```
! pip install datasets jiwer

Requirement already satisfied: datasets in
/usr/local/lib/python3.11/dist-packages (2.14.4)
Requirement already satisfied: jiwer in
/usr/local/lib/python3.11/dist-packages (3.1.0)
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.11/dist-packages (from datasets) (2.0.2)
Requirement already satisfied: pyarrow>=8.0.0 in
/usr/local/lib/python3.11/dist-packages (from datasets) (18.1.0)
Requirement already satisfied: dill<0.3.8,>=0.3.0 in
/usr/local/lib/python3.11/dist-packages (from datasets) (0.3.7)
Requirement already satisfied: pandas in
/usr/local/lib/python3.11/dist-packages (from datasets) (2.2.2)
Requirement already satisfied: requests>=2.19.0 in
/usr/local/lib/python3.11/dist-packages (from datasets) (2.32.3)
Requirement already satisfied: tqdm>=4.62.1 in
/usr/local/lib/python3.11/dist-packages (from datasets) (4.67.1)
Requirement already satisfied: xxhash in
/usr/local/lib/python3.11/dist-packages (from datasets) (3.5.0)
Requirement already satisfied: multiprocess in
/usr/local/lib/python3.11/dist-packages (from datasets) (0.70.15)
Requirement already satisfied: fsspec>=2021.11.1 in
/usr/local/lib/python3.11/dist-packages (from fsspec[http]>=2021.11.1-
>datasets) (2025.3.2)
Requirement already satisfied: aiohttp in
/usr/local/lib/python3.11/dist-packages (from datasets) (3.11.15)
Requirement already satisfied: huggingface-hub<1.0.0,>=0.14.0 in
/usr/local/lib/python3.11/dist-packages (from datasets) (0.31.1)
Requirement already satisfied: packaging in
/usr/local/lib/python3.11/dist-packages (from datasets) (24.2)
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.11/dist-packages (from datasets) (6.0.2)
Requirement already satisfied: click>=8.1.8 in
/usr/local/lib/python3.11/dist-packages (from jiwer) (8.1.8)
Requirement already satisfied: rapidfuzz>=3.9.7 in
/usr/local/lib/python3.11/dist-packages (from jiwer) (3.13.0)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp->datasets)
(2.6.1)
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.11/dist-packages (from aiohttp->datasets)
(1.3.2)
Requirement already satisfied: attrs>=17.3.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp->datasets)
(25.3.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.11/dist-packages (from aiohttp->datasets)
(1.6.0)
Requirement already satisfied: multidict<7.0,>=4.5 in
```

```
/usr/local/lib/python3.11/dist-packages (from aiohttp->datasets)
(6.4.3)
Requirement already satisfied: propcache>=0.2.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp->datasets)
(0.3.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp->datasets)
(1.20.0)
Requirement already satisfied: filelock in
/usr/local/lib/python3.11/dist-packages (from huggingface-
hub<1.0.0,>=0.14.0->datasets) (3.18.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.11/dist-packages (from huggingface-
hub<1.0.0,>=0.14.0->datasets) (4.13.2)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.0 in
/usr/local/lib/python3.11/dist-packages (from huggingface-
hub<1.0.0,>=0.14.0->datasets) (1.1.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests>=2.19.0-
>datasets) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.11/dist-packages (from requests>=2.19.0-
>datasets) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests>=2.19.0-
>datasets) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests>=2.19.0-
>datasets) (2025.4.26)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas->datasets)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.11/dist-packages (from pandas->datasets)
(2025.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.11/dist-packages (from pandas->datasets)
(2025.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2-
>pandas->datasets) (1.17.0)
```

```
import os
import random
import string
import re
import numpy as np
import pandas as pd
import tensorflow as tf
```

```
from transformers import T5Tokenizer, TFT5ForConditionalGeneration
from tensorflow.keras.optimizers import AdamW
from tensorflow.keras.callbacks import EarlyStopping
from datasets import Dataset, DatasetDict
import nltk
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction
from jiwer import wer,cer

from transformers import (
    AutoTokenizer,
    AutoModelForSeq2SeqLM,
    DataCollatorForSeq2Seq,
    Seq2SeqTrainingArguments,
    Seq2SeqTrainer,
    T5Tokenizer,
    T5ForConditionalGeneration
)

try:
    nltk.data.find('tokenizers/punkt')
except LookupError:
    print("'punkt' not found. Downloading...")
    nltk.download('punkt', quiet=True)
    print("'punkt' downloaded.")

print(f"TensorFlow version: {tf.__version__}")
print(f"Num GPUs Available:
{len(tf.config.experimental.list_physical_devices('GPU'))}")
```

```
TensorFlow version: 2.18.0
Num GPUs Available: 1
```

```
FILES_TO_DOWNLOAD = {
    "tune": "tune.tsv",
    "validation": "validation.tsv",
    "test": "test.tsv"
}

def remove_punctuation_and_digits(text):
    chars_to_remove = string.punctuation + string.digits
    translator = str.maketrans('', '', chars_to_remove)
    return text.translate(translator)

def load_sentences_from_tsv(filepath):
    df = pd.read_csv(filepath, sep='\t', header=None, usecols=[0],
names=['sentence'])
    df['sentence'] = df['sentence'].apply(lambda x: x.lstrip("'
").strip())
    df['sentence'] =
df['sentence'].apply(remove_punctuation_and_digits)
```

```
    df['sentence'] = df['sentence'].apply(lambda x: "
".join(x.split()))

    return df['sentence'].tolist()

print("\nLoading sentences...")
raw_tune_sentences =
load_sentences_from_tsv(os.path.join("/content/data/",
FILES_TO_DOWNLOAD["tune"]))
raw_validation_sentences =
load_sentences_from_tsv(os.path.join("/content/data/",
FILES_TO_DOWNLOAD["validation"]))
raw_test_sentences =
load_sentences_from_tsv(os.path.join("/content/data/",
FILES_TO_DOWNLOAD["test"]))

print(f"Loaded {len(raw_tune_sentences)} tune sentences.")
print(f"Loaded {len(raw_validation_sentences)} validation sentences.")
print(f"Loaded {len(raw_test_sentences)} test sentences.")
print(f"Example tune sentence: {raw_tune_sentences[0]}")


Loading sentences...
Loaded 5000 tune sentences.
Loaded 5000 validation sentences.
Loaded 5000 test sentences.
Example tune sentence: was the second sequel to appear though Hooper
did not return to direct due to scheduling conflicts with another film
Spontaneous Combustion

def introduce_char_deletion(word):
    if len(word) <= 1:
        return word
    idx = random.randint(0, len(word) - 1)
    return word[:idx] + word[idx+1:]

def introduce_char_insertion(word):
    if not word:
        return random.choice(string.ascii_lowercase)
    idx = random.randint(0, len(word))
    char_to_insert = random.choice(string.ascii_lowercase)
    return word[:idx] + char_to_insert + word[idx:]

def introduce_char_substitution(word):
    if not word:
        return random.choice(string.ascii_lowercase)
    if len(word) == 0:
        return random.choice(string.ascii_lowercase)
    idx = random.randint(0, len(word) - 1)
    original_char = word[idx]
```

```
    new_char = random.choice(string.ascii_lowercase)
    if len(string.ascii_lowercase) > 1:
         while new_char == original_char.lower():
            new_char = random.choice(string.ascii_lowercase)
    return word[:idx] + new_char + word[idx+1:]

def introduce_char_transposition(word):
    if len(word) < 2:
        return word
    idx = random.randint(0, len(word) - 2)
    chars = list(word)
    chars[idx], chars[idx+1] = chars[idx+1], chars[idx]
    return "".join(chars)

error_functions = [
    introduce_char_deletion,
    introduce_char_insertion,
    introduce_char_substitution,
    introduce_char_transposition
]

def generate_misspelled_sentence_random_errors(sentence,
error_rate_word=0.2):
    words = sentence.split()
    misspelled_words = []

    for word in words:
        if random.random() < error_rate_word and len(word) > 2:
            temp_word = word
            num_errors_to_apply=1

            original_first_char_case = None
            if temp_word and temp_word[0].isupper():
                original_first_char_case = 'upper'
                temp_word_lower = temp_word[0].lower() +
temp_word[1:].lower() if len(temp_word) > 0 else ""
            else:
                temp_word_lower = temp_word.lower()


            for _ in range(num_errors_to_apply):
                if not temp_word_lower: break

                error_func = random.choice(error_functions)
                temp_word_lower = error_func(temp_word_lower)

            if original_first_char_case == 'upper' and
temp_word_lower:
                 temp_word = temp_word_lower[0].upper() +
temp_word_lower[1:]
```

```
            else:
                temp_word = temp_word_lower

            misspelled_words.append(temp_word)
        else:
            misspelled_words.append(word)

    return " ".join(misspelled_words)

print("\nOriginal:", raw_tune_sentences[0])
for i in range(5):
    print(f"Misspelled {i+1}:",
generate_misspelled_sentence_random_errors(raw_tune_sentences[0],
error_rate_word=0.5))
```

```
Original: was the second sequel to appear though Hooper did not return
to direct due to scheduling conflicts with another film Spontaneous
Combustion
Misspelled 1: ws the secnd sequel to appear thoguh Fhooper did not
retkrn to diretc due to schedulng conflicts with jnother fim
Spontaneous Combustion
Misspelled 2: wras the second sequel to appeaer though Hooper did not
weturn to direct ude to suheduling conlficts wih anothxer foilm
Spontaneous Combustiofn
Misspelled 3: was the seocnd sequel to appear thrugh Hooepr did noty
return to disect dse to scheduling conflicts with anothen filmm
Spontaneous Cobustion
Misspelled 4: was the secnod sequel to apeear though Hooer dwd nsot
retukrn to dipect due to scheduling conflictsw with another film
Spaontaneous Combstion
Misspelled 5: was the seconb seqeul to appear thugh Hoouper ddi not
erturn to direct duv to scheduling conflictsp with another film
Spontaneous Ombustion
```

```
MAX_SENTENCES_TUNE = None
MAX_SENTENCES_VAL_TEST = None
VERSIONS_PER_SENTENCE = 4

def create_paired_dataset(correct_sentences, num_versions=1,
subset_name="train", max_sentences=None):
    misspelled_list = []
    correct_list = []

    if max_sentences:
        correct_sentences = correct_sentences[:max_sentences]

    print(f"Generating misspelled data for {subset_name}
({len(correct_sentences)} sentences, {num_versions} versions
each)...")
```

```
    count = 0
    for sentence in correct_sentences:
        if not sentence.strip():
            continue
        for _ in range(num_versions):
            misspelled =
generate_misspelled_sentence_random_errors(sentence)
            if misspelled.strip() and misspelled != sentence:
                misspelled_list.append(misspelled)
                correct_list.append(sentence)
        count += 1
        if count % (len(correct_sentences)//10 if
len(correct_sentences) > 10 else 1) == 0:
             print(f"  Processed {count}/{len(correct_sentences)}
original sentences for {subset_name}...")

    return pd.DataFrame({"misspelled": misspelled_list, "correct":
correct_list})

train_df = create_paired_dataset(raw_tune_sentences,
num_versions=VERSIONS_PER_SENTENCE, subset_name="train",
max_sentences=MAX_SENTENCES_TUNE)
validation_df = create_paired_dataset(raw_validation_sentences,
num_versions=1, subset_name="validation",
max_sentences=MAX_SENTENCES_VAL_TEST)
test_df = create_paired_dataset(raw_test_sentences, num_versions=1,
subset_name="test", max_sentences=MAX_SENTENCES_VAL_TEST)

print(f"\nGenerated {len(train_df)} training pairs.")
print(f"Generated {len(validation_df)} validation pairs.")
print(f"Generated {len(test_df)} test pairs.")

if not train_df.empty:
    print("\nSample of generated training data:")
    print(train_df.head())
else:
    print("Warning: Training DataFrame is empty. Check error
generation or input data.")
    if MAX_SENTENCES_TUNE < 50 and VERSIONS_PER_SENTENCE ==1 :
        print("Consider increasing MAX_SENTENCES_TUNE or
VERSIONS_PER_SENTENCE.")
```

```
Generating misspelled data for train (5000 sentences, 4 versions
each)...
  Processed 500/5000 original sentences for train...
  Processed 1000/5000 original sentences for train...
  Processed 1500/5000 original sentences for train...
  Processed 2000/5000 original sentences for train...
  Processed 2500/5000 original sentences for train...
  Processed 3000/5000 original sentences for train...
```

```
  Processed 3500/5000 original sentences for train...
  Processed 4000/5000 original sentences for train...
  Processed 4500/5000 original sentences for train...
  Processed 5000/5000 original sentences for train...
Generating misspelled data for validation (5000 sentences, 1 versions
each)...
  Processed 500/5000 original sentences for validation...
  Processed 1000/5000 original sentences for validation...
  Processed 1500/5000 original sentences for validation...
  Processed 2000/5000 original sentences for validation...
  Processed 2500/5000 original sentences for validation...
  Processed 3000/5000 original sentences for validation...
  Processed 3500/5000 original sentences for validation...
  Processed 4000/5000 original sentences for validation...
  Processed 4500/5000 original sentences for validation...
  Processed 5000/5000 original sentences for validation...
Generating misspelled data for test (5000 sentences, 1 versions
each)...
  Processed 500/5000 original sentences for test...
  Processed 1000/5000 original sentences for test...
  Processed 1500/5000 original sentences for test...
  Processed 2000/5000 original sentences for test...
  Processed 2500/5000 original sentences for test...
  Processed 3000/5000 original sentences for test...
  Processed 3500/5000 original sentences for test...
  Processed 4000/5000 original sentences for test...
  Processed 4500/5000 original sentences for test...
  Processed 5000/5000 original sentences for test...

Generated 19681 training pairs.
Generated 4912 validation pairs.
Generated 4914 test pairs.

Sample of generated training data:
                                           misspelled  \
0  was the secoqnd sequel to appear though Hooptr...
1  was the second sequel to appear though Hooper ...
2  was the seocnd sequel to appear though Hooper ...
3  aws he second sqeuel to appear though Hooper d...
4  Maolain said to be diminutive of bald name of ...

                                              correct
0  was the second sequel to appear though Hooper ...
1  was the second sequel to appear though Hooper ...
2  was the second sequel to appear though Hooper ...
3  was the second sequel to appear though Hooper ...
4  Maolain said to be diminutive of bald name of ...

MODEL_NAME = 't5-small'
tokenizer = T5Tokenizer.from_pretrained(MODEL_NAME)
```

```
PREFIX = "fix spelling: "
MAX_INPUT_LENGTH = 128
MAX_TARGET_LENGTH = 128

def preprocess_function(examples):
    inputs = [PREFIX + misspelled for misspelled in
examples['misspelled']]
    targets = [correct for correct in examples['correct']]

    model_inputs = tokenizer(inputs, max_length=MAX_INPUT_LENGTH,
truncation=True, padding='max_length')

    labels = tokenizer(targets, max_length=MAX_TARGET_LENGTH,
truncation=True, padding='max_length')

    model_inputs['labels'] = labels['input_ids']

    for i in range(len(model_inputs['labels'])):
        model_inputs['labels'][i] = [
            (l if l != tokenizer.pad_token_id else -100) for l in
model_inputs['labels'][i]
        ]

    return model_inputs

if not train_df.empty and not validation_df.empty and not
test_df.empty :
    dataset_train_hf = Dataset.from_pandas(train_df)
    dataset_val_hf = Dataset.from_pandas(validation_df)
    dataset_test_hf = Dataset.from_pandas(test_df)

    raw_datasets = DatasetDict({
        'train': dataset_train_hf,
        'validation': dataset_val_hf,
        'test': dataset_test_hf
    })

    print("\nTokenizing datasets...")
    tokenized_datasets = raw_datasets.map(preprocess_function,
batched=True, remove_columns=['misspelled', 'correct'])
    print(tokenized_datasets)
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/
_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
```

```
Please note that authentication is recommended but still optional to
access public models or datasets.
  warnings.warn(
You are using the default legacy behaviour of the <class
'transformers.models.t5.tokenization_t5.T5Tokenizer'>. This is
expected, and simply means that the `legacy` (previous) behavior will
be used so nothing changes for you. If you want to use the new
behaviour, set `legacy=False`. This should only be set if you
understand what it means, and thoroughly read the reason why this was
added as explained in
https://github.com/huggingface/transformers/pull/24565


Tokenizing datasets...
```


```
{"model_id":"92f836677dac4765b0c40c01b1e43aaf","version_major":2,"vers
ion_minor":0}

{"model_id":"d238dd6fb33c48d3a260c34541062c5f","version_major":2,"vers
ion_minor":0}

{"model_id":"ccb0b5f1374a40f3b5d1069e830d4462","version_major":2,"vers
ion_minor":0}
```


```
DatasetDict({
    train: Dataset({
        features: ['input_ids', 'attention_mask', 'labels'],
        num_rows: 19681
    })
    validation: Dataset({
        features: ['input_ids', 'attention_mask', 'labels'],
        num_rows: 4912
    })
    test: Dataset({
        features: ['input_ids', 'attention_mask', 'labels'],
        num_rows: 4914
    })
})

BATCH_SIZE = 32
def to_tf_dataset(dataset_hf, batch_size, shuffle=False):
    columns = ['input_ids', 'attention_mask', 'labels']
    dataset_hf.set_format(type='tensorflow', columns=columns)

    features = {x: dataset_hf[x] for x in ['input_ids',
'attention_mask']}
    labels = dataset_hf['labels']

    tf_dataset = tf.data.Dataset.from_tensor_slices((features,
labels))
    if shuffle:
```

```
        tf_dataset = tf_dataset.shuffle(buffer_size=len(dataset_hf))
    tf_dataset = tf_dataset.batch(batch_size)
    return tf_dataset

tf_train_dataset = to_tf_dataset(tokenized_datasets['train'],
BATCH_SIZE, shuffle=True)
tf_validation_dataset =
to_tf_dataset(tokenized_datasets['validation'], BATCH_SIZE)
tf_test_dataset = to_tf_dataset(tokenized_datasets['test'],
BATCH_SIZE)

print("\nSample from tokenized training data (first batch):")
for batch in tf_train_dataset.take(1):
    inputs, labels = batch
    print("Input IDs shape:", inputs['input_ids'].shape)
    print("Attention Mask shape:", inputs['attention_mask'].shape)
    print("Labels shape:", labels.shape)
    print("Decoded Input Sample:",
tokenizer.decode(inputs['input_ids'][0], skip_special_tokens=False))
    print("Decoded Label Sample:", tokenizer.decode([l if l != -100
else tokenizer.pad_token_id for l in labels[0]],
skip_special_tokens=False))
    break
else:
    print("Skipping tokenization and model training due to empty
dataframes.")
    tf_train_dataset, tf_validation_dataset, tf_test_dataset = None,
None, None


Sample from tokenized training data (first batch):
Input IDs shape: (32, 128)
Attention Mask shape: (32, 128)
Labels shape: (32, 128)
Decoded Input Sample: fix spelling: Iceladic sheepdogs are very alert
and aill always give visctors an enthusiasticw welcome without ebing
aggressive</s><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad>
Decoded Label Sample: Icelandic sheepdogs are very alert and will
always give visitors an enthusiastic welcome without being
aggressive</s><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
```

```
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad>
```

T5 Model

```
from tf_keras.optimizers import AdamW as TfKerasAdamW


print("\nInitializing T5 model...")
model = TFT5ForConditionalGeneration.from_pretrained(MODEL_NAME)

LEARNING_RATE = 3e-5

print("Using optimizer from tf_keras.optimizers.AdamW")
optimizer_instance =
TfKerasAdamW(learning_rate=LEARNING_RATE,weight_decay=0.01)
print(f"Optimizer instance created: {optimizer_instance}")

model.compile(optimizer=optimizer_instance)
print("Model compiled successfully with tf_keras.optimizers.AdamW!")


Initializing T5 model...

{"model_id":"3481f2327a484386911499ab1b7e1c4d","version_major":2,"vers
ion_minor":0}

{"model_id":"20c771a7e4ec4d7ea409502c0d509886","version_major":2,"vers
ion_minor":0}

All PyTorch model weights were used when initializing
TFT5ForConditionalGeneration.

All the weights of TFT5ForConditionalGeneration were initialized from
the PyTorch model.
If your task is similar to the task the model of the checkpoint was
trained on, you can already use TFT5ForConditionalGeneration for
predictions without further training.

Using optimizer from tf_keras.optimizers.AdamW
Optimizer instance created: <tf_keras.src.optimizers.adamw.AdamW
object at 0x7c34d3fca990>
Model compiled successfully with tf_keras.optimizers.AdamW!

from tf_keras.callbacks import EarlyStopping

early_stopping_callback = EarlyStopping(
      monitor='val_loss',
      patience=2,
```

```
      restore_best_weights=True,
      verbose=1
)

NUM_EPOCHS = 3
print("\nStarting fine-tuning...")

history = model.fit(
        tf_train_dataset,
        validation_data=tf_validation_dataset,
        epochs=NUM_EPOCHS,
        callbacks=[early_stopping_callback]
        )
```

```
Starting fine-tuning...
Epoch 1/3
616/616 [==============================] - 435s 640ms/step - loss:
0.8387 - val_loss: 0.5588
Epoch 2/3
616/616 [==============================] - 391s 635ms/step - loss:
0.6347 - val_loss: 0.4916
Epoch 3/3
616/616 [==============================] - 391s 635ms/step - loss:
0.5734 - val_loss: 0.4525
Restoring model weights from the end of the best epoch: 3.
```

```
import numpy as np
import tensorflow as tf
from jiwer import wer, cer

def compute_metrics(model, dataset, tokenizer, prefix="fix spelling:
"):
    total_batches = tf.data.experimental.cardinality(dataset).numpy()
    if total_batches < 0:
        total_batches = None

    all_refs = []
    all_hyps = []

    for batch_idx, batch in enumerate(dataset):
        features, labels = batch

        generated_ids = model.generate(
            input_ids=features["input_ids"],
            attention_mask=features["attention_mask"],
            max_length=tokenizer.model_max_length,
        )

        hyps = tokenizer.batch_decode(generated_ids,
```

```
skip_special_tokens=True)

        label_ids = np.where(labels.numpy() == -100,
                             tokenizer.pad_token_id,
                             labels.numpy())
        refs = tokenizer.batch_decode(label_ids,
skip_special_tokens=True)

        all_hyps.extend(hyps)
        all_refs.extend(refs)

        if total_batches:
            print(f"Batch {batch_idx+1}/{total_batches} completed")
        else:
            print(f"Batch {batch_idx+1} completed")

    overall_wer = wer(all_refs, all_hyps)
    overall_cer = cer(all_refs, all_hyps)
    return {
        "wer": overall_wer,
        "cer": overall_cer
        }

metrics = compute_metrics(model, tf_test_dataset, tokenizer)
print(f"\nFinal results → WER: {metrics['wer']:.3%}, CER:
{metrics['cer']:.3%}, Exact-Match Accuracy:
{metrics['exact_match_accuracy']:.3%}")
```

```
Batch 1/153 completed
Batch 2/153 completed
Batch 3/153 completed
Batch 4/153 completed
Batch 5/153 completed
Batch 6/153 completed
Batch 7/153 completed
Batch 8/153 completed
Batch 9/153 completed
Batch 10/153 completed
Batch 11/153 completed
Batch 12/153 completed
Batch 13/153 completed
Batch 14/153 completed
Batch 15/153 completed
Batch 16/153 completed
Batch 17/153 completed
Batch 18/153 completed
Batch 19/153 completed
Batch 20/153 completed
Batch 21/153 completed
Batch 22/153 completed
```

```
Batch 23/153 completed
Batch 24/153 completed
Batch 25/153 completed
Batch 26/153 completed
Batch 27/153 completed
Batch 28/153 completed
Batch 29/153 completed
Batch 30/153 completed
Batch 31/153 completed
Batch 32/153 completed
Batch 33/153 completed
Batch 34/153 completed
Batch 35/153 completed
Batch 36/153 completed
Batch 37/153 completed
Batch 38/153 completed
Batch 39/153 completed
Batch 40/153 completed
Batch 41/153 completed
Batch 42/153 completed
Batch 43/153 completed
Batch 44/153 completed
Batch 45/153 completed
Batch 46/153 completed
Batch 47/153 completed
Batch 48/153 completed
Batch 49/153 completed
Batch 50/153 completed
Batch 51/153 completed
Batch 52/153 completed
Batch 53/153 completed
Batch 54/153 completed
Batch 55/153 completed
Batch 56/153 completed
Batch 57/153 completed
Batch 58/153 completed
Batch 59/153 completed
Batch 60/153 completed
Batch 61/153 completed
Batch 62/153 completed
Batch 63/153 completed
Batch 64/153 completed
Batch 65/153 completed
Batch 66/153 completed
Batch 67/153 completed
Batch 68/153 completed
Batch 69/153 completed
Batch 70/153 completed
Batch 71/153 completed
```

```
Batch 72/153 completed
Batch 73/153 completed
Batch 74/153 completed
Batch 75/153 completed
Batch 76/153 completed
Batch 77/153 completed
Batch 78/153 completed
Batch 79/153 completed
Batch 80/153 completed
Batch 81/153 completed
Batch 82/153 completed
Batch 83/153 completed
Batch 84/153 completed
Batch 85/153 completed
Batch 86/153 completed
Batch 87/153 completed
Batch 88/153 completed
Batch 89/153 completed
Batch 90/153 completed
Batch 91/153 completed
Batch 92/153 completed
Batch 93/153 completed
Batch 94/153 completed
Batch 95/153 completed
Batch 96/153 completed
Batch 97/153 completed
Batch 98/153 completed
Batch 99/153 completed
Batch 100/153 completed
Batch 101/153 completed
Batch 102/153 completed
Batch 103/153 completed
Batch 104/153 completed
Batch 105/153 completed
Batch 106/153 completed
Batch 107/153 completed
Batch 108/153 completed
Batch 109/153 completed
Batch 110/153 completed
Batch 111/153 completed
Batch 112/153 completed
Batch 113/153 completed
Batch 114/153 completed
Batch 115/153 completed
Batch 116/153 completed
Batch 117/153 completed
Batch 118/153 completed
Batch 119/153 completed
Batch 120/153 completed
```

```
Batch 121/153 completed
Batch 122/153 completed
Batch 123/153 completed
Batch 124/153 completed
Batch 125/153 completed
Batch 126/153 completed
Batch 127/153 completed
Batch 128/153 completed
Batch 129/153 completed
Batch 130/153 completed
Batch 131/153 completed
Batch 132/153 completed
Batch 133/153 completed
Batch 134/153 completed
Batch 135/153 completed
Batch 136/153 completed
Batch 137/153 completed
Batch 138/153 completed
Batch 139/153 completed
Batch 140/153 completed
Batch 141/153 completed
Batch 142/153 completed
Batch 143/153 completed
Batch 144/153 completed
Batch 145/153 completed
Batch 146/153 completed
Batch 147/153 completed
Batch 148/153 completed
Batch 149/153 completed
Batch 150/153 completed
Batch 151/153 completed
Batch 152/153 completed
Batch 153/153 completed

Final results → WER: 11.524%, CER: 5.248%, Exact-Match Accuracy:
5.640%
```

```python
SAVE_DIRECTORY = "./my_spell_corrector_t5_small"
if not os.path.exists(SAVE_DIRECTORY):
    os.makedirs(SAVE_DIRECTORY)
    print(f"Created directory: {SAVE_DIRECTORY}")

print(f"\nSaving model to {SAVE_DIRECTORY}...")
model.save_pretrained(SAVE_DIRECTORY)
print("Model weights and config saved.")

print(f"Saving tokenizer to {SAVE_DIRECTORY}...")
tokenizer.save_pretrained(SAVE_DIRECTORY)
print("Tokenizer saved.")
```

```
Created directory: ./my_spell_corrector_t5_small

Saving model to ./my_spell_corrector_t5_small...
Model weights and config saved.
Saving tokenizer to ./my_spell_corrector_t5_small...
Tokenizer saved.
```

```
def
correct_sentences_in_batch(misspelled_sentences_list,model,tokenizer,p
refix,max_input_length,max_target_length,num_beams=4,early_stopping=Tr
ue):
    if not misspelled_sentences_list:
        return []

    prefixed_sentences = [prefix + sentence for sentence in
misspelled_sentences_list]
    inputs = tokenizer(
        prefixed_sentences,
        return_tensors="tf",
        max_length=max_input_length,
        truncation=True,
        padding="longest"
    )
    summary_ids = model.generate(
        inputs['input_ids'],
        attention_mask=inputs['attention_mask'],
        max_length=max_target_length,
        num_beams=num_beams,
        early_stopping=early_stopping
    )
    corrected_sentences = tokenizer.batch_decode(summary_ids,
skip_special_tokens=True)
    return corrected_sentences


print("\n--- Example Inference with Function ---")

custom_sentences_misspelled = [
    "I have a qestion abot ths assignent",
    "teh qwik brwn fox jmps ovr teh lazy dog.",
    "he dont know nuthin abot programing.",
    "ths is anothr exmple to tst."
]

corrected_batch =
correct_sentences_in_batch(custom_sentences_misspelled,model,tokenizer
,PREFIX,MAX_INPUT_LENGTH,MAX_TARGET_LENGTH)
for original, corrected in zip(custom_sentences_misspelled,
corrected_batch):
    print(f"Input Misspelled: {original}")
```

```
    print(f"Corrected Output: {corrected}")
    print("---")


--- Example Inference with Function ---
Input Misspelled: I have a qestion abot ths assignent
Corrected Output: I have a qestion abot the assigned
---
Input Misspelled: teh qwik brwn fox jmps ovr teh lazy dog.
Corrected Output: teh qwik brwn fox jmps and the lazy dog.
---
Input Misspelled: he dont know nuthin abot programing.
Corrected Output: he dont know nuthin abot programing.
---
Input Misspelled: ths is anothr exmple to tst.
Corrected Output: ths is an extension exmple to th.
---

sample_df = test_df.sample(5, random_state=42)
misspelled_examples = sample_df['misspelled'].tolist()
ground_truths       = sample_df['correct'].tolist()

predictions = correct_sentences_in_batch(
    misspelled_examples,
    model,
    tokenizer,
    PREFIX,
    MAX_INPUT_LENGTH,
    MAX_TARGET_LENGTH,
    num_beams=4,
    early_stopping=True
)

for i, (inp, pred, true) in enumerate(zip(misspelled_examples,
predictions, ground_truths), 1):
    print(f"Example {i}")
    print(f" Misspelled: {inp}")
    print(f" Predicted : {pred}")
    print(f" Ground-truth: {true}")
    print("-" * 50)

Example 1
 Misspelled: Joseph Aton October was an Englioh journalist drmatist
and miscellaneous writer born in the son of Wialiam Aston gunsmith of
Deansgate in Manchester
 Predicted : Joseph Aton October was an English journalist drmatist
and miscellaneous writer born in the son of Wialiam Aston gunsmith of
Deansgate in Manchester
 Ground-truth: Joseph Aston October was an English journalist
dramatist and miscellaneous writer born in the son of William Aston
```

```
gunsmith of Deansgate in Manchester
--------------------------------------------------
Example 2
 Misspelled: Teh system continued to drift westwards and strengthened
rapidly that on midnight taat day the JMA furhter pugraded the system
into a Tropical Storm naming it Nock Teny
 Predicted : The system continued to drift westwards and strengthened
rapidly that on midnight taat day the JMA moved the system into a
Tropical Storm naming it Nock Teny
 Ground-truth: The system continued to drift westwards and
strengthened rapidly that on midnight that day the JMA furhter
upgraded the system into a Tropical Storm naming it Nock Ten
--------------------------------------------------
Example 3
 Misspelled: Carnaval music is often a song written especially for the
occasion and is easpy to dance to
 Predicted : Carnaval music is often a song written especially for the
occasion and is easy to dance to
 Ground-truth: Carnaval music is often a song written especially for
the occasion and is easy to dance to
--------------------------------------------------
Example 4
 Misspelled: The company was fvunded in by James Greaves and George
Cotton anb yincorporated in as a private limited company
 Predicted : The company was founded in by James Greaves and George
Cotton and incorporated in as a private limited company
 Ground-truth: The company was founded in by James Greaves and George
Cotton and incorporated in as a private limited company
--------------------------------------------------
Example 5
 Misspelled: Leukocytes in the vicinity of damaged tissue are
attracted to it by thb ccapture process and bniefly adhere to the
vendular endothelium inner cellular lining of veins
 Predicted : Leukocytes in the vicinity of damaged tissue are
attracted to it by the ccapture process and soon adhere to the
vendular endothelium inner cellular lining of veins
 Ground-truth: Leukocytes in the vicinity of damaged tissue are
attracted to it by the capture process and briefly adhere to the
venular endothelium inner cellular lining of veins
--------------------------------------------------
```

BART

```python
from datasets import Dataset, DatasetDict
from transformers import BartTokenizerFast

MODEL_NAME = 'facebook/bart-base'
tokenizer = BartTokenizerFast.from_pretrained(MODEL_NAME)
MAX_INPUT_LENGTH = 128
```

```
MAX_TARGET_LENGTH = 128

def preprocess_function(examples):
    inputs = examples['misspelled']
    targets = [correct for correct in examples['correct']]
    model_inputs = tokenizer(inputs,
                             max_length=MAX_INPUT_LENGTH,
                             truncation=True,
                             padding='max_length')
    labels = tokenizer(text_target=targets,
                       max_length=MAX_TARGET_LENGTH,
                       truncation=True,
                       padding='max_length')
    model_inputs['labels'] = labels['input_ids']
    for i in range(len(model_inputs['labels'])):
        model_inputs['labels'][i] = [
            (label_id if label_id != tokenizer.pad_token_id else -100)
            for label_id in model_inputs['labels'][i]
        ]
    return model_inputs

dataset_train_hf = Dataset.from_pandas(train_df)
dataset_val_hf = Dataset.from_pandas(validation_df)
dataset_test_hf = Dataset.from_pandas(test_df)
raw_datasets = DatasetDict({
        'train': dataset_train_hf,
        'validation': dataset_val_hf,
        'test': dataset_test_hf
    })

tokenized_datasets = raw_datasets.map(
        preprocess_function,
        batched=True,
        remove_columns=['misspelled', 'correct']
    )
print(tokenized_datasets)
print("\nExample of tokenized train data (first item):")
example = tokenized_datasets['train'][0]
print("Input IDs:", example['input_ids'])
print("Decoded Input:", tokenizer.decode(example['input_ids'],
skip_special_tokens=False))
print("Labels:", example['labels'])

decoded_labels_ids = [l if l != -100 else tokenizer.pad_token_id for l
in example['labels']]
print("Decoded Labels:", tokenizer.decode(decoded_labels_ids,
skip_special_tokens=False))
print("Attention Mask:", example['attention_mask'])
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/
_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
access public models or datasets.
  warnings.warn(
```


```
{"model_id":"10064516392f47598f13fd2e19d5638f","version_major":2,"vers
ion_minor":0}

{"model_id":"a9fc88b942a04106a2ff90d98125076c","version_major":2,"vers
ion_minor":0}

{"model_id":"59295bdd5b0440b8a8b6f3fe5da647ac","version_major":2,"vers
ion_minor":0}

{"model_id":"4e0bdaa622a74364b13ebd86478c1608","version_major":2,"vers
ion_minor":0}
```


```
Tokenizing datasets for BART...
```


```
{"model_id":"d46df64999104e80918ce460fe2638e8","version_major":2,"vers
ion_minor":0}

{"model_id":"cafc048b392242d5a1304c0a64c07a3b","version_major":2,"vers
ion_minor":0}

{"model_id":"79505c17e3d5435d83235f331b40166d","version_major":2,"vers
ion_minor":0}
```


```
DatasetDict({
    train: Dataset({
        features: ['input_ids', 'attention_mask', 'labels'],
        num_rows: 19666
    })
    validation: Dataset({
        features: ['input_ids', 'attention_mask', 'labels'],
        num_rows: 4923
    })
    test: Dataset({
        features: ['input_ids', 'attention_mask', 'labels'],
        num_rows: 4924
    })
})

Example of tokenized train data (first item):
```

```
Input IDs: [0, 7325, 3055, 298, 15636, 261, 10398, 12123, 7, 2082,
41090, 4147, 5082, 8428, 222, 45, 671, 7, 2228, 528, 7, 19114, 9549,
19, 277, 822, 2064, 2533, 1728, 6998, 29, 27166, 4193, 1499, 2, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Decoded Input: <s>was teh seconpd sequel to appear thogh Hooper did
not return to direct due to scheduling conflicts with another film
Spontaneouds
Combustion</s><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad>
Labels: [0, 7325, 5, 200, 12123, 7, 2082, 600, 5082, 8428, 222, 45,
671, 7, 2228, 528, 7, 19114, 9549, 19, 277, 822, 2064, 2533, 33101,
27166, 4193, 1499, 2, -100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100]
Decoded Labels: <s>was the second sequel to appear though Hooper did
not return to direct due to scheduling conflicts with another film
Spontaneous
Combustion</s><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad>
Attention Mask: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

BATCH_SIZE = 32
def to_tf_dataset(dataset_hf, batch_size, shuffle=False):
        columns = ['input_ids', 'attention_mask', 'labels']
        dataset_hf.set_format(type='tensorflow', columns=columns)
```

```
        features = {x: dataset_hf[x] for x in ['input_ids',
'attention_mask']}
        labels = dataset_hf['labels']

        tf_dataset = tf.data.Dataset.from_tensor_slices((features,
labels))
        if shuffle:
            tf_dataset =
tf_dataset.shuffle(buffer_size=len(dataset_hf))
        tf_dataset = tf_dataset.batch(batch_size)
        return tf_dataset

tf_train_dataset = to_tf_dataset(tokenized_datasets['train'],
BATCH_SIZE, shuffle=True)
tf_validation_dataset =
to_tf_dataset(tokenized_datasets['validation'], BATCH_SIZE)
tf_test_dataset = to_tf_dataset(tokenized_datasets['test'],
BATCH_SIZE)

for batch in tf_train_dataset.take(1):
        inputs, labels = batch
        print("Input IDs shape:", inputs['input_ids'].shape)
        print("Attention Mask shape:", inputs['attention_mask'].shape)
        print("Labels shape:", labels.shape)
        print("Decoded Input Sample:",
tokenizer.decode(inputs['input_ids'][0], skip_special_tokens=False))
        print("Decoded Label Sample:", tokenizer.decode([l if l != -
100 else tokenizer.pad_token_id for l in labels[0]],
skip_special_tokens=False))
        break
```

```
Sample from tokenized training data (first batch):
Input IDs shape: (32, 128)
Attention Mask shape: (32, 128)
Labels shape: (32, 128)
Decoded Input Sample: <s>Bad Cop tracks downd Emmet Wyldstyle nad
Virtruvius who rae rescued by Wyldstyle s boyfriend Batman who takes
thme to a meeting of the remaining Master
Builders</s><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pa
d><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pa
d><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pa
d><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pa
d><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pa
d><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pa
d><pad><pad><pad><pad>
Decoded Label Sample: <s>Bad Cop tracks down Emmet Wyldstyle and
Virtruvius who are rescued by Wyldstyle s boyfriend Batman who takes
them to a meeting of the remaining Master
```

```
Builders</s><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pa
d><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pa
d><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pa
d><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pa
d><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pa
d><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pa
d><pad><pad><pad><pad><pad><pad><pad><pad>
```

```
from transformers import TFBartForConditionalGeneration
from tf_keras.optimizers import AdamW as TfKerasAdamW
from tf_keras.optimizers import AdamW as KerasAdamW

model = TFBartForConditionalGeneration.from_pretrained(MODEL_NAME)
LEARNING_RATE = 3e-5
WEIGHT_DECAY = 0.01
optimizer_instance = KerasAdamW(learning_rate=LEARNING_RATE,
weight_decay=WEIGHT_DECAY)
print(f"Optimizer instance created: {optimizer_instance}")
model.compile(optimizer=optimizer_instance)
```

```
{"model_id":"5e5ca1ad531f41b3a7b99c8da94eb101","version_major":2,"vers
ion_minor":0}
```

```
All PyTorch model weights were used when initializing
TFBartForConditionalGeneration.

All the weights of TFBartForConditionalGeneration were initialized
from the PyTorch model.
If your task is similar to the task the model of the checkpoint was
trained on, you can already use TFBartForConditionalGeneration for
predictions without further training.

Optimizer instance created: <tf_keras.src.optimizers.adamw.AdamW
object at 0x792309067b50>
```

```
NUM_EPOCHS = 2
history = model.fit(
        tf_train_dataset,
        validation_data=tf_validation_dataset,
        epochs=NUM_EPOCHS
        )
```

```
Epoch 1/2
615/615 [==============================] - 189s 194ms/step - loss:
0.4705 - val_loss: 0.2849
Epoch 2/2
615/615 [==============================] - 112s 182ms/step - loss:
0.2780 - val_loss: 0.2531
```

```
def compute_metrics(model, dataset: tf.data.Dataset, tokenizer):
    all_refs = []
```

```
    all_hyps = []
    iiii=0
    for batch in dataset:
        features, labels = batch
        generated_ids = model.generate(
            input_ids=features["input_ids"],
            attention_mask=features["attention_mask"],
            max_length=MAX_TARGET_LENGTH,
            num_beams=4,
            early_stopping=True
        )
        print(f"batch: {iiii}")
        iiii+=1
        hyps = tokenizer.batch_decode(generated_ids.numpy(),
skip_special_tokens=True)
        label_ids_np = labels.numpy()
        label_ids_for_decode = np.where(label_ids_np == -100,
tokenizer.pad_token_id, label_ids_np)
        refs = tokenizer.batch_decode(label_ids_for_decode,
skip_special_tokens=True)
        all_hyps.extend(hyps)
        all_refs.extend(refs)
        if iiii == 10:
          break
    overall_wer = wer(all_refs, all_hyps)
    overall_cer = cer(all_refs, all_hyps)
    return {
        "wer": overall_wer,
        "cer": overall_cer
        }

metrics = compute_metrics(model, tf_test_dataset, tokenizer)
print(f"\nTest Results → WER: {metrics['wer']:.3%}, CER:
{metrics['cer']:.3%}")
```

```
batch: 0
batch: 1
batch: 2
batch: 3
batch: 4
batch: 5
batch: 6
batch: 7
batch: 8
batch: 9

Test Results → WER: 6.288%, CER: 3.572%
```

```
def predict_spelling_single_string(model, tokenizer, raw_text, prefix,
max_input_length, max_target_length):
```

```
    tokenized_input = tokenizer(
        [raw_text],
        max_length=max_input_length,
        truncation=True,
        padding='max_length',
        return_tensors='tf'
    )
    generated_ids = model.generate(
        input_ids=tokenized_input["input_ids"],
        attention_mask=tokenized_input["attention_mask"],
        max_length=max_target_length,
        num_beams=4,
        early_stopping=True
    )
    prediction = tokenizer.batch_decode(generated_ids.numpy(),
skip_special_tokens=True)[0]
    return prediction

custom_sentences = [
    "Whre are you giong",
    "Teh meat will b tasty",
    "Harry where re you living"
]
print("--- Testing with Custom Sentences ---")
for sentence in custom_sentences:
    prediction = predict_spelling_single_string(
        model, tokenizer, sentence, PREFIX, MAX_INPUT_LENGTH,
MAX_TARGET_LENGTH
    )
    print(f"Input:     {sentence}")
    print(f"Predicted: {prediction}")
    print("-" * 30)
```

```
--- Testing with Custom Sentences ---
Input:     Whre are you giong
Predicted: Where are you going
------------------------------
Input:     Teh meat will b tasty
Predicted: The meat will be tasty
------------------------------
Input:     Harry where re you living
Predicted: Harry where are you living
------------------------------
```

```
def predict_spelling_single_string(model, tokenizer, raw_text, prefix,
max_input_length, max_target_length):
    input_text = raw_text
    tokenized_input = tokenizer(
        [input_text],
        max_length=max_input_length,
```

```
        truncation=True,
        padding='max_length',
        return_tensors='tf'
    )
    generated_ids = model.generate(
        input_ids=tokenized_input["input_ids"],
        attention_mask=tokenized_input["attention_mask"],
        max_length=max_target_length,
        num_beams=4,
        early_stopping=True
    )
    prediction = tokenizer.batch_decode(generated_ids.numpy(),
skip_special_tokens=True)[0]
    return prediction

indices_to_test = [0, 1, 5, 65, 50]
for i in indices_to_test:
    misspelled_input = test_df.loc[i, 'misspelled']
    correct_reference = test_df.loc[i, 'correct']
    prediction = predict_spelling_single_string(
        model, tokenizer, misspelled_input, PREFIX, MAX_INPUT_LENGTH,
MAX_TARGET_LENGTH
    )
    print(f"Index {i}:")
    print(f"Input (Misspelled): {misspelled_input}")
    print(f"Reference (Correct):{correct_reference}")
    print(f"Predicted:          {prediction}")
    print("-" * 40)
```

```
Index 0:
Input (Misspelled): Ubandolier Budgie a free itupes app for iPad
iPhone and iPod touch released in December tellb thj story of the
making of Bandolier in the band s own words including an extensive
audio interview with Buzke Shelley
Reference (Correct):Bandolier Budgie a free iTunes app for iPad iPhone
and iPod touch released in December tells the story of the making of
Bandolier in the band s own words including an extensive audio
interview with Burke Shelley
Predicted:          Bandolier Budgie a free downloads app for iPad
iPhone and iPod touch released in December tell the story of the
making of Bandolier in the band s own words including an extensive
audio interview with Bruce Shelley
----------------------------------------
Index 1:
Input (Misspelled): Eden Black was grown fbom seed in the lat s by
Stephen Morley under his conditions it produces pitchers that are
almost completley black
Reference (Correct):Eden Black was grown from seed in the late s by
Stephen Morley under his conditions it produces pitchers that are
almost completley black
```

```
Predicted:          Eden Black was grown from seed in the late s by
Stephen Morley under his conditions it produces pitchers that are
almost completely black
----------------------------------------
Index 5:
Input (Misspelled): Aeqodynamic is an instrumental song by Daflt Punk
that is particularly well known for its robotic guitar solo
Reference (Correct):Aerodynamic is an instrumental song by Daft Punk
that is particularly well known for its robotic guitar solo
Predicted:          Aeqodynamic is an instrumental song by Daft Punk
that is particularly well known for its robotic guitar solo
----------------------------------------
Index 65:
Input (Misspelled): A crook is also a slang term or a criminal or a
person of questionable morality hte adjective crooked can refer to
such persons or actionsq
Reference (Correct):A crook is also a slang term for a criminal or a
person of questionable morality the adjective crooked can refer to
such persons or actions
Predicted:          A crook is also a slang term for a criminal or a
person of questionable morality the adjective crooked can refer to
such persons or actions
----------------------------------------
Index 50:
Input (Misspelled): A estimate by tjhe International Organization for
Mgiration suggests that between and Suanese re ilving in London whilst
a fairly vague estimate of to ahs been placed in Briuhton
Reference (Correct):A estimate by the International Organization for
Migration suggests that between and Sudanese are living in London
whilst a fairly vague estimate of to has been placed in Brighton
Predicted:          A estimate by the International Organization for
Migration suggests that between and Suanese are living in London
whilst a fairly vague estimate of to has been placed in Briethton
----------------------------------------
```

Fine Tuned BART

```python
MODEL_CHECKPOINT = "oliverguhr/spelling-correction-english-base"
tokenizer = AutoTokenizer.from_pretrained(MODEL_CHECKPOINT)
MAX_INPUT_LENGTH = 128
MAX_TARGET_LENGTH = 128

def preprocess_function(examples):
    inputs = examples['misspelled']
    targets = [correct for correct in examples['correct']]
    model_inputs = tokenizer(inputs,
                             max_length=MAX_INPUT_LENGTH,
                             truncation=True,
                             padding='max_length')
```

```
    labels = tokenizer(text_target=targets,
                       max_length=MAX_TARGET_LENGTH,
                       truncation=True,
                       padding='max_length')
    model_inputs['labels'] = labels['input_ids']
    for i in range(len(model_inputs['labels'])):
        model_inputs['labels'][i] = [
            (label_id if label_id != tokenizer.pad_token_id else -100)
            for label_id in model_inputs['labels'][i]
        ]
    return model_inputs


dataset_train_hf = Dataset.from_pandas(train_df)
dataset_val_hf = Dataset.from_pandas(validation_df)
dataset_test_hf = Dataset.from_pandas(test_df)
raw_datasets = DatasetDict({
        'train': dataset_train_hf,
        'validation': dataset_val_hf,
        'test': dataset_test_hf
    })

tokenized_datasets = raw_datasets.map(
        preprocess_function,
        batched=True,
        remove_columns=['misspelled', 'correct']
    )
print(tokenized_datasets)
print("\nExample of tokenized train data (first item):")
example = tokenized_datasets['train'][0]
print("Input IDs:", example['input_ids'])
print("Decoded Input:", tokenizer.decode(example['input_ids'],
skip_special_tokens=False))
print("Labels:", example['labels'])

decoded_labels_ids = [l if l != -100 else tokenizer.pad_token_id for l
in example['labels']]
print("Decoded Labels:", tokenizer.decode(decoded_labels_ids,
skip_special_tokens=False))
print("Attention Mask:", example['attention_mask'])

{"model_id":"f6d0a5ebbdda4f7db9ec8606b2731fc9","version_major":2,"vers
ion_minor":0}

{"model_id":"dd7a90785874491c805c90b217116196","version_major":2,"vers
ion_minor":0}

{"model_id":"26b00df4faf94994ac7398af70f2be3f","version_major":2,"vers
ion_minor":0}
```

{"model_id":"ae40be4ec285417dab3b62a32acf1456","version_major":2,"version_minor":0}

{"model_id":"e2bf32c478374220ba88e390bc7717cb","version_major":2,"version_minor":0}

{"model_id":"d953779f1fd446ebb4b2d932c946bc4b","version_major":2,"version_minor":0}

{"model_id":"639b6d7ba6f848f1835f3dab8de406d2","version_major":2,"version_minor":0}

{"model_id":"38e44fd689e2427e820d548c059825cf","version_major":2,"version_minor":0}

```
DatasetDict({
    train: Dataset({
        features: ['input_ids', 'attention_mask', 'labels'],
        num_rows: 19666
    })
    validation: Dataset({
        features: ['input_ids', 'attention_mask', 'labels'],
        num_rows: 4923
    })
    test: Dataset({
        features: ['input_ids', 'attention_mask', 'labels'],
        num_rows: 4924
    })
})

Example of tokenized train data (first item):
Input IDs: [0, 7325, 3055, 298, 15636, 261, 10398, 12123, 7, 2082,
41090, 4147, 5082, 8428, 222, 45, 671, 7, 2228, 528, 7, 19114, 9549,
19, 277, 822, 2064, 2533, 1728, 6998, 29, 27166, 4193, 1499, 2, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Decoded Input: <s>was teh seconpd sequel to appear thogh Hooper did
not return to direct due to scheduling conflicts with another film
Spontaneouds
Combustion</s><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad>
Labels: [0, 7325, 5, 200, 12123, 7, 2082, 600, 5082, 8428, 222, 45,
671, 7, 2228, 528, 7, 19114, 9549, 19, 277, 822, 2064, 2533, 33101,
```

```
27166, 4193, 1499, 2, -100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100]
Decoded Labels: <s>was the second sequel to appear though Hooper did
not return to direct due to scheduling conflicts with another film
Spontaneous
Combustion</s><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><
pad><pad><pad><pad>
Attention Mask: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
MODEL_CHECKPOINT
```


```
{"type":"string"}
```


```python
model =
TFBartForConditionalGeneration.from_pretrained(MODEL_CHECKPOINT)
LEARNING_RATE = 3e-5
WEIGHT_DECAY = 0.01
optimizer_instance = KerasAdamW(learning_rate=LEARNING_RATE,
weight_decay=WEIGHT_DECAY)
print(f"Optimizer instance created: {optimizer_instance}")
model.compile(optimizer=optimizer_instance)
```

```
All PyTorch model weights were used when initializing
TFBartForConditionalGeneration.

All the weights of TFBartForConditionalGeneration were initialized
from the PyTorch model.
If your task is similar to the task the model of the checkpoint was
trained on, you can already use TFBartForConditionalGeneration for
predictions without further training.
```

```
Optimizer instance created: <tf_keras.src.optimizers.adamw.AdamW
object at 0x79202bfd21d0>
```

```
NUM_EPOCHS = 2
history = model.fit(
        tf_train_dataset,
        validation_data=tf_validation_dataset,
        epochs=NUM_EPOCHS
        )
```

```
Epoch 1/2
615/615 [==============================] - 182s 193ms/step - loss:
0.2113 - val_loss: 0.1602
Epoch 2/2
615/615 [==============================] - 112s 182ms/step - loss:
0.1335 - val_loss: 0.1608
```

```
def compute_metrics(model, dataset: tf.data.Dataset, tokenizer):
    all_refs = []
    all_hyps = []
    iiii=0
    for batch in dataset:
        features, labels = batch
        generated_ids = model.generate(
            input_ids=features["input_ids"],
            attention_mask=features["attention_mask"],
            max_length=MAX_TARGET_LENGTH,
            num_beams=4,
            early_stopping=True
        )
        print(f"batch: {iiii}")
        iiii+=1
        hyps = tokenizer.batch_decode(generated_ids.numpy(),
skip_special_tokens=True)
        label_ids_np = labels.numpy()
        label_ids_for_decode = np.where(label_ids_np == -100,
tokenizer.pad_token_id, label_ids_np)
        refs = tokenizer.batch_decode(label_ids_for_decode,
skip_special_tokens=True)
        all_hyps.extend(hyps)
        all_refs.extend(refs)
        if iiii == 10:
          break
    overall_wer = wer(all_refs, all_hyps)
    overall_cer = cer(all_refs, all_hyps)

    return {
        "wer": overall_wer,
        "cer": overall_cer
        }
```

```
metrics = compute_metrics(model, tf_test_dataset, tokenizer)
print(f"\nTest Results → WER: {metrics['wer']:.3%}, CER:
{metrics['cer']:.3%}")
```

```
batch: 0
batch: 1
batch: 2
batch: 3
batch: 4
batch: 5
batch: 6
batch: 7
batch: 8
batch: 9

Test Results → WER: 3.877%, CER: 1.354%
```

```
def predict_spelling_single_string(model, tokenizer, raw_text, prefix,
max_input_length, max_target_length):
    tokenized_input = tokenizer(
        [raw_text],
        max_length=max_input_length,
        truncation=True,
        padding='max_length',
        return_tensors='tf'
    )
    generated_ids = model.generate(
        input_ids=tokenized_input["input_ids"],
        attention_mask=tokenized_input["attention_mask"],
        max_length=max_target_length,
        num_beams=4,
        early_stopping=True
    )
    prediction = tokenizer.batch_decode(generated_ids.numpy(),
skip_special_tokens=True)[0]
    return prediction

custom_sentences = [
    "Whre are you giong",
    "Teh meat will b tasty",
    "Harry where re yu living",
    "Iam runing int the cra",
    "my naem is Ziyad ",
    "hs name is essam"
]

print("--- Testing with Custom Sentences ---")
for sentence in custom_sentences:
    prediction = predict_spelling_single_string(
```

```
        model, tokenizer, sentence, PREFIX, MAX_INPUT_LENGTH,
MAX_TARGET_LENGTH
    )
    print(f"Input:     {sentence}")
    print(f"Predicted: {prediction}")
    print("-" * 30)

--- Testing with Custom Sentences ---
Input:     Whre are you giong
Predicted: Where are you going
------------------------------
Input:     Teh meat will b tasty
Predicted: The meat will be tasty
------------------------------
Input:     Harry where re yu living
Predicted: Harry where are you living
------------------------------
Input:     Iam runing int the cra
Predicted: I am running into the car
------------------------------
Input:     my naem is Ziyad
Predicted: My name is Ziyad
------------------------------
Input:     hs name is essam
Predicted: His name is essam
------------------------------

def predict_spelling_single_string(model, tokenizer, raw_text, prefix,
max_input_length, max_target_length):
    input_text = raw_text
    tokenized_input = tokenizer(
        [input_text],
        max_length=max_input_length,
        truncation=True,
        padding='max_length',
        return_tensors='tf'
    )

    generated_ids = model.generate(
        input_ids=tokenized_input["input_ids"],
        attention_mask=tokenized_input["attention_mask"],
        max_length=max_target_length,
        num_beams=4,
        early_stopping=True
    )
    prediction = tokenizer.batch_decode(generated_ids.numpy(),
skip_special_tokens=True)[0]
    return prediction
```

```
indices_to_test = [0, 1, 5, 65, 50]
for i in indices_to_test:
    misspelled_input = test_df.loc[i, 'misspelled']
    correct_reference = test_df.loc[i, 'correct']
    prediction = predict_spelling_single_string(
        model, tokenizer, misspelled_input, PREFIX, MAX_INPUT_LENGTH,
MAX_TARGET_LENGTH
    )
    print(f"Index {i}:")
    print(f"Input (Misspelled): {misspelled_input}")
    print(f"Reference (Correct):{correct_reference}")
    print(f"Predicted:          {prediction}")
    print("-" * 40)
```

```
Index 0:
Input (Misspelled): Ubandolier Budgie a free itupes app for iPad
iPhone and iPod touch released in December tellb thj story of the
making of Bandolier in the band s own words including an extensive
audio interview with Buzke Shelley
Reference (Correct):Bandolier Budgie a free iTunes app for iPad iPhone
and iPod touch released in December tells the story of the making of
Bandolier in the band s own words including an extensive audio
interview with Burke Shelley
Predicted:          Bandolier Budgie a free items app for iPad iPhone
and iPod touch released in December tells the story of the making of
Bandolier in the band s own words including an extensive audio
interview with Bruce Shelley
----------------------------------------
Index 1:
Input (Misspelled): Eden Black was grown fbom seed in the lat s by
Stephen Morley under his conditions it produces pitchers that are
almost completley black
Reference (Correct):Eden Black was grown from seed in the late s by
Stephen Morley under his conditions it produces pitchers that are
almost completley black
Predicted:          Ed Black was grown from seed in the late s by
Stephen Morley under his conditions it produces pitchers that are
almost completely black
----------------------------------------
Index 5:
Input (Misspelled): Aeqodynamic is an instrumental song by Daflt Punk
that is particularly well known for its robotic guitar solo
Reference (Correct):Aerodynamic is an instrumental song by Daft Punk
that is particularly well known for its robotic guitar solo
Predicted:          Aeqodynamic is an instrumental song by Dave Punk
that is particularly well known for its robotic guitar solo
----------------------------------------
Index 65:
Input (Misspelled): A crook is also a slang term or a criminal or a
person of questionable morality hte adjective crooked can refer to
```

```
such persons or actionsq
Reference (Correct):A crook is also a slang term for a criminal or a person of questionable morality the adjective crooked can refer to such persons or actions
Predicted:          A crook is also a slang term for a criminal or a person of questionable morality the adjective crook can refer to such persons or actions
----------------------------------------
Index 50:
Input (Misspelled): A estimate by tjhe International Organization for Mgiration suggests that between and Suanese re ilving in London whilst a fairly vague estimate of to ahs been placed in Briuhton
Reference (Correct):A estimate by the International Organization for Migration suggests that between and Sudanese are living in London whilst a fairly vague estimate of to has been placed in Brighton
Predicted:          A estimate by the International Organization for Migration suggests that between and Sudanese are living in London whilst a fairly vague estimate of to has been placed in Britain
----------------------------------------
```