```
import numpy as np
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, Dense

# Dummy dataset (replace with your actual data)
input_texts = ['Hello', 'How are you?', 'Goodbye']
target_texts = ['Hallo', 'Wie geht es dir?', 'Auf Wiedersehen']

# Tokenization and vocabulary creation
input_vocab = set(' '.join(input_texts))
target_vocab = set(' '.join(target_texts))

input_token_index = dict((c, i) for i, c in enumerate(input_vocab))
target_token_index = dict((c, i) for i, c in enumerate(target_vocab))

num_encoder_tokens = len(input_vocab)
num_decoder_tokens = len(target_vocab)

max_encoder_seq_length = max(len(seq) for seq in input_texts)
max_decoder_seq_length = max(len(seq) for seq in target_texts)

# Data preprocessing
encoder_input_data = np.zeros((len(input_texts), max_encoder_seq_length, num_encoder_tokens), dtype='float32')
decoder_input_data = np.zeros((len(input_texts), max_decoder_seq_length, num_decoder_tokens), dtype='float32')
decoder_target_data = np.zeros((len(input_texts), max_decoder_seq_length, num_decoder_tokens), dtype='float32')

for i, (input_text, target_text) in enumerate(zip(input_texts, target_texts)):
    for t, char in enumerate(input_text):
        encoder_input_data[i, t, input_token_index[char]] = 1.0
    for t, char in enumerate(target_text):
        decoder_input_data[i, t, target_token_index[char]] = 1.0
        if t > 0:
            decoder_target_data[i, t - 1, target_token_index[char]] = 1.0

# Define the model
latent_dim = 256

encoder_inputs = Input(shape=(None, num_encoder_tokens))
encoder = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_inputs)
encoder_states = [state_h, state_c]

decoder_inputs = Input(shape=(None, num_decoder_tokens))
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_inputs, 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)

# Compile and train the model
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit([encoder_input_data, decoder_input_data], decoder_target_data, epochs=100, batch_size=1, validation_split=0.2)

# Use the trained model for inference
encoder_model = Model(encoder_inputs, encoder_states)

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_outputs, state_h, state_c = decoder_lstm(decoder_inputs, initial_state=decoder_states_inputs)
decoder_states = [state_h, state_c]
decoder_outputs = decoder_dense(decoder_outputs)

decoder_model = Model([decoder_inputs] + decoder_states_inputs, [decoder_outputs] + decoder_states)

# Add the end-of-sequence token to the target_token_index
target_token_index['\n'] = len(target_token_index)

def translate_sentence(input_sentence):
    input_seq = np.zeros((1, max_encoder_seq_length, num_encoder_tokens))
    for t, char in enumerate(input_sentence):
        input_seq[0, t, input_token_index[char]] = 1.0

    states_value = encoder_model.predict(input_seq)
    # Find the actual start-of-sequence token in your dataset
    start_token = '\t' if '\t' in target_token_index else list(target_token_index.keys())[0]
    end_token = '\n'    # Use the correct end-of-sequence token

    target_seq = np.zeros((1, 1, num_decoder_tokens))
    target_seq[0, 0, target_token_index[start_token]] = 1.0

    translated_sentence = ''
    stop_condition = False
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([target_seq] + states_value)

        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char = [char for char, index in target_token_index.items() if index == sampled_token_index][0]
        translated_sentence += sampled_char

        if sampled_char == end_token or len(translated_sentence) > max_decoder_seq_length:
            stop_condition = True

        target_seq = np.zeros((1, 1, num_decoder_tokens))
        target_seq[0, 0, sampled_token_index] = 1.0

        states_value = [h, c]

    return translated_sentence
```

```
Epoch 99/100
2/2 [==============================] - 0s 61ms/step - loss: 0.6576 - accuracy: 0.3438 - val_loss: 3.8644 - val_accuracy: 0.0000e+00
Epoch 100/100
2/2 [==============================] - 0s 63ms/step - loss: 0.7001 - accuracy: 0.3438 - val_loss: 3.9296 - val_accuracy: 0.0625
```