This is just a simple translation using tensorflow

```
!pip install tensorflow
!pip install tensorflow_hub
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.15.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.5.26)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (16.0.6)
Requirement already satisfied: ml-dtypes~=0.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.25.2)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.20.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.9.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.36.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.60.1)
Requirement already satisfied: tensorboard<2.16,>=2.15 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.2)
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: keras<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow) (0.42.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (2.27.0)
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (1.2.0)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (3.5.2)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (2.31.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (3.0.1)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (5.3.2)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (0.3.0)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow) (1.3.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow) (2024.2.2)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.16,>=2.15->tensorflow) (2.1.5)
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (0.5.1)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow) (3.2.2)
Requirement already satisfied: tensorflow_hub in /usr/local/lib/python3.10/dist-packages (0.16.1)
Requirement already satisfied: numpy>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow_hub) (1.25.2)
Requirement already satisfied: protobuf>=3.19.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow_hub) (3.20.3)
Requirement already satisfied: tf-keras>=2.14.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow_hub) (2.15.0)
```

```
import tensorflow as tf
import tensorflow_hub as hub
```

```
# Load the pre-trained model
model = hub.load("https://tfhub.dev/google/universal-sentence-encoder/4")
```

```
# Prepare the input text
input_text = tf.constant(["This is a test sentence."])
```

```
# Translate the input text
translated_text = model(input_text)
```

```
# Print the translated text
print(translated_text)
```

```
        -3.01300883e-02 -2.89490055e-02  3.17230970e-02  6.68863580e-02
        -1.41959367e-02  7.58978128e-02  1.65919792e-02  1.04179636e-01
        -4.21793461e-02  2.60824393e-02  3.10045574e-02 -4.25525084e-02
        -1.90855954e-02 -2.93722581e-02 -6.37147278e-02 -2.45957691e-02
        -2.85304412e-02  3.54192629e-02 -4.08651121e-02  5.81426127e-03
         8.23812634e-02  3.26108630e-03  3.11811864e-02 -3.80185544e-02
         2.16928087e-02 -7.43319979e-03  4.61075343e-02 -3.72890905e-02
        -2.30689142e-02 -3.26360725e-02 -5.67682907e-02 -2.39231829e-02
         6.74895793e-02  1.01416958e-02  1.90340001e-02  1.80256180e-02
         6.71907738e-02  9.26855113e-03 -7.28064030e-02 -2.94844843e-02
        -2.20977180e-02 -3.33367735e-02  3.82499173e-02  4.15171571e-02
        -4.14510034e-02 -7.14882761e-02  4.13339920e-02  2.89827920e-02
        -4.07406242e-06 -7.40425065e-02  6.65657921e-03  8.49623978e-03
        -7.24824294e-02 -5.59931323e-02  5.95660589e-04 -6.11735834e-03
        -3.35865356e-02  3.55748343e-03  2.41048858e-02 -1.57586448e-02
         3.27728651e-02  7.71152377e-02  7.02363206e-03 -3.13840769e-02
        -7.71562457e-02 -5.64544275e-02  1.74520016e-02  4.91298735e-02
        -8.10368806e-02  2.10891683e-02 -4.04082499e-02 -2.88767763e-03
         4.01595272e-02  2.28060596e-02  3.83489169e-02 -7.37585053e-02
        -9.80521813e-02 -1.53557146e-02 -5.51318377e-02  2.17683241e-02
        -9.19746906e-02  3.01940311e-02  1.41931570e-03 -1.15337002e-03
        -3.12948935e-02  1.38553027e-02  3.28962039e-03  2.55961232e-02
        -4.92589362e-03 -3.89230326e-02  2.61092260e-02 -6.54028580e-02
         6.38111234e-02  5.69475144e-02 -3.15414090e-03 -7.54501820e-02
        -2.28602607e-02 -2.64068730e-02 -9.70430002e-02  3.85353118e-02
         1.85709074e-02 -6.66959509e-02 -3.70545015e-02  8.27716365e-02
        -3.84048745e-02  3.41351926e-02 -9.85209271e-02 -3.24625373e-02
        -5.78852072e-02  3.86500210e-02  4.31780443e-02  2.49577351e-02
        -3.31829153e-02  1.24967294e-02 -2.18769579e-04  3.47556099e-02
        -1.95759591e-02 -1.05974056e-01  2.56895106e-02 -4.45117727e-02
        -1.85238682e-02 -1.67316739e-02 -2.47752648e-02 -7.21923560e-02
        -4.93568704e-02  5.67485578e-03 -4.06870060e-02 -2.55157165e-02
        -3.64671163e-02  5.05664200e-02  7.32817277e-02 -8.63996148e-03
         3.93536836e-02 -3.34311500e-02  6.03819564e-02  1.39662679e-02
         3.53739560e-02 -8.90377723e-03  4.59648063e-03 -4.32675779e-02
         2.14483682e-03 -9.44860727e-02  1.61788967e-02 -6.86139539e-02
        -5.86694665e-02  2.87320907e-03  9.22455732e-03  4.90848832e-02
        -5.22798300e-02  4.72008996e-02  3.48760523e-02 -3.84825952e-02
         1.02138976e-02  4.79665399e-02  3.40327658e-02 -3.32646482e-02
         3.14922594e-02  3.64791378e-02  1.53155485e-02 -9.71562192e-02
        -7.40751298e-03 -9.85972285e-02 -8.43001530e-02 -7.17975246e-03
         1.90600045e-02 -5.32503752e-03  3.90672237e-02  1.00903269e-02
         2.43073404e-02  6.82938518e-03 -1.32313725e-02  5.88762611e-02
         3.35316435e-02 -1.46035654e-02  2.31467038e-02  7.77961267e-03
         2.41037421e-02  4.82568219e-02 -4.38138954e-02  7.78268790e-03
         4.59212624e-02  3.69006880e-02 -1.17429402e-02 -4.46420396e-03
         7.83799961e-03 -6.61487132e-02  5.45003393e-04  4.02018093e-02
        -2.64081042e-02  3.04416697e-02 -6.51315376e-02  5.57356328e-02
        -4.47253212e-02 -7.74192512e-02 -5.12564406e-02  9.08388000e-04
         6.20976882e-03  2.24201214e-02  1.85031984e-02  4.52402085e-02
         1.01687852e-03 -2.40422506e-02  8.45768303e-02 -5.08780964e-02
        -3.45358299e-03 -6.48599537e-03 -1.08858727e-01  1.12834619e-02
        -5.64255565e-03  4.32399027e-02  4.50733490e-03 -7.15910345e-02
         2.71526985e-02 -5.62557019e-02  4.33727503e-02  3.62941921e-02
         2.20444445e-02  6.03780197e-03  5.13474680e-02 -2.36744303e-02
        -2.15599015e-02 -4.04884256e-02 -1.01267256e-01  1.89443342e-02
         7.56918788e-02 -8.41497332e-02  3.43204923e-02 -1.59713032e-03
        -2.58260313e-02 -7.61800483e-02 -1.12202959e-02  6.65166974e-02]], shape=(1, 512), dtype=float32)
```

To implement a simple neural machine translation (NMT) model in Python, we can use the TensorFlow and Keras libraries. TensorFlow is a popular deep learning framework that provides tools for building and training neural networks, while Keras is a high-level neural networks API that runs on top of TensorFlow.

Here's a step-by-step explanation and implementation of a simple NMT model using TensorFlow and Keras:

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

Prepare Data: We need a dataset containing pairs of sentences in two languages (source and target). For simplicity, we'll use dummy data.

```
# Dummy data
input_texts = ['hello', 'good morning', 'how are you']
target_texts = ['bonjour', 'bonjour', 'comment ça va']

# Generate vocabulary
input_vocab = set()
target_vocab = set()
for input_text, target_text in zip(input_texts, target_texts):
    input_vocab.update(input_text)
    target_vocab.update(target_text)

input_vocab = sorted(input_vocab)
target_vocab = sorted(target_vocab)

# Add special tokens for padding and start/end of sequence
input_vocab_size = len(input_vocab) + 2
target_vocab_size = len(target_vocab) + 2

# Create dictionaries to map characters to indices and vice versa
input_token_index = dict([(char, i+1) for i, char in enumerate(input_vocab)])
target_token_index = dict([(char, i+1) for i, char in enumerate(target_vocab)])

reverse_input_char_index = dict((i, char) for char, i in input_token_index.items())
reverse_target_char_index = dict((i, char) for char, i in target_token_index.items())

# Define maximum sequence lengths
max_encoder_seq_length = max([len(txt) for txt in input_texts])
max_decoder_seq_length = max([len(txt) for txt in target_texts])

# Initialize arrays for encoder and decoder inputs
encoder_input_data = np.zeros((len(input_texts), max_encoder_seq_length, input_vocab_size), dtype='float32')
decoder_input_data = np.zeros((len(input_texts), max_decoder_seq_length, target_vocab_size), dtype='float32')
decoder_target_data = np.zeros((len(input_texts), max_decoder_seq_length, target_vocab_size), dtype='float32')

# Fill the arrays with one-hot encoding
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: We'll create a simple sequence-to-sequence model with an LSTM encoder and decoder.

```
# Define input sequence
encoder_inputs = Input(shape=(None, input_vocab_size))
# LSTM encoding
encoder = LSTM(256, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_inputs)
# Discard encoder outputs, we only need the states
encoder_states = [state_h, state_c]

# Set up the decoder, using encoder_states as initial state
decoder_inputs = Input(shape=(None, target_vocab_size))
# We set up our decoder to return full output sequences and to return internal states as well
decoder_lstm = LSTM(256, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_inputs, initial_state=encoder_states)
decoder_dense = Dense(target_vocab_size, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

# Define the model that will turn
# `encoder_input_data` & `decoder_input_data` into `decoder_target_data`
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

Compile and Train the Model: Compile the model with appropriate loss function and optimizer, then fit the model to the data.

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
model.fit([encoder_input_data, decoder_input_data], decoder_target_data, batch_size=1, epochs=50, validation_split=0.2)
```

```
Epoch 1/50
2/2 [==============================] - 4s 1s/step - loss: 1.2676 - val_loss: 2.5636
Epoch 2/50
2/2 [==============================] - 0s 91ms/step - loss: 1.2222 - val_loss: 2.5753
Epoch 3/50
2/2 [==============================] - 0s 97ms/step - loss: 1.1608 - val_loss: 2.6765
Epoch 4/50
2/2 [==============================] - 0s 105ms/step - loss: 0.9860 - val_loss: 2.9888
Epoch 5/50
2/2 [==============================] - 0s 88ms/step - loss: 0.7581 - val_loss: 2.9783
Epoch 6/50
2/2 [==============================] - 0s 98ms/step - loss: 0.6612 - val_loss: 3.0365
Epoch 7/50
2/2 [==============================] - 0s 110ms/step - loss: 0.5937 - val_loss: 3.2744
Epoch 8/50
2/2 [==============================] - 0s 120ms/step - loss: 0.5306 - val_loss: 3.1573
Epoch 9/50
2/2 [==============================] - 0s 145ms/step - loss: 0.5890 - val_loss: 3.5419
Epoch 10/50
2/2 [==============================] - 0s 60ms/step - loss: 0.4633 - val_loss: 3.5553
Epoch 11/50
2/2 [==============================] - 0s 57ms/step - loss: 0.4380 - val_loss: 3.6005
Epoch 12/50
2/2 [==============================] - 0s 58ms/step - loss: 0.4994 - val_loss: 3.7645
Epoch 13/50
2/2 [==============================] - 0s 70ms/step - loss: 0.5144 - val_loss: 3.5281
Epoch 14/50
2/2 [==============================] - 0s 58ms/step - loss: 0.4292 - val_loss: 3.6488
Epoch 15/50
2/2 [==============================] - 0s 59ms/step - loss: 0.3893 - val_loss: 3.3874
Epoch 16/50
2/2 [==============================] - 0s 56ms/step - loss: 0.4906 - val_loss: 3.7700
Epoch 17/50
2/2 [==============================] - 0s 60ms/step - loss: 0.3658 - val_loss: 3.8665
Epoch 18/50
2/2 [==============================] - 0s 59ms/step - loss: 0.4504 - val_loss: 3.6383
Epoch 19/50
2/2 [==============================] - 0s 61ms/step - loss: 0.3617 - val_loss: 3.6916
Epoch 20/50
2/2 [==============================] - 0s 66ms/step - loss: 0.3366 - val_loss: 3.5291
Epoch 21/50
2/2 [==============================] - 0s 57ms/step - loss: 0.4122 - val_loss: 3.7942
Epoch 22/50
2/2 [==============================] - 0s 57ms/step - loss: 0.3103 - val_loss: 3.6386
Epoch 23/50
2/2 [==============================] - 0s 65ms/step - loss: 0.3507 - val_loss: 3.5251
Epoch 24/50
2/2 [==============================] - 0s 59ms/step - loss: 0.4078 - val_loss: 3.8488
Epoch 25/50
2/2 [==============================] - 0s 63ms/step - loss: 0.2887 - val_loss: 3.8408
Epoch 26/50
2/2 [==============================] - 0s 66ms/step - loss: 0.2877 - val_loss: 3.9479
Epoch 27/50
2/2 [==============================] - 0s 62ms/step - loss: 0.3778 - val_loss: 3.7542
Epoch 28/50
2/2 [==============================] - 0s 60ms/step - loss: 0.2884 - val_loss: 3.8105
Epoch 29/50
2/2 [==============================] - 0s 63ms/step - loss: 0.2600 - val_loss: 3.7580
```

Inference: To test the model, we need to set up the inference mode separately.

```
# Define encoder model
encoder_model = Model(encoder_inputs, encoder_states)

# Define decoder model
decoder_state_input_h = Input(shape=(256,))
decoder_state_input_c = Input(shape=(256,))
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)
```

```
print(target_token_index)
```

```
{' ': 1, 'a': 2, 'b': 3, 'c': 4, 'e': 5, 'j': 6, 'm': 7, 'n': 8, 'o': 9, 'r': 10, 't': 11, 'u': 12, 'v': 13, 'ç': 14}
```

```
if '\t' not in target_token_index:
    print('The character "\t" is not present in the target_token_index dictionary.')
```

```
The character "  " is not present in the target_token_index dictionary.
```

```
target_token_index['\t'] = len(target_token_index) + 1
```

To evaluate the performance and check the accuracy of the neural machine translation (NMT) model, we can perform inference on the test data and calculate the accuracy based on the model's predictions compared to the ground truth translations.

```
# Define a function to decode sequences
def decode_sequence(input_seq):
    # Encode the input sequence to get the internal state vectors
    states_value = encoder_model.predict(input_seq)

    # Generate empty target sequence of length 1
    target_seq = np.zeros((1, 1, target_vocab_size))
    # Populate the first character of target sequence with the start character
    target_seq[0, 0, target_token_index['\t']] = 1.0

    # Sampling loop for a batch of sequences
    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([target_seq] + states_value)

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char = reverse_target_char_index[sampled_token_index]
        decoded_sentence += sampled_char

        # Exit condition: either hit max length or find stop character
        if (sampled_char == '\n' or len(decoded_sentence) > max_decoder_seq_length):
            stop_condition = True

        # Update the target sequence (length 1).
        target_seq = np.zeros((1, 1, target_vocab_size))
        target_seq[0, 0, sampled_token_index] = 1.0

        # Update states
        states_value = [h, c]

    return decoded_sentence

# Define test data
test_input_texts = ['hello', 'good morning', 'how are you']
test_target_texts = ['bonjour', 'bonjour', 'comment ça va']

# Initialize variables for accuracy calculation
total_samples = len(test_input_texts)
correct_predictions = 0

# Iterate through test data
for input_text, target_text in zip(test_input_texts, test_target_texts):
    # Convert input sequence to one-hot encoding
    encoder_input_data = np.zeros((1, max_encoder_seq_length, input_vocab_size), dtype='float32')
    for t, char in enumerate(input_text):
        encoder_input_data[0, t, input_token_index[char]] = 1.0

    # Decode the input sequence
    decoded_sentence = decode_sequence(encoder_input_data)

    # Print the input, target, and predicted translations
    print('Input:', input_text)
    print('Target:', target_text)
    print('Predicted:', decoded_sentence)

    # Update accuracy count
    if decoded_sentence.strip() == target_text.strip():
        correct_predictions += 1

# Calculate accuracy
accuracy = (correct_predictions / total_samples) * 100
print('Accuracy:', accuracy, '%')
```

```
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 22ms/step
Input: hello
Target: bonjour
Predicted: onjourooooooooo
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 25ms/step
Input: good morning
Target: bonjour
Predicted: onjourooooooooo
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
Input: how are you
Target: comment ça va
Predicted: onjourooooooooo
Accuracy: 0.0 %
```

This code will perform inference on the test data, printing the input, target, and predicted translations for each example. Finally, it will calculate and print the accuracy of the model's translations compared to the ground truth translations.

Ensure that the model has been trained properly on suitable data and adjust the code accordingly if your dataset or model architecture differs.

2nd Part :- To implement another simple neural machine translation (NMT) model in Python, we can use the PyTorch library. PyTorch is a popular deep learning framework known for its flexibility and ease of use. Create a basic sequence-to-sequence model with an encoder and decoder architecture.

Here's the step-by-step explanation and implementation:

Import Libraries: We start by importing the necessary libraries.

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
```

Prepare Data: We need a dataset containing pairs of sentences in two languages (source and target). For simplicity, use dummy data similar to the previous example.

```
# Dummy data
input_texts = ['hello', 'good morning', 'how are you']
target_texts = ['bonjour', 'bonjour', 'comment ça va']

# Generate vocabulary
input_vocab = set()
target_vocab = set()
for input_text, target_text in zip(input_texts, target_texts):
    input_vocab.update(input_text)
    target_vocab.update(target_text)

input_vocab = sorted(input_vocab)
target_vocab = sorted(target_vocab)

# Add special tokens for padding and start/end of sequence
input_vocab_size = len(input_vocab) + 2
target_vocab_size = len(target_vocab) + 2
```

Define the Model: Create a simple sequence-to-sequence model with an encoder and decoder architecture using PyTorch.

```
input_token_index = dict([(char, i+1) for i, char in enumerate(input_vocab)])
class Encoder(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(Encoder, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)

    def forward(self, input):
        embedded = self.embedding(input).view(1, 1, -1)
        output, hidden = self.gru(embedded)
        return output, hidden

class Decoder(nn.Module):
    def __init__(self, hidden_size, output_size):
        super(Decoder, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        output = self.embedding(input).view(1, 1, -1)
        output = nn.functional.relu(output)
        output, hidden = self.gru(output, hidden)
        output = self.softmax(self.out(output[0]))
        return output, hidden
```

Training the Model: Need to define functions for training the model.

```
def train(input_tensor, target_tensor, encoder, decoder, encoder_optimizer, decoder_optimizer, criterion, max_length=max_decoder_seq_length):
    encoder_hidden = encoder.init_hidden()

    encoder_optimizer.zero_grad()
    decoder_optimizer.zero_grad()

    input_length = input_tensor.size(0)
    target_length = target_tensor.size(0)

    loss = 0

    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(input_tensor[ei], encoder_hidden)

    decoder_input = torch.tensor([[SOS_token]])

    decoder_hidden = encoder_hidden

    for di in range(target_length):
        decoder_output, decoder_hidden = decoder(decoder_input, decoder_hidden)
        topv, topi = decoder_output.topk(1)
        decoder_input = topi.squeeze().detach()

        loss += criterion(decoder_output, target_tensor[di])
        if decoder_input.item() == EOS_token:
            break

    loss.backward()

    encoder_optimizer.step()
    decoder_optimizer.step()

    return loss.item() / target_length

def trainIters(encoder, decoder, n_iters, print_every=1000, plot_every=100, learning_rate=0.01):
    plot_losses = []
    print_loss_total = 0  # Reset every print_every
    plot_loss_total = 0  # Reset every plot_every

    encoder_optimizer = optim.SGD(encoder.parameters(), lr=learning_rate)
    decoder_optimizer = optim.SGD(decoder.parameters(), lr=learning_rate)
    criterion = nn.NLLLoss()

    for iter in range(1, n_iters + 1):
        training_pair = tensorsFromPair(random.choice(pairs))
        input_tensor = training_pair[0]
        target_tensor = training_pair[1]

        loss = train(input_tensor, target_tensor, encoder,
                     decoder, encoder_optimizer, decoder_optimizer, criterion)
        print_loss_total += loss
        plot_loss_total += loss

        if iter % print_every == 0:
            print_loss_avg = print_loss_total / print_every
            print_loss_total = 0
            print('(%d %d%%) %.4f' % (iter, iter / n_iters * 100, print_loss_avg))
```

Inference: Need to define functions for inference and evaluation.

```
def evaluate(encoder, decoder, sentence, max_length=max_decoder_seq_length):
    with torch.no_grad():
        input_tensor = tensorFromSentence(input_lang, sentence)
        input_length = input_tensor.size()[0]
        encoder_hidden = encoder.init_hidden()

        for ei in range(input_length):
            encoder_output, encoder_hidden = encoder(input_tensor[ei], encoder_hidden)

        decoder_input = torch.tensor([[SOS_token]])
        decoder_hidden = encoder_hidden
        decoded_words = []

        for di in range(max_length):
            decoder_output, decoder_hidden = decoder(decoder_input, decoder_hidden)
            topv, topi = decoder_output.data.topk(1)
            if topi.item() == EOS_token:
                decoded_words.append('<EOS>')
                break
            else:
                decoded_words.append(output_lang.index2word[topi.item()])

            decoder_input = topi.squeeze().detach()

        return decoded_words
```

Conclusion: This implementation provides a basic understanding of how an NMT model works using a sequence-to-sequence architecture with an encoder and decoder. To evaluate the model's performance, we can perform inference on the test data, compare the predicted translations with the ground truth translations, and compute the error rate or accuracy.

Now, to perform inference and check the accuracy by computing the error rate, you need to train the model using the provided training data, then evaluate it on the test data. You'll compare the model's predictions with the actual translations and calculate the accuracy accordingly.

Ensure that the model is trained properly and adjust the code accordingly if your dataset or model architecture differs.

```
!ls
```

```
sample_data
```

```
import os
print(os.getcwd())
```

```
/content
```

```
!pip show torch
```