Importing

```
import tensorflow as tf
import os
```

Data Loading

```
java_train_path = '/content/drive/MyDrive/java_to_python_translator/java_programs_train.tfrecord'
python_train_path = '/content/drive/MyDrive/java_to_python_translator/python_programs_train.tfrecord'

java_valid_path = '/content/drive/MyDrive/java_to_python_translator/java_programs_valid.tfrecord'
python_valid_path = '/content/drive/MyDrive/java_to_python_translator/python_programs_valid.tfrecord'

java_test_path = '/content/drive/MyDrive/java_to_python_translator/java_programs_test.tfrecord'
python_test_path = '/content/drive/MyDrive/java_to_python_translator/python_programs_test.tfrecord'
```

Data Preprocessing: Converting to proper format and tokenizing

```
def parse_code(example_proto):
    feature_description = {
        'code': tf.io.FixedLenFeature([], tf.string),
    }
    parsed_example = tf.io.parse_single_example(example_proto, feature_description)
    return parsed_example['code']



def load_paired_dataset(java_path, python_path, batch_size=32, buffer_size=1000):
    java_dataset = tf.data.TFRecordDataset(java_path).map(parse_code)
    python_dataset = tf.data.TFRecordDataset(python_path).map(parse_code)

    paired_dataset = tf.data.Dataset.zip((java_dataset, python_dataset))
    paired_dataset = paired_dataset.shuffle(buffer_size).batch(batch_size).prefetch(tf.data.AUTOTUNE)

    return paired_dataset

train_dataset = load_paired_dataset(java_train_path, python_train_path)
valid_dataset = load_paired_dataset(java_valid_path, python_valid_path)
test_dataset  = load_paired_dataset(java_test_path, python_test_path)



from tensorflow.keras.layers import TextVectorization

vocab_size = 10000
sequence_length = 150

java_vectorizer = TextVectorization(max_tokens=vocab_size, output_sequence_length=sequence_length)
python_vectorizer = TextVectorization(max_tokens=vocab_size, output_sequence_length=sequence_length)

# Prepare text datasets for vocab adaptation
java_texts = train_dataset.map(lambda java, py: java)
python_texts = train_dataset.map(lambda java, py: py)

# Build vocab
java_vectorizer.adapt(java_texts)
python_vectorizer.adapt(python_texts)



def preprocess(java, python):
    java = java_vectorizer(java)
    python_inp = python_vectorizer(python)
    python_tar = tf.concat([python_inp[:, 1:], tf.zeros((tf.shape(python_inp)[0], 1), dtype=python_inp.dtype)], axis=1)
    return (java, python_inp), python_tar
```

```
train_dataset = train_dataset.map(preprocess)
valid_dataset = valid_dataset.map(preprocess)
test_dataset = test_dataset.map(preprocess)
```

Model Building

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, LSTM, Dense

embedding_dim = 256
lstm_units = 512

# Encoder
encoder_inputs = Input(shape=(None,))
enc_emb = Embedding(vocab_size, embedding_dim)(encoder_inputs)
encoder_outputs, state_h, state_c = LSTM(lstm_units, return_state=True)(enc_emb)
encoder_states = [state_h, state_c]

# Decoder
decoder_inputs = Input(shape=(None,))
dec_emb = Embedding(vocab_size, embedding_dim)(decoder_inputs)
decoder_lstm = LSTM(lstm_units, return_sequences=True)
decoder_outputs = decoder_lstm(dec_emb, initial_state=encoder_states)
decoder_dense = Dense(vocab_size, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "functional"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, None)	0	-
input_layer_1 (InputLayer)	(None, None)	0	-
embedding (Embedding)	(None, None, 256)	2,560,000	input_layer[0][0]
embedding_1 (Embedding)	(None, None, 256)	2,560,000	input_layer_1[0]…
lstm (LSTM)	[(None, 512), (None, 512), (None, 512)]	1,574,912	embedding[0][0]
lstm_1 (LSTM)	(None, None, 512)	1,574,912	embedding_1[0][0… lstm[0][1], lstm[0][2]
dense (Dense)	(None, None, 10000)	5,130,000	lstm_1[0][0]

```
Total params: 13,399,824 (51.12 MB)
Trainable params: 13,399,824 (51.12 MB)
```

Model Training

```
model.fit(train_dataset, validation_data=valid_dataset, epochs=1)
```

```
493/493 ━━━━━━━━━━━━━━━━━━━━ 4721s 10s/step - accuracy: 0.7162 - loss: 1.4734 - val_accuracy: 0.7183 - val_loss: 1.4078
<keras.src.callbacks.history.History at 0x78e701cc6c50>
```

Model Evaluation

```
loss, accuracy = model.evaluate(test_dataset)
```

```
print(f"Test Loss: {loss:.4f}, Accuracy: {accuracy:.4f}")
```

```
132/132 ━━━━━━━━━━━━━━━━━━━━ 395s 3s/step - accuracy: 0.7135 - loss: 1.4402
Test Loss: 1.4462, Accuracy: 0.7120
```

Model Saving

```
model.save("/content/drive/MyDrive/java_to_python_translator/java_to_python_seq2seq_model.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consi
```

```
model.save('/content/drive/MyDrive/java_to_python_translator/seq2seq_kerasFormat_model.keras')
```