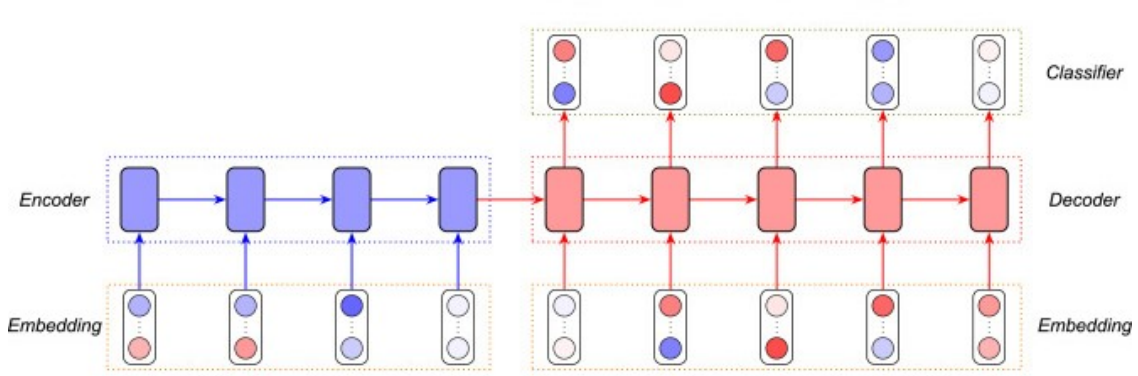
ENGLISHH_GERMANN

June 30, 2023

1 Machine Translation: ENGLISH TO GERMAN

2 (Encoder-Decoder)



3 Import and preprocessing the data

[2]:
```
import pandas as pd
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Input, LSTM, Dense, Embedding
from tensorflow.keras.models import Model
import numpy as np

# Load and preprocess the data
data = pd.read_csv('/kaggle/input/hey-buddy/GERMAN_ENGLISH_TRANSLATION.csv')
data = data.drop_duplicates(subset=['ENGLISH'])
data = data.head(20000)

english_sentences = data['ENGLISH'].str.lower().str.replace('[^\w\s]', '').
 ↪tolist()
german_sentences = data['GERMAN'].str.lower().str.replace('[^\w\s]', '').
 ↪apply(lambda x: '<start> ' + x + ' <end>').tolist()
```

```
/tmp/ipykernel_28/3543679006.py:13: FutureWarning: The default value of regex
will change from True to False in a future version.
```

```
  english_sentences = data['ENGLISH'].str.lower().str.replace('[^\w\s]',
'').tolist()
/tmp/ipykernel_28/3543679006.py:14: FutureWarning: The default value of regex
will change from True to False in a future version.
  german_sentences = data['GERMAN'].str.lower().str.replace('[^\w\s]',
'').apply(lambda x: '<start> ' + x + ' <end>').tolist()
```

3.0.1 sample of data

[3]:
```
print(data.head())
```

```
   Unnamed: 0 ENGLISH      GERMAN
0           0      hi       hallo
2           2     run        lauf
3           3     wow  potzdonner
5           5    fire       feuer
6           6    help       hilfe
```

[4]:
```
print(german_sentences[0:5])
```

```
['<start> hallo <end>', '<start> lauf <end>', '<start> potzdonner <end>',
'<start> feuer <end>', '<start> hilfe <end>']
```

4 Generate Tokenization

[5]:
```
# Integer encode sentences
eng_token = Tokenizer(filters='')
eng_token.fit_on_texts(english_sentences)
eng_token_ind = eng_token.texts_to_sequences(english_sentences)

ger_token = Tokenizer(filters='')
ger_token.fit_on_texts(german_sentences)
ger_token_ind= ger_token.texts_to_sequences(german_sentences)

# Pad encoded sentences
max_encoder_seq_length = max([len(seq) for seq in eng_token_ind])
max_decoder_seq_length = max([len(seq) for seq in ger_token_ind])
print(max_encoder_seq_length,max_decoder_seq_length)
```

```
6 13
```

4.0.1 Create Encoder (input data) and decoder (input,target) data

[6]:
```
encoder_input_data = pad_sequences(eng_token_ind,␣
 ↪maxlen=max_encoder_seq_length, padding='post')
decoder_input_data = pad_sequences(ger_token_ind,␣
 ↪maxlen=max_decoder_seq_length, padding='post')
```

```
# target data for the decoder
decoder_target_data = []
for seq in ger_token_ind:
    decoder_target_data.append(seq[1:])
decoder_target_data = pad_sequences(decoder_target_data,␣
 ↪maxlen=max_decoder_seq_length, padding='post')

num_decoder_tokens = len(ger_token.word_index) + 1
decoder_output = np.zeros((len(ger_token_ind), max_decoder_seq_length,␣
 ↪num_decoder_tokens), dtype='float32')
for i, seq in enumerate(decoder_target_data):
    for t, token in enumerate(seq):
        decoder_output[i, t, token] = 1
```

5 Architecture- LSTM

[7]:
```
# Create the model
latent_dim = 256
num_encoder_tokens = len(eng_token.word_index) + 1

eng_embedding_layer = Embedding(num_encoder_tokens, latent_dim)
ger_embedding_layer = Embedding(num_decoder_tokens, latent_dim)

encoder_inputs = Input(shape=(None,))
encoder_embedding = eng_embedding_layer(encoder_inputs)
encoder_lstm = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder_lstm(encoder_embedding)
encoder_states = [state_h, state_c]

decoder_inputs = Input(shape=(None,))
decoder_embedding = ger_embedding_layer(decoder_inputs)
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_embedding,␣
 ↪initial_state=encoder_states)
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

6 Train the model

[8]:
```
# Compile and train the model
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
model.fit([encoder_input_data, decoder_input_data], decoder_output,
          batch_size=64,
```

```
          epochs=60,
          validation_split=0.2)
```

```
Epoch 1/60
250/250 [==============================] - 36s 112ms/step - loss: 2.4325 -
accuracy: 0.6773 - val_loss: 2.2305 - val_accuracy: 0.6750
Epoch 2/60
250/250 [==============================] - 13s 50ms/step - loss: 1.7670 -
accuracy: 0.7303 - val_loss: 2.1167 - val_accuracy: 0.6826
Epoch 3/60
250/250 [==============================] - 13s 53ms/step - loss: 1.6095 -
accuracy: 0.7475 - val_loss: 1.9734 - val_accuracy: 0.7143
Epoch 4/60
250/250 [==============================] - 13s 52ms/step - loss: 1.4438 -
accuracy: 0.7709 - val_loss: 1.8405 - val_accuracy: 0.7317
Epoch 5/60
250/250 [==============================] - 13s 53ms/step - loss: 1.2906 -
accuracy: 0.7913 - val_loss: 1.7410 - val_accuracy: 0.7482
Epoch 6/60
250/250 [==============================] - 13s 53ms/step - loss: 1.1592 -
accuracy: 0.8067 - val_loss: 1.6627 - val_accuracy: 0.7595
Epoch 7/60
250/250 [==============================] - 12s 48ms/step - loss: 1.0489 -
accuracy: 0.8194 - val_loss: 1.5983 - val_accuracy: 0.7660
Epoch 8/60
250/250 [==============================] - 12s 47ms/step - loss: 0.9498 -
accuracy: 0.8308 - val_loss: 1.5485 - val_accuracy: 0.7729
Epoch 9/60
250/250 [==============================] - 12s 50ms/step - loss: 0.8572 -
accuracy: 0.8416 - val_loss: 1.5138 - val_accuracy: 0.7784
Epoch 10/60
250/250 [==============================] - 12s 47ms/step - loss: 0.7717 -
accuracy: 0.8518 - val_loss: 1.4852 - val_accuracy: 0.7831
Epoch 11/60
250/250 [==============================] - 12s 49ms/step - loss: 0.6926 -
accuracy: 0.8622 - val_loss: 1.4530 - val_accuracy: 0.7874
Epoch 12/60
250/250 [==============================] - 12s 49ms/step - loss: 0.6185 -
accuracy: 0.8734 - val_loss: 1.4297 - val_accuracy: 0.7921
Epoch 13/60
250/250 [==============================] - 12s 47ms/step - loss: 0.5494 -
accuracy: 0.8846 - val_loss: 1.4196 - val_accuracy: 0.7946
Epoch 14/60
250/250 [==============================] - 13s 50ms/step - loss: 0.4862 -
accuracy: 0.8961 - val_loss: 1.4135 - val_accuracy: 0.7980
Epoch 15/60
250/250 [==============================] - 12s 48ms/step - loss: 0.4278 -
accuracy: 0.9083 - val_loss: 1.4041 - val_accuracy: 0.7996
```

```
Epoch 16/60
250/250 [==============================] - 12s 49ms/step - loss: 0.3744 -
accuracy: 0.9198 - val_loss: 1.3971 - val_accuracy: 0.8014
Epoch 17/60
250/250 [==============================] - 12s 49ms/step - loss: 0.3267 -
accuracy: 0.9311 - val_loss: 1.4019 - val_accuracy: 0.8036
Epoch 18/60
250/250 [==============================] - 12s 48ms/step - loss: 0.2838 -
accuracy: 0.9408 - val_loss: 1.4059 - val_accuracy: 0.8062
Epoch 19/60
250/250 [==============================] - 13s 51ms/step - loss: 0.2459 -
accuracy: 0.9501 - val_loss: 1.4106 - val_accuracy: 0.8077
Epoch 20/60
250/250 [==============================] - 12s 48ms/step - loss: 0.2123 -
accuracy: 0.9581 - val_loss: 1.4192 - val_accuracy: 0.8096
Epoch 21/60
250/250 [==============================] - 12s 48ms/step - loss: 0.1822 -
accuracy: 0.9656 - val_loss: 1.4227 - val_accuracy: 0.8096
Epoch 22/60
250/250 [==============================] - 13s 50ms/step - loss: 0.1566 -
accuracy: 0.9713 - val_loss: 1.4388 - val_accuracy: 0.8101
Epoch 23/60
250/250 [==============================] - 12s 47ms/step - loss: 0.1339 -
accuracy: 0.9767 - val_loss: 1.4452 - val_accuracy: 0.8104
Epoch 24/60
250/250 [==============================] - 12s 50ms/step - loss: 0.1147 -
accuracy: 0.9805 - val_loss: 1.4676 - val_accuracy: 0.8102
Epoch 25/60
250/250 [==============================] - 12s 49ms/step - loss: 0.0986 -
accuracy: 0.9840 - val_loss: 1.4709 - val_accuracy: 0.8115
Epoch 26/60
250/250 [==============================] - 12s 48ms/step - loss: 0.0837 -
accuracy: 0.9873 - val_loss: 1.4846 - val_accuracy: 0.8120
Epoch 27/60
250/250 [==============================] - 13s 51ms/step - loss: 0.0716 -
accuracy: 0.9897 - val_loss: 1.5073 - val_accuracy: 0.8125
Epoch 28/60
250/250 [==============================] - 12s 47ms/step - loss: 0.0608 -
accuracy: 0.9918 - val_loss: 1.5230 - val_accuracy: 0.8118
Epoch 29/60
250/250 [==============================] - 12s 47ms/step - loss: 0.0515 -
accuracy: 0.9935 - val_loss: 1.5281 - val_accuracy: 0.8137
Epoch 30/60
250/250 [==============================] - 13s 50ms/step - loss: 0.0437 -
accuracy: 0.9950 - val_loss: 1.5483 - val_accuracy: 0.8127
Epoch 31/60
250/250 [==============================] - 12s 48ms/step - loss: 0.0378 -
accuracy: 0.9958 - val_loss: 1.5557 - val_accuracy: 0.8125
```

```
Epoch 32/60
250/250 [==============================] - 12s 49ms/step - loss: 0.0334 -
accuracy: 0.9965 - val_loss: 1.5761 - val_accuracy: 0.8130
Epoch 33/60
250/250 [==============================] - 12s 49ms/step - loss: 0.0276 -
accuracy: 0.9976 - val_loss: 1.5800 - val_accuracy: 0.8138
Epoch 34/60
250/250 [==============================] - 12s 49ms/step - loss: 0.0237 -
accuracy: 0.9981 - val_loss: 1.6075 - val_accuracy: 0.8137
Epoch 35/60
250/250 [==============================] - 12s 48ms/step - loss: 0.0205 -
accuracy: 0.9985 - val_loss: 1.6161 - val_accuracy: 0.8127
Epoch 36/60
250/250 [==============================] - 12s 49ms/step - loss: 0.0170 -
accuracy: 0.9990 - val_loss: 1.6249 - val_accuracy: 0.8139
Epoch 37/60
250/250 [==============================] - 12s 49ms/step - loss: 0.0140 -
accuracy: 0.9993 - val_loss: 1.6451 - val_accuracy: 0.8142
Epoch 38/60
250/250 [==============================] - 12s 48ms/step - loss: 0.0127 -
accuracy: 0.9994 - val_loss: 1.6596 - val_accuracy: 0.8134
Epoch 39/60
250/250 [==============================] - 12s 47ms/step - loss: 0.0115 -
accuracy: 0.9995 - val_loss: 1.6724 - val_accuracy: 0.8130
Epoch 40/60
250/250 [==============================] - 12s 48ms/step - loss: 0.0105 -
accuracy: 0.9995 - val_loss: 1.6794 - val_accuracy: 0.8136
Epoch 41/60
250/250 [==============================] - 12s 49ms/step - loss: 0.0144 -
accuracy: 0.9986 - val_loss: 1.7091 - val_accuracy: 0.8108
Epoch 42/60
250/250 [==============================] - 12s 47ms/step - loss: 0.0266 -
accuracy: 0.9955 - val_loss: 1.7163 - val_accuracy: 0.8108
Epoch 43/60
250/250 [==============================] - 12s 50ms/step - loss: 0.0244 -
accuracy: 0.9959 - val_loss: 1.6964 - val_accuracy: 0.8120
Epoch 44/60
250/250 [==============================] - 12s 49ms/step - loss: 0.0123 -
accuracy: 0.9988 - val_loss: 1.7175 - val_accuracy: 0.8154
Epoch 45/60
250/250 [==============================] - 12s 49ms/step - loss: 0.0062 -
accuracy: 0.9998 - val_loss: 1.7298 - val_accuracy: 0.8166
Epoch 46/60
250/250 [==============================] - 12s 47ms/step - loss: 0.0039 -
accuracy: 0.9999 - val_loss: 1.7422 - val_accuracy: 0.8171
Epoch 47/60
250/250 [==============================] - 12s 49ms/step - loss: 0.0030 -
accuracy: 1.0000 - val_loss: 1.7519 - val_accuracy: 0.8172
```

```
Epoch 48/60
250/250 [==============================] - 12s 48ms/step - loss: 0.0025 -
accuracy: 1.0000 - val_loss: 1.7625 - val_accuracy: 0.8167
Epoch 49/60
250/250 [==============================] - 12s 49ms/step - loss: 0.0022 -
accuracy: 1.0000 - val_loss: 1.7718 - val_accuracy: 0.8170
Epoch 50/60
250/250 [==============================] - 12s 49ms/step - loss: 0.0020 -
accuracy: 1.0000 - val_loss: 1.7816 - val_accuracy: 0.8172
Epoch 51/60
250/250 [==============================] - 12s 49ms/step - loss: 0.0019 -
accuracy: 1.0000 - val_loss: 1.7907 - val_accuracy: 0.8166
Epoch 52/60
250/250 [==============================] - 12s 49ms/step - loss: 0.0016 -
accuracy: 1.0000 - val_loss: 1.8021 - val_accuracy: 0.8170
Epoch 53/60
250/250 [==============================] - 12s 50ms/step - loss: 0.0016 -
accuracy: 1.0000 - val_loss: 1.8104 - val_accuracy: 0.8167
Epoch 54/60
250/250 [==============================] - 12s 47ms/step - loss: 0.0014 -
accuracy: 1.0000 - val_loss: 1.8188 - val_accuracy: 0.8161
Epoch 55/60
250/250 [==============================] - 12s 47ms/step - loss: 0.0012 -
accuracy: 1.0000 - val_loss: 1.8314 - val_accuracy: 0.8166
Epoch 56/60
250/250 [==============================] - 12s 50ms/step - loss: 0.0010 -
accuracy: 1.0000 - val_loss: 1.8432 - val_accuracy: 0.8161
Epoch 57/60
250/250 [==============================] - 12s 49ms/step - loss: 0.0011 -
accuracy: 1.0000 - val_loss: 1.8484 - val_accuracy: 0.8132
Epoch 58/60
250/250 [==============================] - 12s 49ms/step - loss: 0.0788 -
accuracy: 0.9777 - val_loss: 1.7945 - val_accuracy: 0.8074
Epoch 59/60
250/250 [==============================] - 12s 48ms/step - loss: 0.0278 -
accuracy: 0.9931 - val_loss: 1.7867 - val_accuracy: 0.8123
Epoch 60/60
250/250 [==============================] - 12s 49ms/step - loss: 0.0075 -
accuracy: 0.9992 - val_loss: 1.7950 - val_accuracy: 0.8167
```

[8]: <keras.callbacks.History at 0x7da4d0490940>

7 Create the encoder and decoder models for inference

[9]:
```
encoder_model = Model(encoder_inputs, encoder_states)
encoder_model.save('encoder_model.h5')
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]
decoder_embedding_inference = ger_embedding_layer(decoder_inputs)
decoder_outputs_inference, state_h_inference, state_c_inference =␣
 ↪decoder_lstm(decoder_embedding_inference,

                                                                  ␣
 ↪initial_state=decoder_states_inputs)
decoder_states_inference = [state_h_inference, state_c_inference]
decoder_outputs_inference = decoder_dense(decoder_outputs_inference)
decoder_model = Model([decoder_inputs] + decoder_states_inputs,
                      [decoder_outputs_inference] + decoder_states_inference)
decoder_model.save('decoder_model.h5')
```

[10]:
```
import json

# Save eng_tokenizer
eng_token_dict = {"word_index": eng_token.word_index}
with open('eng_tokenizer.json', 'w') as f:
    json.dump(eng_token_dict, f)

# Save ger_tokenizer
ger_token_dict = {"word_index": ger_token.word_index}
with open('ger_tokenizer.json', 'w') as f:
    json.dump(ger_token_dict, f)
```

[11]:
```
import json

# Save ger_token's index_word dictionary
ger_token_dict = {"index_word": ger_token.index_word}
with open('ger_token_index_word.json', 'w') as f:
    json.dump(ger_token_dict, f)
```

7.1 Decode the sentence

[12]:
```
# Function to decode a new sentence
def decode_sequence(input_seq):
    states_value = encoder_model.predict(input_seq)
    target_seq = np.zeros((1, 1))
    target_seq[0, 0] = ger_token.word_index['<start>']
    stop_condition=False
    decoded_sentence=''
    while not stop_condition:
```

```
        output_tokens,h,c=decoder_model.
↪predict([target_seq]+states_value,verbose=0)
        sampled_token_index=np.argmax(output_tokens[0,-1,:])
        sampled_word=ger_token.index_word[sampled_token_index]
        if sampled_word != '<end>':
            decoded_sentence += ' '+sampled_word

        if (sampled_word == '<end>' or len(decoded_sentence.split()) >␣
↪max_decoder_seq_length):
            stop_condition=True

        target_seq=np.zeros((1,1))
        target_seq[0,0]=sampled_token_index

        states_value=[h,c]
    return decoded_sentence
```

8 Testing the model

[13]:
```
listt=[
    'hello',
    'I won',
    'Go Away',
    'I gave up',
    'I am a man',
    'Tom really got a bad deal',
    'he will go with us',
    'what are you reading',
    'hi',
    'The reason does not matter',
    'I love your cat',
    'go',
    'get inside'
]
actual=[
    'hallo',
    'ich habe gewonnen',
    'geh weg',
    'ich gab auf',
    'ich bin ein Mann',
    'tom hat wirklich ein schlechtes Geschäft gemacht',
    'er wird mit uns gehen',
    'was liest du?',
    'hallo',
    'der Grund ist egal',
    'ich liebe deine Katze',
```

```
    'geh',
    'Komm herein'
]
for i in range(len(listt)):
    new_english_sentence=listt[i]
    new_english_sentence.lower().replace('[^\w\s]', '')
    new_eng_integer_encoded=eng_token.texts_to_sequences([new_english_sentence])
   ␣
 ↪new_encoder_input_data=pad_sequences(new_eng_integer_encoded,maxlen=max_encoder_seq_length,
    decoded_sentence=decode_sequence(new_encoder_input_data)
    decoded_sentence=decoded_sentence.strip()
    print('Input sentence:', new_english_sentence)
    print('Actual sentence:', actual[i])
    print('Decoded sentence:', decoded_sentence)
```

```
1/1 [==============================] - 0s 380ms/step
Input sentence: hello
Actual sentence: hallo
Decoded sentence: hallo
1/1 [==============================] - 0s 17ms/step
Input sentence: I won
Actual sentence: ich habe gewonnen
Decoded sentence: ich hab gewonnen
1/1 [==============================] - 0s 17ms/step
Input sentence: Go Away
Actual sentence: geh weg
Decoded sentence: geh weg
1/1 [==============================] - 0s 17ms/step
Input sentence: I gave up
Actual sentence: ich gab auf
Decoded sentence: ich habe gekotzt
1/1 [==============================] - 0s 20ms/step
Input sentence: I am a man
Actual sentence: ich bin ein Mann
Decoded sentence: ich bin ein mann
1/1 [==============================] - 0s 16ms/step
Input sentence: Tom really got a bad deal
Actual sentence: tom hat wirklich ein schlechtes Geschäft gemacht
Decoded sentence: tom hat wirklich schlecht
1/1 [==============================] - 0s 16ms/step
Input sentence: he will go with us
Actual sentence: er wird mit uns gehen
Decoded sentence: er soll mit dem gehen
1/1 [==============================] - 0s 18ms/step
Input sentence: what are you reading
Actual sentence: was liest du?
Decoded sentence: was liest mich
1/1 [==============================] - 0s 18ms/step
```

```
Input sentence: hi
Actual sentence: hallo
Decoded sentence: hallo
1/1 [==============================] - 0s 17ms/step
Input sentence: The reason does not matter
Actual sentence: der Grund ist egal
Decoded sentence: der wirtschaft geht es mir schlecht
1/1 [==============================] - 0s 17ms/step
Input sentence: I love your cat
Actual sentence: ich liebe deine Katze
Decoded sentence: ich liebe deine katze
1/1 [==============================] - 0s 18ms/step
Input sentence: go
Actual sentence: geh
Decoded sentence: geh
1/1 [==============================] - 0s 22ms/step
Input sentence: get inside
Actual sentence: Komm herein
Decoded sentence: geh rein
```

9 Bleu Score of model Translation

[14]:
```
from nltk.translate.bleu_score import corpus_bleu
references = []
hypotheses = []
for i in range(len(listt)):
    input_seq=listt[i]
    input_seq.lower().replace('[^\w\s]', '')
    new_eng_integer_encoded=eng_token.texts_to_sequences([input_seq])
   ␣
 ↪new_encoder_input_data=pad_sequences(new_eng_integer_encoded,maxlen=max_encoder_seq_length,
    actual_sentence = actual[i]
    decoded_sentence = decode_sequence(new_encoder_input_data)
    actual_sentence=actual_sentence.strip()
    decoded_sentence=decoded_sentence.strip()
    references.append([actual_sentence.split()])
    hypotheses.append(decoded_sentence.split())
bleu_score = corpus_bleu(references, hypotheses)
print('BLEU score:', bleu_score)
```

```
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 17ms/step
```

```
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 18ms/step
BLEU score: 0.3805647320367025

/opt/conda/lib/python3.10/site-packages/nltk/translate/bleu_score.py:490:
UserWarning:
Corpus/Sentence contains 0 counts of 4-gram overlaps.
BLEU scores might be undesirable; use SmoothingFunction().
  warnings.warn(_msg)
```

[ ]: