Team Members:

- Nada Gaber 20217014
- Malak Sherif 20216137
- Malak Mahmoud 20217010
- Malak Gamal 20217009
- Zaynab ELAgamy 20215016

```
!pip install jiwer
```

```
Collecting jiwer
  Downloading jiwer-3.1.0-py3-none-any.whl.metadata (2.6 kB)
Requirement already satisfied: click>=8.1.8 in /usr/local/lib/python3.11/dist-packages (from jiwer) (8.1.8)
Collecting rapidfuzz>=3.9.7 (from jiwer)
  Downloading rapidfuzz-3.13.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Downloading jiwer-3.1.0-py3-none-any.whl (22 kB)
Downloading rapidfuzz-3.13.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.1/3.1 MB 72.1 MB/s eta 0:00:00
Installing collected packages: rapidfuzz, jiwer
Successfully installed jiwer-3.1.0 rapidfuzz-3.13.0
```

```
!pip install fuzzywuzzy
```

```
Collecting fuzzywuzzy
  Downloading fuzzywuzzy-0.18.0-py2.py3-none-any.whl.metadata (4.9 kB)
Downloading fuzzywuzzy-0.18.0-py2.py3-none-any.whl (18 kB)
Installing collected packages: fuzzywuzzy
Successfully installed fuzzywuzzy-0.18.0
```

```
import nltk
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```
import pandas as pd
import random
import tensorflow as tf
from transformers import TFT5ForConditionalGeneration, T5Tokenizer
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import SparseCategoricalCrossentropy
from tensorflow.keras.metrics import SparseCategoricalAccuracy
from transformers import AdamWeightDecay
from jiwer import wer, cer
from fuzzywuzzy import fuzz
from nltk.translate import meteor_score
```

```
/usr/local/lib/python3.11/dist-packages/fuzzywuzzy/fuzz.py:11: UserWarning: Using slow pure-python SequenceMatcher. Install python-Leven
  warnings.warn('Using slow pure-python SequenceMatcher. Install python-Levenshtein to remove this warning')
```

```
train_df = pd.read_csv("tune.tsv", sep="\t", names=["original", "split"])
val_df = pd.read_csv("validation.tsv", sep="\t", names=["original", "split"])
test_df = pd.read_csv("test.tsv", sep="\t", names=["original", "split"])
```

```
train_sentences = train_df["original"].tolist()
val_sentences = val_df["original"].tolist()
test_sentences = test_df["original"].tolist()
```

```
train_df
```

	original	split
0	' (1990) was the second sequel to appear , t...	' (1990) was the second sequel to appear . <...
1	' Maolain ' said to be diminutive of ' bald ' ...	' Maolain ' said to be diminuative of ' bald '...
2	' Skycruiser ' seating is available on the rem...	' Skycruiser ' seating is available on Boeing ...
3	' This group of nobles had supported the Engli...	' This group of nobles had supported the Engli...
4	' was a 14 - issue ongoing series by Devil 's ...	' was a fourteen issue ongoing series by Devil...
...	...	...
4995	Zaki Ibrahim is a South African - Canadian sin...	Zaki Ibrahim is a South African - Canadian sin...
4996	Zhang received many posthumous honors for his ...	Zhang received many posthumous honors for his ...
4997	Zhejiang is one of the provinces of China with...	Zhejiang is one of the provinces of China with...
4998	Zidisha financed three additional Kenya loans ...	Zidisha financed three pilot loans in the remo...
4999	Zubizarreta made his debut for Spain on 23 Jan...	Zubizarreta made his debut for Spain on 23 Jan...

5000 rows × 2 columns

Next steps: Generate code with train_df | View recommended plots | New interactive sheet

```
train_sentences
```

Show hidden output

```
keyboard_adj = {
    'a': 'qs', 'b': 'vn', 'c': 'xv', 'd': 'sf', 'e': 'wr', 'f': 'dg', 'g': 'fh',
    'h': 'gj', 'i': 'uo', 'j': 'hk', 'k': 'jl', 'l': 'k', 'm': 'n', 'n': 'bm',
    'o': 'ip', 'p': 'o', 'q': 'wa', 'r': 'et', 's': 'ad', 't': 'ry', 'u': 'yi',
    'v': 'cb', 'w': 'qe', 'x': 'zs', 'y': 'tu', 'z': 'xs'
}

alphabet = 'abcdefghijklmnopqrstuvwxyz'

context_confusions = {
    "their": "there", "there": "their", "they're": "there",
    "your": "you're", "you're": "your",
    "to": "too", "too": "two", "two": "too",
    "know": "no", "no": "know",
    "its": "it's", "it's": "its",
    "then": "than", "than": "then"
}


def corrupt_word(word, error_rate=0.2):
    if len(word) < 3 or random.random() > error_rate:
        return word

    error_type = random.choice(["keyboard", "phonetic", "duplicate", "basic"])

    idx = random.randint(0, len(word) - 1)

    if error_type == "keyboard":
        char = word[idx].lower()
        if char in keyboard_adj:
            typo_char = random.choice(keyboard_adj[char])
            return word[:idx] + typo_char + word[idx + 1:]

    elif error_type == "phonetic":
        swaps = {'ph': 'f', 'f': 'ph', 'c': 'k', 'k': 'c', 's': 'z', 'z': 's'}
        for key, val in swaps.items():
            if key in word:
                return word.replace(key, val, 1)

    elif error_type == "duplicate":
        return word[:idx] + word[idx] * 2 + word[idx:]

    elif error_type == "basic":
        basic_type = random.choice(["delete", "swap", "replace", "insert"])
        if basic_type == "delete":
            return word[:idx] + word[idx + 1:]
        elif basic_type == "swap" and len(word) > 1:
```

```
            if idx == len(word) - 1: idx -= 1
            word = list(word)
            word[idx], word[idx + 1] = word[idx + 1], word[idx]
            return ''.join(word)
        elif basic_type == "replace":
            return word[:idx] + random.choice(alphabet) + word[idx + 1:]
        elif basic_type == "insert":
            return word[:idx] + random.choice(alphabet) + word[idx:]

    return word


def apply_context_mistake(word):
    return context_confusions.get(word.lower(), word)


def generate_realistic_typo(sentence, context_prob=0.2, creative_prob=0.3):
    words = sentence.split()
    new_words = []
    for word in words:
        if random.random() < context_prob:
            word = apply_context_mistake(word)
        word = corrupt_word(word, error_rate=creative_prob)
        new_words.append(word)
    return ' '.join(new_words)


correct = "Their house is bigger than ours because they’re rich."
noisy = generate_realistic_typo(correct)
print("Original:", correct)
print("Corrupted:", noisy)
```

```
Original: Their house is bigger than ours because they’re rich.
Corrupted: Their house is bigger tjan ourz bekause they’re rich.
```

```
def generate_noisy_pairs(sentences, versions=3, clean_ratio=0.1):
    noisy_sentences = []
    correct_sentences = []

    for sentence in sentences:
        for i in range(versions):
            if random.random() < clean_ratio:
                noisy_sentences.append(sentence)
            else:
                corrupted = generate_realistic_typo(sentence)
                noisy_sentences.append(corrupted)

            correct_sentences.append(sentence)

    return pd.DataFrame({"noisy": noisy_sentences, "correct": correct_sentences})


train_pairs = generate_noisy_pairs(train_sentences)
val_pairs = generate_noisy_pairs(val_sentences)
test_pairs = generate_noisy_pairs(test_sentences)


train_pairs
```

Show hidden output

Get the Max sequence length

```
tokenizer = T5Tokenizer.from_pretrained("AventIQ-AI/t5-small-grammar-correction")
model = TFT5ForConditionalGeneration.from_pretrained("AventIQ-AI/t5-small-grammar-correction")
```

```
/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 secre
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(
tokenizer_config.json: 100%                                  21.8k/21.8k [00:00<00:00, 1.54MB/s]
spiece.model: 100%                                           792k/792k [00:00<00:00, 28.1MB/s]
added_tokens.json: 100%                                      2.69k/2.69k [00:00<00:00, 61.3kB/s]
special_tokens_map.json: 100%                                2.67k/2.67k [00:00<00:00, 277kB/s]
config.json: 100%                                            1.59k/1.59k [00:00<00:00, 179kB/s]
model.safetensors: 100%                                      121M/121M [00:05<00:00, 24.2MB/s]
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 pre
```

```
def get_max_token_length(sentences, tokenizer):
    max_len = 0
    lengths = []

    for sentence in sentences:
        tokens = tokenizer.encode(sentence)
        length = len(tokens)
        lengths.append(length)
        if length > max_len:
            max_len = length

    avg_len = sum(lengths) / len(lengths)
    p95_len = sorted(lengths)[int(len(lengths) * 0.95)]

    return {
        "max_length": max_len,
        "average_length": avg_len,
        "95th_percentile": p95_len,
        "length_distribution": lengths
    }


original_stats = get_max_token_length(train_sentences, tokenizer)
print(f"Original sentences stats:")
print(f"Max length: {original_stats['max_length']}")
print(f"Average length: {original_stats['average_length']:.2f}")
print(f"95th percentile: {original_stats['95th_percentile']}")

noisy_stats = get_max_token_length(train_pairs['noisy'].tolist(), tokenizer)
print(f"\nNoisy sentences stats:")
print(f"Max length: {noisy_stats['max_length']}")
print(f"Average length: {noisy_stats['average_length']:.2f}")
print(f"95th percentile: {noisy_stats['95th_percentile']}")
```

```
Original sentences stats:
Max length: 115
Average length: 47.75
95th percentile: 71

Noisy sentences stats:
Max length: 130
Average length: 60.03
95th percentile: 91
```

```
max_input_length = 128
max_target_length = 128

train_data = train_pairs.to_dict(orient="records")
val_data = val_pairs.to_dict(orient="records")
test_data = test_pairs.to_dict(orient="records")


train_data
```

Show hidden output

The current max_input_length=128 is sufficient to handle all original sentences because It covers almost all noisy sentences (only a tiny fraction exceed 128 tokens) The 95th percentile for noisy sentences is 91, well below our current limit

```
def encode(tokenizer, example, model_name):
  if model_name == 'T5':
    input_text = f"Fix spelling only, no other changes: {example['noisy']}"
  else:
    input_text = example['noisy']

  input = tokenizer(
      input_text, padding='max_length', truncation=True, max_length=max_input_length, return_tensors="tf"
  )
  target = tokenizer(
      example['correct'], padding='max_length', truncation=True, max_length=max_target_length, return_tensors="tf"
  )
  labels = target['input_ids'].numpy().squeeze()
  labels[labels == tokenizer.pad_token_id] = -100
  return {
      "input_ids": input['input_ids'].numpy().squeeze(),
      "attention_mask": input['attention_mask'].numpy().squeeze(),
      "labels": labels.astype('int32')
  }


train_dataset = tf.data.Dataset.from_generator(
    lambda: (encode(tokenizer, x, 'T5') for x in train_data),
    output_signature={
        "input_ids": tf.TensorSpec(shape=(128,), dtype=tf.int32),
        "attention_mask": tf.TensorSpec(shape=(128,), dtype=tf.int32),
        "labels": tf.TensorSpec(shape=(128,), dtype=tf.int32),
    }
)
val_dataset = tf.data.Dataset.from_generator(
    lambda: (encode(tokenizer, x, 'T5') for x in val_data),
    output_signature={
        "input_ids": tf.TensorSpec(shape=(128,), dtype=tf.int32),
        "attention_mask": tf.TensorSpec(shape=(128,), dtype=tf.int32),
        "labels": tf.TensorSpec(shape=(128,), dtype=tf.int32),
    }
)
test_dataset = tf.data.Dataset.from_generator(
    lambda: (encode(tokenizer, x, 'T5') for x in test_data),
    output_signature={
        "input_ids": tf.TensorSpec(shape=(128,), dtype=tf.int32),
        "attention_mask": tf.TensorSpec(shape=(128,), dtype=tf.int32),
        "labels": tf.TensorSpec(shape=(128,), dtype=tf.int32),
    }
)


train_dataset
```

```
<_FlatMapDataset element_spec={'input_ids': TensorSpec(shape=(128,), dtype=tf.int32, name=None), 'attention_mask': TensorSpec(shape=
(128,), dtype=tf.int32, name=None), 'labels': TensorSpec(shape=(128,), dtype=tf.int32, name=None)}>
```

```
BATCH_SIZE = 8
train_dataset = train_dataset.shuffle(1000).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
val_dataset = val_dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
test_dataset = test_dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)


train_dataset
```

```
<_PrefetchDataset element_spec={'input_ids': TensorSpec(shape=(None, 128), dtype=tf.int32, name=None), 'attention_mask':
TensorSpec(shape=(None, 128), dtype=tf.int32, name=None), 'labels': TensorSpec(shape=(None, 128), dtype=tf.int32, name=None)}>
```

```
model.compile(
    optimizer=AdamWeightDecay(learning_rate=3e-5),
)


model.config
```

Show hidden output

```
history=model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=2
)
```

```
Epoch 1/2
1875/1875 [==============================] - 503s 256ms/step - loss: 0.5484 - val_loss: 0.3891
Epoch 2/2
1875/1875 [==============================] - 463s 246ms/step - loss: 0.4303 - val_loss: 0.3431
```

Test

```
model.evaluate(test_dataset)
```

```
1875/1875 [==============================] - 135s 72ms/step - loss: 0.3375
0.33746886253356934
```

```
def generate_predictions(tokenizer, model, dataset, max_len=128):
    predictions = []
    references = []

    for batch in dataset:
        input_ids = batch['input_ids']
        attention_mask = batch['attention_mask']
        labels = batch['labels']

        outputs = model.generate(input_ids=input_ids, attention_mask=attention_mask, max_length=max_len, do_sample=False)
        decoded_preds = tokenizer.batch_decode(outputs.numpy(), skip_special_tokens=True)

        labels_for_decode = labels.numpy()
        labels_for_decode[labels_for_decode == -100] = tokenizer.pad_token_id
        decoded_labels = tokenizer.batch_decode(labels_for_decode, skip_special_tokens=True)

        predictions.extend(decoded_preds)
        references.extend(decoded_labels)

    return predictions, references


def compute_accuracy(preds, refs):
    correct = sum([p.strip() == r.strip() for p, r in zip(preds, refs)])
    return correct / len(refs)


from nltk.translate.bleu_score import corpus_bleu # N-Gram Overlap between the predictions and references

def compute_bleu(preds, refs):
    refs = [[ref.split()] for ref in refs]
    preds = [pred.split() for pred in preds]
    return corpus_bleu(refs, preds)


def spelling_error_rate(predictions, references):
    errors = 0
    total = 0

    for pred, ref in zip(predictions, references):
        pred_words = pred.split()
        ref_words = ref.split()

        total += len(ref_words)
        errors += sum([1 for p, r in zip(pred_words, ref_words) if p != r])

    error_rate = errors / total if total > 0 else 0
    return error_rate


def fuzzy_match_accuracy(predictions, references):
    total_score = 0
    total_words = 0

    for pred, ref in zip(predictions, references):
        pred_words = pred.split()
        ref_words = ref.split()
```

```
        for p, r in zip(pred_words, ref_words):
            total_score += fuzz.ratio(p, r)
            total_words += 1

    avg_score = total_score / total_words if total_words > 0 else 0
    return avg_score


def word_accuracy(predictions, references):
    correct = 0
    total = 0
    for pred, ref in zip(predictions, references):
        pred_words = pred.split()
        ref_words = ref.split()

        total += len(ref_words)
        correct += sum([1 for p, r in zip(pred_words, ref_words) if p == r])

    accuracy = correct / total if total > 0 else 0
    return accuracy


test_subset = test_dataset.take(10)
preds, refs = generate_predictions(tokenizer, model, test_subset)


accuracy_t5 = compute_accuracy(preds, refs)
ser_t5 = spelling_error_rate(preds, refs)
bleu_t5 = compute_bleu(preds, refs)
fuzzy_score_t5 = fuzzy_match_accuracy(preds, refs)
wer_score_t5 = wer(refs, preds)
cer_score_t5 = cer(refs, preds)
word_acc_t5 = word_accuracy(preds,refs)

print('---- T5 EVALUATION METRICS ----')
print(f"Test Accuracy: {accuracy_t5:.2f}")
print(f"Word Accuracy: {word_acc_t5:.2f}")
print(f"Fuzzy Match Accuracy: {fuzzy_score_t5:.2f}")
print(f"BLEU Score: {bleu_t5:.2f}")
print(f"Spelling Error Rate: {ser_t5:.2f}")
print(f"WER: {wer_score_t5:.2f}")
print(f"CER: {cer_score_t5:.2f}")
```

```
---- T5 EVALUATION METRICS ----
Test Accuracy: 0.12
Word Accuracy: 0.82
Fuzzy Match Accuracy: 89.08
BLEU Score: 0.78
Spelling Error Rate: 0.18
WER: 0.11
CER: 0.05
```

```
N = len(preds)
noisy_inputs = test_pairs["noisy"].tolist()[:N]
original_refs = test_pairs["correct"].tolist()[:N]

for noisy, pred, ref in list(zip(noisy_inputs, preds, original_refs))[:5]:
    print("REF   :", ref)
    print("NOISY: ", noisy)
    print("PRED  :", pred)
    print("---")
```

```
REF   : ' Bandolier - Budgie ' , a free iTunes app for iPad , iPhone and iPod touch , released in December 2011 , tells the story of the
NOISY:  ' Bandolier - Budgie ' , a free iTunes app for iPad , iPhoneee and iPo touch , released in December 2011 , teeells the stiry of
PRED  : ' Bandolier - Budgie ', a free iTunes app for iPad, iPhone and iPod touch, released in December 2011, describes the stiry of the
---
REF   : ' Bandolier - Budgie ' , a free iTunes app for iPad , iPhone and iPod touch , released in December 2011 , tells the story of the
NOISY:  ' Bandolier - Budgei ' , a free iTunes app for iPad , iPhone and iPod touch , released in December 2011 , tttells the story of t
PRED  : ' Bandolier - Budgei ', a free iTunes app for iPad, iPhone and iPod touch, released in December 2011, tells the story of the mak
---
REF   : ' Bandolier - Budgie ' , a free iTunes app for iPad , iPhone and iPod touch , released in December 2011 , tells the story of the
NOISY:  ' Bandolier - Budbie ' , a free iTunes app for iPad , iPhone and iPod touch , released in December 2011 , tells thje story of th
PRED  : ' Bandolier - Budbie ', a free iTunes app for iPad, iPhone and iPod touch, released in December 2011, tells the story of the mak
---
REF   : ' Eden Black ' was grown from seed in the late 1980s by Stephen Morley , under his conditions it produces pitchers that are almo
NOISY:  ' Eden Black ' was grown from seed in the late 1980s by Stephen Morley , under his conditions it produces pitchers that are almo
PRED  : ' Eden Black'was grown from seed in the late 1980s by Stephen Morley, under his conditions it produces pitchers that are almost
---
```

```
REF   : ' Eden Black ' was grown from seed in the late 1980s by Stephen Morley , under his conditions it produces pitchers that are almo
NOISY:  ' Eden Black ' was grown from seed in the late 1980s by Stephen Morllley , under his conditiosn it produces pitchers that are al
PRED  : ' Eden Black'was grown from seed in the late 1980s by Stephen Morley, under his belt it produces pitchers that are almost comple
---
```

```
from transformers import TFBartForConditionalGeneration, BartTokenizer

bart_tokenizer = BartTokenizer.from_pretrained("veghar/spell_correct_bart_base")
bart_model = TFBartForConditionalGeneration.from_pretrained("veghar/spell_correct_bart_base")
```

```
tokenizer_config.json: 100%  1.31k/1.31k [00:00<00:00, 80.1kB/s]
vocab.json: 100%  999k/999k [00:00<00:00, 4.54MB/s]
merges.txt: 100%  456k/456k [00:00<00:00, 1.68MB/s]
special_tokens_map.json: 100%  957/957 [00:00<00:00, 102kB/s]
config.json: 100%  1.74k/1.74k [00:00<00:00, 102kB/s]
tf_model.h5: 100%  558M/558M [00:22<00:00, 24.4MB/s]
All model checkpoint layers were used when initializing TFBartForConditionalGeneration.

All the layers of TFBartForConditionalGeneration were initialized from the model checkpoint at veghar/spell_correct_bart_base.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFBartForConditionalGeneration for p
generation_config.json: 100%  292/292 [00:00<00:00, 33.7kB/s]
```

```
bart_train_dataset = tf.data.Dataset.from_generator(
    lambda: (encode(bart_tokenizer, x, 'bart') for x in train_data),
    output_signature={
        "input_ids": tf.TensorSpec(shape=(128,), dtype=tf.int32),
        "attention_mask": tf.TensorSpec(shape=(128,), dtype=tf.int32),
        "labels": tf.TensorSpec(shape=(128,), dtype=tf.int32),
    }
)
bart_val_dataset = tf.data.Dataset.from_generator(
    lambda: (encode(bart_tokenizer, x, 'bart') for x in val_data),
    output_signature={
        "input_ids": tf.TensorSpec(shape=(128,), dtype=tf.int32),
        "attention_mask": tf.TensorSpec(shape=(128,), dtype=tf.int32),
        "labels": tf.TensorSpec(shape=(128,), dtype=tf.int32),
    }
)
bart_test_dataset = tf.data.Dataset.from_generator(
    lambda: (encode(bart_tokenizer, x, 'bart') for x in test_data),
    output_signature={
        "input_ids": tf.TensorSpec(shape=(128,), dtype=tf.int32),
        "attention_mask": tf.TensorSpec(shape=(128,), dtype=tf.int32),
        "labels": tf.TensorSpec(shape=(128,), dtype=tf.int32),
    }
)


BATCH_SIZE = 8
bart_train_dataset = bart_train_dataset.shuffle(1000).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
bart_val_dataset = bart_val_dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
bart_test_dataset = bart_test_dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)


bart_model.compile(
    optimizer=AdamWeightDecay(learning_rate=3e-5),
)


bart_history = bart_model.fit(
    bart_train_dataset,
    validation_data=bart_val_dataset,
    epochs=2
)
```

```
Epoch 1/2
1875/1875 [==============================] - 1022s 529ms/step - loss: 0.6553 - val_loss: 0.3181
Epoch 2/2
1875/1875 [==============================] - 984s 524ms/step - loss: 0.2758 - val_loss: 0.2610
```

```
bart_test_subset = bart_test_dataset.take(10)
bart_preds, bart_refs = generate_predictions(bart_tokenizer, bart_model, bart_test_subset)
```

```
bart_model.evaluate(bart_test_dataset)
```

```
1875/1875 [==============================] - 269s 143ms/step - loss: 0.2577
0.2576565444469452
```

```
bart_accuracy = compute_accuracy(bart_preds, bart_refs)
bart_bleu = compute_bleu(bart_preds, bart_refs)
bart_ser = spelling_error_rate(bart_preds, bart_refs)
bart_fuzzy_score = fuzzy_match_accuracy(bart_preds, bart_refs)
bart_wer_score = wer(bart_refs, bart_preds)
bart_cer_score = cer(bart_refs, bart_preds)
bart_word_acc = word_accuracy(bart_preds, bart_refs)
```

```
print('---- BART EVALUATION METRICS ----')
print(f"BART Test Accuracy: {bart_accuracy:.2f}")
print(f"BART BLEU Score: {bart_bleu:.2f}")
print(f"Spelling Error Rate: {bart_ser:.2f}")
print(f"Fuzzy Match Accuracy: {bart_fuzzy_score:.2f}")
print(f"WER: {bart_wer_score:.2f}")
print(f"CER: {bart_cer_score:.2f}")
print(f"Word Accuracy: {bart_word_acc:.2f}")
```

```
---- BART EVALUATION METRICS ----
BART Test Accuracy: 0.16
BART BLEU Score: 0.84
Spelling Error Rate: 0.11
Fuzzy Match Accuracy: 93.08
WER: 0.07
CER: 0.04
Word Accuracy: 0.88
```

```
N = len(bart_preds)
bart_noisy_inputs = test_pairs["noisy"].tolist()[:N]
bart_original_refs = test_pairs["correct"].tolist()[:N]

for noisy, pred, ref in list(zip(bart_noisy_inputs, bart_preds, bart_original_refs))[:5]:
    print("REF   :", ref)
    print("NOISY: ", noisy)
    print("PRED  :", pred)
    print("---")
```

```
REF   : ' Bandolier - Budgie ' , a free iTunes app for iPad , iPhone and iPod touch , released in December 2011 , tells the story of the
NOISY:  ' Bandolier - Budgie ' , a free iTunes app for iPad , iPhoneee and iPo touch , released in December 2011 , teeells the stiry of
PRED  : ' Bandolier - Budgie ' , a free iTunes app for iPad , iPhone and iPod touch , released in December 2011 , tells the story of the
---
REF   : ' Bandolier - Budgie ' , a free iTunes app for iPad , iPhone and iPod touch , released in December 2011 , tells the story of the
NOISY:  ' Bandolier - Budgei ' , a free iTunes app for iPad , iPhone and iPod touch , released in December 2011 , tttells the story of t
PRED  : ' Bandolier - Budgei ' , a free iTunes app for iPad , iPhone and iPod touch , released in December 2011 , tells the story of the
---
REF   : ' Bandolier - Budgie ' , a free iTunes app for iPad , iPhone and iPod touch , released in December 2011 , tells the story of the
NOISY:  ' Bandolier - Budbie ' , a free iTunes app for iPad , iPhone and iPod touch , released in December 2011 , tells thje story of th
PRED  : ' Bandolier - Budbie ' , a free iTunes app for iPad , iPhone and iPod touch , released in December 2011 , tells the story of the
---
REF   : ' Eden Black ' was grown from seed in the late 1980s by Stephen Morley , under his conditions it produces pitchers that are almo
NOISY:  ' Eden Black ' was grown from seed in the late 1980s by Stephen Morley , under his conditions it produces pitchers that are almo
PRED  : ' Eden Black ' was grown from seed in the late 1980s by Stephen Morley , under his conditions it produces pitchers that are almo
---
REF   : ' Eden Black ' was grown from seed in the late 1980s by Stephen Morley , under his conditions it produces pitchers that are almo
NOISY:  ' Eden Black ' was grown from seed in the late 1980s by Stephen Morllley , under his conditiosn it produces pitchers that are al
PRED  : ' Eden Black ' was grown from seed in the late 1980s by Stephen Morley , under his command it produces pitchers that are almost
---
```

3rd model

```
from transformers import TFBartForConditionalGeneration, BartTokenizer

bart_tokenizer_v2 = BartTokenizer.from_pretrained("facebook/bart-base")
bart_model_v2 = TFBartForConditionalGeneration.from_pretrained("facebook/bart-base")
```

```
/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 secre
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(
vocab.json: 100%                                                    899k/899k [00:00<00:00, 3.61MB/s]
merges.txt: 100%                                                    456k/456k [00:00<00:00, 5.55MB/s]
tokenizer.json: 100%                                                1.36M/1.36M [00:00<00:00, 4.13MB/s]
config.json: 100%                                                   1.72k/1.72k [00:00<00:00, 195kB/s]
model.safetensors: 100%                                             558M/558M [00:01<00:00, 330MB/s]
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 p
```

```
bart_train_dataset = tf.data.Dataset.from_generator(
    lambda: (encode(bart_tokenizer_v2, x, 'bart') for x in train_data),
    output_signature={
        "input_ids": tf.TensorSpec(shape=(128,), dtype=tf.int32),
        "attention_mask": tf.TensorSpec(shape=(128,), dtype=tf.int32),
        "labels": tf.TensorSpec(shape=(128,), dtype=tf.int32),
    }
)
bart_val_dataset = tf.data.Dataset.from_generator(
    lambda: (encode(bart_tokenizer_v2, x, 'bart') for x in val_data),
    output_signature={
        "input_ids": tf.TensorSpec(shape=(128,), dtype=tf.int32),
        "attention_mask": tf.TensorSpec(shape=(128,), dtype=tf.int32),
        "labels": tf.TensorSpec(shape=(128,), dtype=tf.int32),
    }
)
bart_test_dataset = tf.data.Dataset.from_generator(
    lambda: (encode(bart_tokenizer_v2, x, 'bart') for x in test_data),
    output_signature={
        "input_ids": tf.TensorSpec(shape=(128,), dtype=tf.int32),
        "attention_mask": tf.TensorSpec(shape=(128,), dtype=tf.int32),
        "labels": tf.TensorSpec(shape=(128,), dtype=tf.int32),
    }
)


BATCH_SIZE = 8
bart_train_dataset = bart_train_dataset.shuffle(1000).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
bart_val_dataset = bart_val_dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
bart_test_dataset = bart_test_dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)


bart_model_v2.compile(
    optimizer=AdamWeightDecay(learning_rate=3e-5, weight_decay_rate=0.01),
)


bart_history = bart_model_v2.fit(
    bart_train_dataset,
    validation_data=bart_val_dataset,
    epochs=2
)
```

```
Epoch 1/2
1875/1875 [==============================] - 1014s 538ms/step - loss: 0.4056 - val_loss: 0.2362
Epoch 2/2
1875/1875 [==============================] - 1016s 541ms/step - loss: 0.2018 - val_loss: 0.2085
```

```
bart_model_v2.evaluate(bart_test_dataset)
```

```
1875/1875 [==============================] - 269s 144ms/step - loss: 0.2112
0.21117892861366272
```

```
bart_test_subset2 = bart_test_dataset.take(10)
bart_preds2, bart_refs2 = generate_predictions(bart_tokenizer_v2, bart_model_v2, bart_test_subset2)
```

```
bart_accuracy2 = compute_accuracy(bart_preds2, bart_refs2)
bart_bleu2 = compute_bleu(bart_preds2, bart_refs2)
ser2 = spelling_error_rate(bart_preds2, bart_refs2)
fuzzy_score2 = fuzzy_match_accuracy(bart_preds2, bart_refs2)
bart_wer_score2 = wer(bart_refs2, bart_preds2)
bart_cer_score2 = cer(bart_refs2, bart_preds2)
word_acc2 = word_accuracy(bart_preds2, bart_refs2)


print('---- BART BASE EVALUATION METRICS ----')
print(f"BART Test Accuracy: {bart_accuracy2:.2f}")
print(f"BART BLEU Score: {bart_bleu2:.2f}")
print(f"Spelling Error Rate: {ser2:.2f}")
print(f"Fuzzy Match Accuracy: {fuzzy_score2:.2f}")
print(f"WER: {bart_wer_score2:.2f}")
print(f"CER: {bart_cer_score2:.2f}")
print(f"Word Accuracy: {word_acc2:.2f}")
```

```
---- BART BASE EVALUATION METRICS ----
BART Test Accuracy: 0.19
BART BLEU Score: 0.87
Spelling Error Rate: 0.14
Fuzzy Match Accuracy: 90.32
WER: 0.06
CER: 0.03
Word Accuracy: 0.86
```

```
N = len(bart_preds2)
bart_noisy_inputs = test_pairs["noisy"].tolist()[:N]
bart_original_refs = test_pairs["correct"].tolist()[:N]

for noisy, pred, ref in list(zip(bart_noisy_inputs, bart_preds2, bart_original_refs))[:5]:
    print("REF   :", ref)
    print("NOISY: ", noisy)
    print("PRED  :", pred)
    print("---")
```

```
REF   : ' Bandolier - Budgie ' , a free iTunes app for iPad , iPhone and iPod touch , released in December 2011 , tells the story of the
NOISY:  ' Bandolier - Budgie ' , a free iTunesss app gor iPad , iPhonnne and iPod toucg , releazed in December 0211 , tellz te story of
PRED  : ' Bandolier - Budgie ' , a free iTunes app for iPad , iPad and iPod Touch , released in December 2011 , tells the story of the m
---
REF   : ' Bandolier - Budgie ' , a free iTunes app for iPad , iPhone and iPod touch , released in December 2011 , tells the story of the
NOISY:  ' Bandolier - Budgie ' , a free iunes app for iPad , iPhone and iPod tttouch , released in December 2p11 , tellz the story of th
PRED  : ' Bandolier - Budgie ' , a free iunes app for iPad , iPhone and iPod touch , released in December 2011 , tells the story of the
---
REF   : ' Bandolier - Budgie ' , a free iTunes app for iPad , iPhone and iPod touch , released in December 2011 , tells the story of the
NOISY:  ' Bandolier - Budgie ' , a freeee iTunes app for iPad , iPhone amd iPod touch , released in December 2011 , tells the story of t
PRED  : ' Bandolier - Budgie ' , a free iTunes app for iPad , iPhone and iPod touch , released in December 2011 , tells the story of the
---
REF   : ' Eden Black ' was grown from seed in the late 1980s by Stephen Morley , under his conditions it produces pitchers that are almo
NOISY:  ' Eden Black ' was grown from seed in the late 1980s by Stepheeen Morley , under his conditions it prrroduces pitchers that are
PRED  : ' Eden Black ' was grown from seed in the late 1980s by Stephen Morley , under his conditions it produces pitchers that are almo
---
REF   : ' Eden Black ' was grown from seed in the late 1980s by Stephen Morley , under his conditions it produces pitchers that are almo
NOISY:  ' Eden Black ' was grown phrom seeeed in the late 1980es by Stephen MMMorley , uuunder his conditions it produkes pitchers that
PRED  : ' Eden Black ' was grown from seed in the late 1980s by Stephen Mooreley , under his conditions it produces pitchers that are al
```