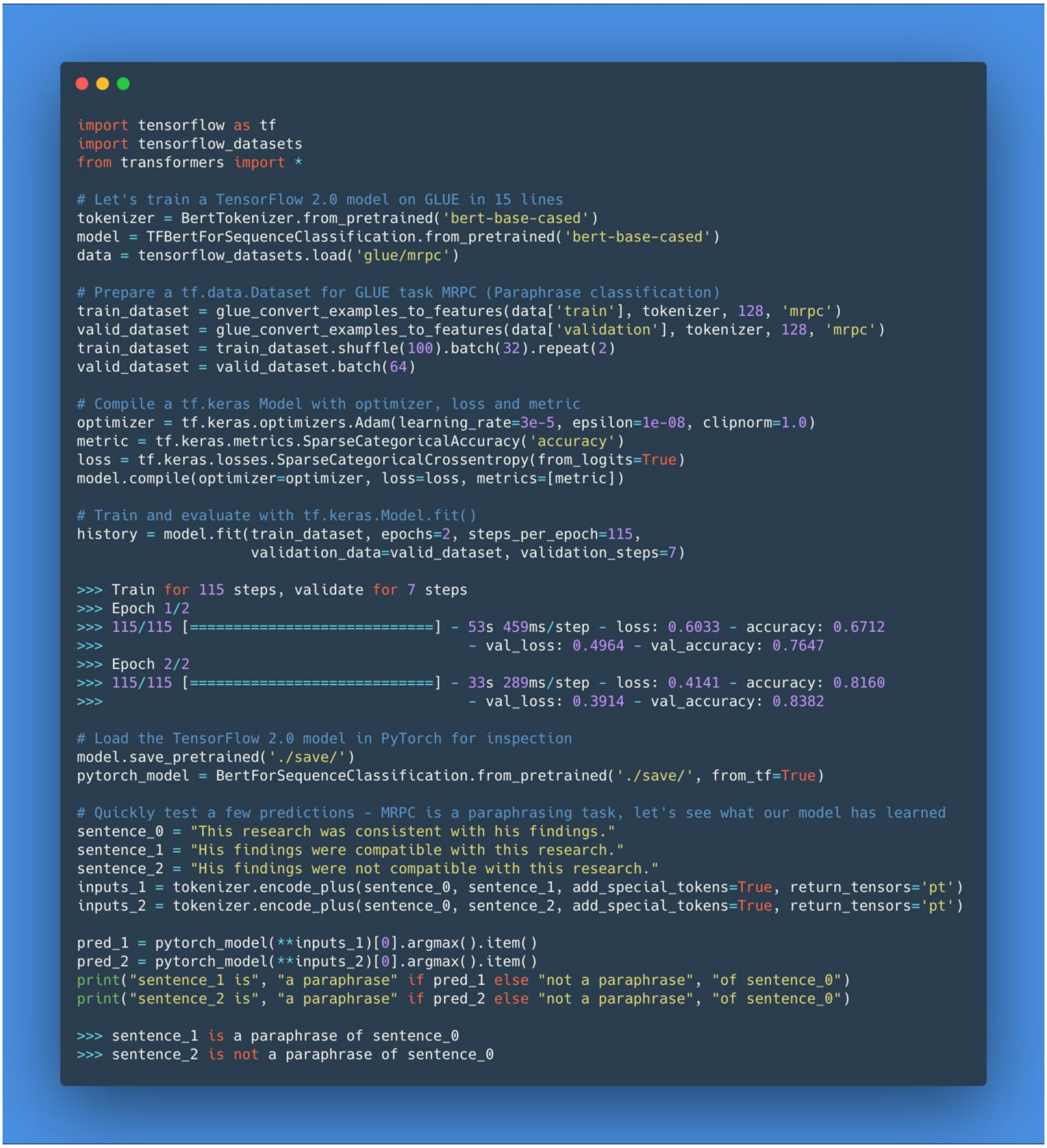

```
import tensorflow as tf
import tensorflow_datasets
from transformers import *

# Let's train a TensorFlow 2.0 model on GLUE in 15 lines
tokenizer = BertTokenizer.from_pretrained('bert-base-cased')
model = TFBertForSequenceClassification.from_pretrained('bert-base-cased')
data = tensorflow_datasets.load('glue/mrpc')

# Prepare a tf.data.Dataset for GLUE task MRPC (Paraphrase classification)
train_dataset = glue_convert_examples_to_features(data['train'], tokenizer, 128, 'mrpc')
valid_dataset = glue_convert_examples_to_features(data['validation'], tokenizer, 128, 'mrpc')
train_dataset = train_dataset.shuffle(100).batch(32).repeat(2)
valid_dataset = valid_dataset.batch(64)

# Compile a tf.keras Model with optimizer, loss and metric
optimizer = tf.keras.optimizers.Adam(learning_rate=3e-5, epsilon=1e-08, clipnorm=1.0)
metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer=optimizer, loss=loss, metrics=[metric])

# Train and evaluate with tf.keras.Model.fit()
history = model.fit(train_dataset, epochs=2, steps_per_epoch=115,
                    validation_data=valid_dataset, validation_steps=7)

>>> Train for 115 steps, validate for 7 steps
>>> Epoch 1/2
>>> 115/115 [============================] - 53s 459ms/step - loss: 0.6033 - accuracy: 0.6712
>>>                                        - val_loss: 0.4964 - val_accuracy: 0.7647
>>> Epoch 2/2
>>> 115/115 [============================] - 33s 289ms/step - loss: 0.4141 - accuracy: 0.8160
>>>                                        - val_loss: 0.3914 - val_accuracy: 0.8382

# Load the TensorFlow 2.0 model in PyTorch for inspection
model.save_pretrained('./save/')
pytorch_model = BertForSequenceClassification.from_pretrained('./save/', from_tf=True)

# Quickly test a few predictions - MRPC is a paraphrasing task, let's see what our model has learned
sentence_0 = "This research was consistent with his findings."
sentence_1 = "His findings were compatible with this research."
sentence_2 = "His findings were not compatible with this research."
inputs_1 = tokenizer.encode_plus(sentence_0, sentence_1, add_special_tokens=True, return_tensors='pt')
inputs_2 = tokenizer.encode_plus(sentence_0, sentence_2, add_special_tokens=True, return_tensors='pt')

pred_1 = pytorch_model(**inputs_1)[0].argmax().item()
pred_2 = pytorch_model(**inputs_2)[0].argmax().item()
print("sentence_1 is", "a paraphrase" if pred_1 else "not a paraphrase", "of sentence_0")
print("sentence_2 is", "a paraphrase" if pred_2 else "not a paraphrase", "of sentence_0")

>>> sentence_1 is a paraphrase of sentence_0
>>> sentence_2 is not a paraphrase of sentence_0
```