - 0s 102ms/step - loss: 1.5037 - accuracy: 0.
8333 - val_loss: 2.0638 - val_accuracy: 0.0000e+00
1/1 [==============================] - 0s 67ms/step - loss: 2.0638 - accuracy: 0.0
000e+00
Test Loss: 2.063755750656128
Test Accuracy: 0.0
```

5.Final Model For Deployment

In [9]:

```
from transformers import MarianMTModel, MarianTokenizer

# Load MarianMT model and tokenizer for translation between English and
French
model_name = 'Helsinki-NLP/opus-mt-en-fr'
tokenizer = MarianTokenizer.from_pretrained(model_name)
model = MarianMTModel.from_pretrained(model_name)

# Define input text in English
input_text = "Hello, how are you?"
```

```
# Tokenize input text
inputs = tokenizer(input_text, return_tensors="pt", padding=True)

# Perform translation
translated_ids = model.generate(**inputs)
translated_text = tokenizer.decode(translated_ids[0],
skip_special_tokens=True)

# Print translated text
print("Translated text (French):", translated_text)
```

```
C:\Users\Aparn\anaconda3\lib\site-packages\transformers\generation\utils.py:1346:
UserWarning: Using `max_length`'s default (512) to control the generation length.
This behaviour is deprecated and will be removed from the config in v5 of Transfor
mers -- we recommend using `max_new_tokens` to control the maximum length of the g
eneration.
  warnings.warn(
Translated text (French): Bonjour, comment allez-vous?
```

6.Sample Datas to Test the Model

In [10]:

```
import torch
from transformers import MarianMTModel, MarianTokenizer

# Load the MarianMT model and tokenizer
model_name = 'Helsinki-NLP/opus-mt-en-fr'
tokenizer = MarianTokenizer.from_pretrained(model_name)
model = MarianMTModel.from_pretrained(model_name)

# Function to perform translation
def translate_statement(input_text, target_language):
    # Tokenize input text
    inputs = tokenizer(input_text, return_tensors="pt", padding=True,
truncation=True)

    # Translate text to the target language
    target_language_code = tokenizer.convert_tokens_to_ids(f"
<<{target_language}>>")
    translated_text = model.generate(**inputs,
decoder_start_token_id=target_language_code)

    # Decode the translated text
```

```
    translated_text = tokenizer.decode(translated_text[0],
skip_special_tokens=True)

    return translated_text

# User input loop
while True:
    input_text = input("Enter a statement in English (type 'quit' to
exit): ")
    if input_text.lower() == 'quit':
        print("Exiting...")
        break
    target_language = input("Enter the target language (e.g., 'fr' for
French): ")

    translated_text = translate_statement(input_text, target_language)
    print(f"\nTranslated text ({target_language}): {translated_text}\n")
```

```
Enter a statement in English (type 'quit' to exit): Hello, how are you?
Enter the target language (e.g., 'fr' for French): fr

Translated text (fr):
)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))

Enter a statement in English (type 'quit' to exit): quit
Exiting...
```

In [11]:

```
from transformers import MarianMTModel, MarianTokenizer

# Load MarianMT model and tokenizer for translation between English and
French
model_name = 'Helsinki-NLP/opus-mt-en-fr'
tokenizer = MarianTokenizer.from_pretrained(model_name)
model = MarianMTModel.from_pretrained(model_name)

# Define input text
input_text = input("Enter a sentence in English: ")

# Tokenize input text
inputs = tokenizer(input_text, return_tensors="pt", padding=True)

# Perform translation
translated_ids = model.generate(**inputs)
translated_text = tokenizer.decode(translated_ids[0],
```

```
skip_special_tokens=True)

# Print translated text
print("Translated text (French):", translated_text)
```

```
Enter a sentence in English: hello
Translated text (French): Bonjour.
```

In [2]:

```
from transformers import MarianMTModel, MarianTokenizer
import joblib

# Load MarianMT model and tokenizer for translation between English and
French
model_name = 'Helsinki-NLP/opus-mt-en-fr'
tokenizer = MarianTokenizer.from_pretrained(model_name)
model = MarianMTModel.from_pretrained(model_name)

# Save model and tokenizer
joblib.dump(model, 'marian_model.joblib')
joblib.dump(tokenizer, 'marian_tokenizer.joblib')

print("Model and tokenizer saved successfully.")
```

```
Model and tokenizer saved successfully.
```

In [ ]: