```
import os

os.environ["KERAS_BACKEND"] = "tensorflow"

import pathlib
import random
import string
import re
import numpy as np

import tensorflow.data as tf_data
import tensorflow.strings as tf_strings

import keras
from keras import layers
from keras import ops
from keras.layers import TextVectorization


import tensorflow as tf


text_file = keras.utils.get_file(
    fname="spa-eng.zip",
    origin="http://storage.googleapis.com/download.tensorflow.org/data/spa-eng.zip",
    extract=True,
)
text_file = pathlib.Path(text_file).parent /"spa-eng_extracted"/ "spa-eng" / "spa.txt"
```

```
Downloading data from http://storage.googleapis.com/download.tensorflow.org/data/spa-eng.zip
2638744/2638744 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

```
with open(text_file) as f:
    lines = f.read().split("\n")[:-1]
text_pairs = []
for line in lines:
    eng, spa = line.split("\t")
    spa = "[start] " + spa + " [end]"
    text_pairs.append((eng, spa))


for _ in range(5):
    print(random.choice(text_pairs))
```

```
("There's nobody there.", '[start] No hay nadie ahí. [end]')
('I have an identical twin.', '[start] Tengo un gemelo idéntico. [end]')
('She is brushing her hair.', '[start] Ella se está cepillando el pelo. [end]')
('Mary can cook anything without using a recipe.', '[start] Mary puede cocinar de todo sin utilizar recetas. [end]')
('Say goodbye.', '[start] Decí adiós. [end]')
```

```
random.shuffle(text_pairs)
num_val_samples = int(0.15 * len(text_pairs))
num_train_samples = len(text_pairs) - 2 * num_val_samples
train_pairs = text_pairs[:num_train_samples]
val_pairs = text_pairs[num_train_samples : num_train_samples + num_val_samples]
test_pairs = text_pairs[num_train_samples + num_val_samples :]

print(f"{len(text_pairs)} total pairs")
print(f"{len(train_pairs)} training pairs")
print(f"{len(val_pairs)} validation pairs")
print(f"{len(test_pairs)} test pairs")
```

```
118964 total pairs
83276 training pairs
17844 validation pairs
17844 test pairs
```

```
strip_chars = string.punctuation + "¿"
strip_chars = strip_chars.replace("[", "")
strip_chars = strip_chars.replace("]", "")

vocab_size = 15000
sequence_length = 20
batch_size = 64
```

```
def custom_standardization(input_string):
    lowercase = tf_strings.lower(input_string)
    return tf_strings.regex_replace(lowercase, "[%s]" % re.escape(strip_chars), "")


eng_vectorization = TextVectorization(
    max_tokens=vocab_size,
    output_mode="int",
    output_sequence_length=sequence_length,
)
spa_vectorization = TextVectorization(
    max_tokens=vocab_size,
    output_mode="int",
    output_sequence_length=sequence_length + 1,
    standardize=custom_standardization,
)
train_eng_texts = [pair[0] for pair in train_pairs]
train_spa_texts = [pair[1] for pair in train_pairs]
eng_vectorization.adapt(train_eng_texts)
spa_vectorization.adapt(train_spa_texts)


def format_dataset(eng, spa):
    eng = eng_vectorization(eng)
    spa = spa_vectorization(spa)
    return (
        {
            "encoder_inputs": eng,
            "decoder_inputs": spa[:, :-1],
        },
        spa[:, 1:],
    )


def make_dataset(pairs):
    eng_texts, spa_texts = zip(*pairs)
    eng_texts = list(eng_texts)
    spa_texts = list(spa_texts)
    dataset = tf_data.Dataset.from_tensor_slices((eng_texts, spa_texts))
    dataset = dataset.batch(batch_size)
    dataset = dataset.map(format_dataset)
    return dataset.cache().shuffle(2048).prefetch(16)


train_ds = make_dataset(train_pairs)
val_ds = make_dataset(val_pairs)


for inputs, targets in train_ds.take(1):
    print(f'inputs["encoder_inputs"].shape: {inputs["encoder_inputs"].shape}')
    print(f'inputs["decoder_inputs"].shape: {inputs["decoder_inputs"].shape}')
    print(f"targets.shape: {targets.shape}")
```

```
inputs["encoder_inputs"].shape: (64, 20)
inputs["decoder_inputs"].shape: (64, 20)
targets.shape: (64, 20)
```

```
import keras.ops as ops


class TransformerEncoder(layers.Layer):
    def __init__(self, embed_dim, dense_dim, num_heads, **kwargs):
        super().__init__(**kwargs)
        self.embed_dim = embed_dim
        self.dense_dim = dense_dim
        self.num_heads = num_heads
        self.attention = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim
        )
        self.dense_proj = keras.Sequential(
            [
                layers.Dense(dense_dim, activation="relu"),
                layers.Dense(embed_dim),
            ]
        )
        self.layernorm_1 = layers.LayerNormalization()
        self.layernorm_2 = layers.LayerNormalization()
```

```
            self.supports_masking = True

        def call(self, inputs, mask=None):
            if mask is not None:
                padding_mask = ops.cast(mask[:, None, :], dtype="int32")
            else:
                padding_mask = None

            attention_output = self.attention(
                query=inputs, value=inputs, key=inputs, attention_mask=padding_mask
            )
            proj_input = self.layernorm_1(inputs + attention_output)
            proj_output = self.dense_proj(proj_input)
            return self.layernorm_2(proj_input + proj_output)

        def get_config(self):
            config = super().get_config()
            config.update(
                {
                    "embed_dim": self.embed_dim,
                    "dense_dim": self.dense_dim,
                    "num_heads": self.num_heads,
                }
            )
            return config


    class PositionalEmbedding(layers.Layer):
        def __init__(self, sequence_length, vocab_size, embed_dim, **kwargs):
            super().__init__(**kwargs)
            self.token_embeddings = layers.Embedding(
                input_dim=vocab_size, output_dim=embed_dim
            )
            self.position_embeddings = layers.Embedding(
                input_dim=sequence_length, output_dim=embed_dim
            )
            self.sequence_length = sequence_length
            self.vocab_size = vocab_size
            self.embed_dim = embed_dim

        def call(self, inputs):
            length = ops.shape(inputs)[-1]
            positions = ops.arange(0, length, 1)
            embedded_tokens = self.token_embeddings(inputs)
            embedded_positions = self.position_embeddings(positions)
            return embedded_tokens + embedded_positions

        def compute_mask(self, inputs, mask=None):
            return ops.not_equal(inputs, 0)

        def get_config(self):
            config = super().get_config()
            config.update(
                {
                    "sequence_length": self.sequence_length,
                    "vocab_size": self.vocab_size,
                    "embed_dim": self.embed_dim,
                }
            )
            return config


    class TransformerDecoder(layers.Layer):
        def __init__(self, embed_dim, latent_dim, num_heads, **kwargs):
            super().__init__(**kwargs)
            self.embed_dim = embed_dim
            self.latent_dim = latent_dim
            self.num_heads = num_heads
            self.attention_1 = layers.MultiHeadAttention(
                num_heads=num_heads, key_dim=embed_dim
            )
            self.attention_2 = layers.MultiHeadAttention(
                num_heads=num_heads, key_dim=embed_dim
            )
            self.dense_proj = keras.Sequential(
                [
                    layers.Dense(latent_dim, activation="relu"),
```

```
                layers.Dense(embed_dim),
            ]
        )
        self.layernorm_1 = layers.LayerNormalization()
        self.layernorm_2 = layers.LayerNormalization()
        self.layernorm_3 = layers.LayerNormalization()
        self.supports_masking = True

    def call(self, inputs, mask=None):
        inputs, encoder_outputs = inputs
        causal_mask = self.get_causal_attention_mask(inputs)

        if mask is None:
            inputs_padding_mask, encoder_outputs_padding_mask = None, None
        else:
            inputs_padding_mask, encoder_outputs_padding_mask = mask

        attention_output_1 = self.attention_1(
            query=inputs,
            value=inputs,
            key=inputs,
            attention_mask=causal_mask,
            query_mask=inputs_padding_mask,
        )
        out_1 = self.layernorm_1(inputs + attention_output_1)

        attention_output_2 = self.attention_2(
            query=out_1,
            value=encoder_outputs,
            key=encoder_outputs,
            query_mask=inputs_padding_mask,
            key_mask=encoder_outputs_padding_mask,
        )
        out_2 = self.layernorm_2(out_1 + attention_output_2)

        proj_output = self.dense_proj(out_2)
        return self.layernorm_3(out_2 + proj_output)

    def get_causal_attention_mask(self, inputs):
        input_shape = ops.shape(inputs)
        batch_size, sequence_length = input_shape[0], input_shape[1]
        i = ops.arange(sequence_length)[:, None]
        j = ops.arange(sequence_length)
        mask = ops.cast(i >= j, dtype="int32")
        mask = ops.reshape(mask, (1, input_shape[1], input_shape[1]))
        mult = ops.concatenate(
            [ops.expand_dims(batch_size, -1), ops.convert_to_tensor([1, 1])],
            axis=0,
        )
        return ops.tile(mask, mult)

    def get_config(self):
        config = super().get_config()
        config.update(
            {
                "embed_dim": self.embed_dim,
                "latent_dim": self.latent_dim,
                "num_heads": self.num_heads,
            }
        )
        return config


embed_dim = 256
latent_dim = 2048
num_heads = 8

encoder_inputs = keras.Input(shape=(None,), dtype="int64", name="encoder_inputs")
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(encoder_inputs)
encoder_outputs = TransformerEncoder(embed_dim, latent_dim, num_heads)(x)
encoder = keras.Model(encoder_inputs, encoder_outputs)

decoder_inputs = keras.Input(shape=(None,), dtype="int64", name="decoder_inputs")
encoded_seq_inputs = keras.Input(shape=(None, embed_dim), name="decoder_state_inputs")
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(decoder_inputs)
x = TransformerDecoder(embed_dim, latent_dim, num_heads)([x, encoder_outputs])
x = layers.Dropout(0.5)(x)
```

```
decoder_outputs = layers.Dense(vocab_size, activation="softmax")(x)
decoder = keras.Model([decoder_inputs, encoded_seq_inputs], decoder_outputs)

transformer = keras.Model(
    {"encoder_inputs": encoder_inputs, "decoder_inputs": decoder_inputs},
    decoder_outputs,
    name="transformer",
)


epochs = 1  # This should be at least 30 for convergence

transformer.summary()
transformer.compile(
    "rmsprop",
    loss=keras.losses.SparseCategoricalCrossentropy(ignore_class=0),
    metrics=["accuracy"],
)
transformer.fit(train_ds, epochs=epochs, validation_data=val_ds)
```

Model: "transformer"

Layer (type)	Output Shape	Param #	Connected to
encoder_inputs (InputLayer)	(None, None)	0	-
decoder_inputs (InputLayer)	(None, None)	0	-
positional_embedding (PositionalEmbedding)	(None, None, 256)	3,845,120	encoder_inputs[0][0]
not_equal (NotEqual)	(None, None)	0	encoder_inputs[0][0]
positional_embedding_1 (PositionalEmbedding)	(None, None, 256)	3,845,120	decoder_inputs[0][0]
transformer_encoder (TransformerEncoder)	(None, None, 256)	3,155,456	positional_embedding[… not_equal[0][0]
not_equal_1 (NotEqual)	(None, None)	0	decoder_inputs[0][0]
transformer_decoder (TransformerDecoder)	(None, None, 256)	5,259,520	positional_embedding_… transformer_encoder[0… not_equal_1[0][0], not_equal[0][0]
dropout_3 (Dropout)	(None, None, 256)	0	transformer_decoder[0…
dense_4 (Dense)	(None, None, 15000)	3,855,000	dropout_3[0][0]

Total params: 19,960,216 (76.14 MB)
Trainable params: 19,960,216 (76.14 MB)

```
spa_vocab = spa_vectorization.get_vocabulary()
spa_index_lookup = dict(zip(range(len(spa_vocab)), spa_vocab))
max_decoded_sentence_length = 20


def decode_sequence(input_sentence):
    tokenized_input_sentence = eng_vectorization([input_sentence])
    decoded_sentence = "[start]"
    for i in range(max_decoded_sentence_length):
        tokenized_target_sentence = spa_vectorization([decoded_sentence])[:, :-1]
        predictions = transformer(
            {
                "encoder_inputs": tokenized_input_sentence,
                "decoder_inputs": tokenized_target_sentence,
            }
        )

        # ops.argmax(predictions[0, i, :]) is not a concrete value for jax here
        sampled_token_index = ops.convert_to_numpy(
            ops.argmax(predictions[0, i, :])
        ).item(0)
        sampled_token = spa_index_lookup[sampled_token_index]
        decoded_sentence += " " + sampled_token
```

```
        if sampled_token == "[end]":
            break
    return decoded_sentence


test_eng_texts = [pair[0] for pair in test_pairs]
for _ in range(30):
    input_sentence = random.choice(test_eng_texts)
    translated = decode_sequence(input_sentence)


translated
```

```
input_sentence
```

```
# Guardar el modelo entrenado en disco
transformer.save("translation_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
```

```
# Usar embeddings preentrenados (GloVe)
# Descargar GloVe si no está disponible
!wget http://nlp.stanford.edu/data/glove.6B.zip -O glove.6B.zip
!unzip -q glove.6B.zip -d glove/
```

```
--2025-03-21 03:06:17--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2025-03-21 03:06:17--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2025-03-21 03:06:18--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  4.55MB/s    in 2m 51s

2025-03-21 03:09:10 (4.80 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]
```

```
# Cargar embeddings de GloVe
def load_glove_embeddings(file_path, embedding_dim=100):
    embeddings_index = {}
    with open(file_path, encoding='utf-8') as f:
        for line in f:
            values = line.split()
            word = values[0]
            coefs = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = coefs
    return embeddings_index

glove_path = "glove/glove.6B.100d.txt"
embeddings_index = load_glove_embeddings(glove_path)


# Visualizar activaciones de las capas
def display_layer_activations(model, input_text):
    layer_outputs = [layer.output for layer in model.layers if 'dense' in layer.name or 'attention' in layer.name]
    activation_model = tf.keras.models.Model(inputs=model.input, outputs=layer_outputs)
    activations = activation_model.predict(input_text)

    for i, activation in enumerate(activations):
        print(f"Layer {i + 1} activations:")
        print(activation)
```

```
# Preparar datos de entrenamiento y validación
batch_size = 64
buffer_size = 20000
max_vocab_size = 20000
sequence_length = 50


# Crear vectorizadores para inglés y español
eng_vectorizer = TextVectorization(max_tokens=max_vocab_size, output_mode="int", output_sequence_length=sequence_length)
spa_vectorizer = TextVectorization(max_tokens=max_vocab_size, output_mode="int", output_sequence_length=sequence_length)


# Asegurar que las variables `text_pairs` estén definidas
random.shuffle(text_pairs)
num_val_samples = int(0.15 * len(text_pairs))
train_pairs = text_pairs[:-num_val_samples]
val_pairs = text_pairs[-num_val_samples:]


# Extraer textos de entrenamiento y ajustar los vectorizadores
eng_texts_train, spa_texts_train = zip(*train_pairs)
eng_vectorizer.adapt(list(eng_texts_train))
spa_vectorizer.adapt(list(spa_texts_train))

def format_dataset(pairs):
    eng_texts, spa_texts = zip(*pairs)
    eng_vectorized = eng_vectorizer(list(eng_texts))
    spa_vectorized = spa_vectorizer(list(spa_texts))

    return {"encoder_inputs": eng_vectorized, "decoder_inputs": spa_vectorized[:, :-1]}, spa_vectorized[:, 1:]

train_data = tf.data.Dataset.from_tensor_slices(format_dataset(train_pairs))
train_data = train_data.shuffle(buffer_size).batch(batch_size).prefetch(1)

val_data = tf.data.Dataset.from_tensor_slices(format_dataset(val_pairs))
val_data = val_data.batch(batch_size).prefetch(1)


# Ajustar hiperparámetros y entrenar nuevamente
new_epochs = 100  # Más de 30 epochs
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0005)  # Cambio de learning rate
transformer.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])


# Callback para registrar la información de cada época
class EpochLogger(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        logs = logs or {}
        print(f"Epoch {epoch + 1}: Loss = {logs.get('loss'):.4f}, Accuracy = {logs.get('accuracy'):.4f}")


# Entrenar con el callback
history = transformer.fit(
    train_data,
    epochs=new_epochs,
    validation_data=val_data,
    callbacks=[EpochLogger()]
)
```

```
Epoch 89/100
1579/1580 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8552 - loss: nanEpoch 89: Loss = nan, Accuracy = 0.8555
1580/1580 ━━━━━━━━━━━━━━━━━━━━ 29s 18ms/step - accuracy: 0.8552 - loss: nan - val_accuracy: 0.8553 - val_loss: nan
Epoch 90/100
1578/1580 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8553 - loss: nanEpoch 90: Loss = nan, Accuracy = 0.8555
1580/1580 ━━━━━━━━━━━━━━━━━━━━ 28s 18ms/step - accuracy: 0.8553 - loss: nan - val_accuracy: 0.8553 - val_loss: nan
Epoch 91/100
1579/1580 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8553 - loss: nanEpoch 91: Loss = nan, Accuracy = 0.8555
1580/1580 ━━━━━━━━━━━━━━━━━━━━ 29s 18ms/step - accuracy: 0.8553 - loss: nan - val_accuracy: 0.8553 - val_loss: nan
Epoch 92/100
1579/1580 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8551 - loss: nanEpoch 92: Loss = nan, Accuracy = 0.8555
1580/1580 ━━━━━━━━━━━━━━━━━━━━ 28s 18ms/step - accuracy: 0.8551 - loss: nan - val_accuracy: 0.8553 - val_loss: nan
Epoch 93/100
1579/1580 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8552 - loss: nanEpoch 93: Loss = nan, Accuracy = 0.8555
1580/1580 ━━━━━━━━━━━━━━━━━━━━ 29s 18ms/step - accuracy: 0.8552 - loss: nan - val_accuracy: 0.8553 - val_loss: nan
Epoch 94/100
1579/1580 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8551 - loss: nanEpoch 94: Loss = nan, Accuracy = 0.8555
1580/1580 ━━━━━━━━━━━━━━━━━━━━ 29s 18ms/step - accuracy: 0.8551 - loss: nan - val_accuracy: 0.8553 - val_loss: nan
Epoch 95/100
1579/1580 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8552 - loss: nanEpoch 95: Loss = nan, Accuracy = 0.8555
1580/1580 ━━━━━━━━━━━━━━━━━━━━ 30s 19ms/step - accuracy: 0.8552 - loss: nan - val_accuracy: 0.8553 - val_loss: nan
Epoch 96/100
1579/1580 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8553 - loss: nanEpoch 96: Loss = nan, Accuracy = 0.8555
1580/1580 ━━━━━━━━━━━━━━━━━━━━ 29s 18ms/step - accuracy: 0.8553 - loss: nan - val_accuracy: 0.8553 - val_loss: nan
Epoch 97/100
1579/1580 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8553 - loss: nanEpoch 97: Loss = nan, Accuracy = 0.8555
1580/1580 ━━━━━━━━━━━━━━━━━━━━ 29s 18ms/step - accuracy: 0.8553 - loss: nan - val_accuracy: 0.8553 - val_loss: nan
Epoch 98/100
1579/1580 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8554 - loss: nanEpoch 98: Loss = nan, Accuracy = 0.8555
1580/1580 ━━━━━━━━━━━━━━━━━━━━ 29s 18ms/step - accuracy: 0.8554 - loss: nan - val_accuracy: 0.8553 - val_loss: nan
Epoch 99/100
1579/1580 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8554 - loss: nanEpoch 99: Loss = nan, Accuracy = 0.8555
1580/1580 ━━━━━━━━━━━━━━━━━━━━ 29s 18ms/step - accuracy: 0.8554 - loss: nan - val_accuracy: 0.8553 - val_loss: nan
Epoch 100/100
1579/1580 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8553 - loss: nanEpoch 100: Loss = nan, Accuracy = 0.8555
1580/1580 ━━━━━━━━━━━━━━━━━━━━ 29s 18ms/step - accuracy: 0.8553 - loss: nan - val_accuracy: 0.8553 - val_loss: nan
```

Empieza a programar o a crear código con IA.

Empieza a programar o a crear código con IA.

```
!pip install rouge-score
```

```
Collecting rouge-score
  Downloading rouge_score-0.1.2.tar.gz (17 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: absl-py in /usr/local/lib/python3.11/dist-packages (from rouge-score) (1.4.0)
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages (from rouge-score) (3.9.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from rouge-score) (2.0.2)
Requirement already satisfied: six>=1.14.0 in /usr/local/lib/python3.11/dist-packages (from rouge-score) (1.17.0)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from nltk->rouge-score) (8.1.8)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages (from nltk->rouge-score) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist-packages (from nltk->rouge-score) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from nltk->rouge-score) (4.67.1)
Building wheels for collected packages: rouge-score
  Building wheel for rouge-score (setup.py) ... done
  Created wheel for rouge-score: filename=rouge_score-0.1.2-py3-none-any.whl size=24935 sha256=15b46520492547674497169769547 59bccaf79b33
  Stored in directory: /root/.cache/pip/wheels/1e/19/43/8a442dc83660ca25e163e1bd1f89919284ab0d0c1475475148
Successfully built rouge-score
Installing collected packages: rouge-score
Successfully installed rouge-score-0.1.2
```

```
!pip install rouge
```

```
Collecting rouge
  Downloading rouge-1.0.1-py3-none-any.whl.metadata (4.1 kB)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages (from rouge) (1.17.0)
Downloading rouge-1.0.1-py3-none-any.whl (13 kB)
Installing collected packages: rouge
Successfully installed rouge-1.0.1
```

```
# Evaluar con BLEU y ROUGE
from nltk.translate.bleu_score import sentence_bleu
from rouge_score import rouge_scorer
from rouge import Rouge

def evaluate_model(predictions, references):
    bleu_scores = [sentence_bleu([ref.split()], pred.split()) for pred, ref in zip(predictions, references)]
```

```
    rouge = Rouge()
    rouge_scores = rouge.get_scores(predictions, references, avg=True)

    print("BLEU Score:", np.mean(bleu_scores))
    print("ROUGE Score:", rouge_scores)


# Ejemplo de evaluación
test_predictions = ["hello world", "good morning"]
test_references = ["hello world", "morning good"]
evaluate_model(test_predictions, test_references)
```

```
BLEU Score: 7.45834073120031e-155
ROUGE Score: {'rouge-1': {'r': 1.0, 'p': 1.0, 'f': 0.999999995}, 'rouge-2': {'r': 0.5, 'p': 0.5, 'f': 0.4999999975}, 'rouge-l': {'r': 0.
/usr/local/lib/python3.11/dist-packages/nltk/translate/bleu_score.py:577: UserWarning:
The hypothesis contains 0 counts of 3-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
/usr/local/lib/python3.11/dist-packages/nltk/translate/bleu_score.py:577: UserWarning:
The hypothesis contains 0 counts of 4-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
/usr/local/lib/python3.11/dist-packages/nltk/translate/bleu_score.py:577: UserWarning:
The hypothesis contains 0 counts of 2-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
```

Empieza a programar o a crear código con IA.