```
# --- Environment Setup ---
!pip install transformers datasets evaluate jiwer sentencepiece
accelerate -q
!pip show transformers datasets evaluate jiwer # Optional: check
versions

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.0/84.0 kB 2.0 MB/s eta
0:00:0000:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 193.6/193.6 kB 5.7 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.1/3.1 MB 39.3 MB/s eta
0:00:0000:0100:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 4.3 MB/s eta
0:00:000:00:010:01m
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 1.9 MB/s eta
0:00:00:00:010:02mm
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 1.4 MB/s eta
0:00:00:00:0100:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 10.1 MB/s eta
0:00:0000:0100:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 6.6 MB/s eta
0:00:00:00:01:00:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 4.1 MB/s eta
0:00:00:00:0100:01m
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 7.8 MB/s eta
0:00:000:00:010:01
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
cesium 0.12.4 requires numpy<3.0,>=2.0, but you have numpy 1.26.4
which is incompatible.
bigframes 1.42.0 requires rich<14,>=12.4.4, but you have rich 14.0.0
which is incompatible.
gcsfs 2025.3.2 requires fsspec==2025.3.2, but you have fsspec 2025.3.0
which is incompatible.
Name: transformers
Version: 4.51.3
Summary: State-of-the-art Machine Learning for JAX, PyTorch and
TensorFlow
Home-page: https://github.com/huggingface/transformers
Author: The Hugging Face team (past and future) with the help of all
our contributors
(https://github.com/huggingface/transformers/graphs/contributors)
Author-email: transformers@huggingface.co
License: Apache 2.0 License
Location: /usr/local/lib/python3.11/dist-packages
Requires: filelock, huggingface-hub, numpy, packaging, pyyaml, regex,
requests, safetensors, tokenizers, tqdm
Required-by: kaggle-environments, peft, sentence-transformers
---
```

```
Name: datasets
Version: 3.6.0
Summary: HuggingFace community-driven open-source library of datasets
Home-page: https://github.com/huggingface/datasets
Author: HuggingFace Inc.
Author-email: thomas@huggingface.co
License: Apache 2.0
Location: /usr/local/lib/python3.11/dist-packages
Requires: dill, filelock, fsspec, huggingface-hub, multiprocess,
numpy, packaging, pandas, pyarrow, pyyaml, requests, tqdm, xxhash
Required-by: evaluate, torchtune
---
Name: evaluate
Version: 0.4.3
Summary: HuggingFace community-driven open-source library of
evaluation
Home-page: https://github.com/huggingface/evaluate
Author: HuggingFace Inc.
Author-email: leandro@huggingface.co
License: Apache 2.0
Location: /usr/local/lib/python3.11/dist-packages
Requires: datasets, dill, fsspec, huggingface-hub, multiprocess,
numpy, packaging, pandas, requests, tqdm, xxhash
Required-by:
---
Name: jiwer
Version: 3.1.0
Summary: Evaluate your speech-to-text system with similarity measures
such as word error rate (WER)
Home-page:
Author:
Author-email: Nik Vaessen <nikvaes@gmail.com>
License:
Location: /usr/local/lib/python3.11/dist-packages
Requires: click, rapidfuzz
Required-by:
```

```python
# --- Imports ---
import os
import random
import re
import gc # For garbage collection
import warnings

import numpy as np
import pandas as pd
import torch
from datasets import Dataset as HFDataset
from nltk.metrics.distance import edit_distance
import nltk
```

```python
from sklearn.model_selection import train_test_split
from tqdm.auto import tqdm # For progress bars

from transformers import (
    T5ForConditionalGeneration,
    T5Tokenizer,
    Seq2SeqTrainingArguments,
    Seq2SeqTrainer,
    DataCollatorForSeq2Seq,
    EarlyStoppingCallback
)
import evaluate

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
if torch.cuda.is_available():
    print(f"GPU Name: {torch.cuda.get_device_name(0)}")
try:
    nltk.data.find('tokenizers/punkt')
except nltk.downloader.DownloadError:
    print("Downloading NLTK punkt tokenizer...")
    nltk.download('punkt', quiet=True)
    print("NLTK punkt tokenizer downloaded.")

# Set random seeds for reproducibility
def set_seed(seed_value=42):
    random.seed(seed_value)
    np.random.seed(seed_value)
    torch.manual_seed(seed_value)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed_value)
    print(f"Seed set to {seed_value}")

set_seed(42)

# Define paths (CRITICAL: ADJUST DATA_DIR)
DATA_DIR =
"/kaggle/input/transformer-mode-for-spelling-correction/wiki-split-
master"


CACHE_DIR = "/kaggle/working/cache_dir_spell_correction"
OUTPUT_DIR = "/kaggle/working/spell_correction_outputs"

os.makedirs(CACHE_DIR, exist_ok=True)
os.makedirs(OUTPUT_DIR, exist_ok=True)
print(f"Cache directory: {os.path.abspath(CACHE_DIR)}")
print(f"Output directory: {os.path.abspath(OUTPUT_DIR)}")
```

```
# Model Configuration
MODEL_NAME = "t5-small"
print(f"Using model: {MODEL_NAME}")

# Dataset Configuration (ADJUST THESE FOR YOUR RUN)
# Number of ORIGINAL correct sentences to load from WikiSplit
# For a quick test:
MAX_TRAIN_SAMPLES_ORIGINAL = 5000  # Reduced for quick testing
MAX_VAL_SAMPLES_ORIGINAL = 500    # Reduced
MAX_TEST_SAMPLES_ORIGINAL = 500   # Reduced
# For a more serious run, increase these significantly (e.g., 50000,
5000, 5000)

VARIANTS_PER_SENTENCE = 1 # Number of misspelled versions per correct
sentence
print(f"Max original train samples: {MAX_TRAIN_SAMPLES_ORIGINAL},
Validation: {MAX_VAL_SAMPLES_ORIGINAL}, Test:
{MAX_TEST_SAMPLES_ORIGINAL}")
print(f"Misspelled variants per sentence: {VARIANTS_PER_SENTENCE}")

# Tokenizer Configuration
MAX_SOURCE_LENGTH = 128 # Max length for misspelled input
MAX_TARGET_LENGTH = 128 # Max length for corrected output
print(f"Max source length: {MAX_SOURCE_LENGTH}, Max target length:
{MAX_TARGET_LENGTH}")
```

```
Using device: cpu
Seed set to 42
Cache directory: /kaggle/working/cache_dir_spell_correction
Output directory: /kaggle/working/spell_correction_outputs
Using model: t5-small
Max original train samples: 5000, Validation: 500, Test: 500
Misspelled variants per sentence: 1
Max source length: 128, Max target length: 128
```

```
# --- 1. Load the Sentences (from WikiSplit) ---

def load_wikisplit_data(file_path, max_samples=None,
file_label="data"):

    print(f"Loading {file_label} from {file_path}...")
    try:
        df = pd.read_csv(file_path, sep='\t', header=None,
names=['correct', 'simple'], on_bad_lines='warn')
        df = df[['correct']] # Keep only the 'correct' column
        df.dropna(subset=['correct'], inplace=True) # Remove rows
where 'correct' is NaN
        df['correct'] = df['correct'].astype(str).str.strip() # Ensure
string type and strip whitespace
        df = df[df['correct'] != ''] # Remove empty strings
    except FileNotFoundError:
        print(f"ERROR: File not found at {file_path}. Please ensure
DATA_DIR is correct and file exists.")
        return pd.DataFrame(columns=['correct'])
    except Exception as e:
        print(f"An error occurred while loading {file_path}: {e}")
        return pd.DataFrame(columns=['correct'])

    if df.empty:
        print(f"Warning: No data loaded from {file_path}. The
DataFrame is empty.")
        return df

    if max_samples and len(df) > max_samples:
        print(f"Sampling {max_samples} from {len(df)} available
sentences.")
        df = df.sample(n=max_samples, random_state=42) # Use fixed
random_state for reproducibility

    print(f"Successfully loaded {len(df)} samples from {file_path}.")
    return df

# Load original correct sentences from WikiSplit files
print("\nLoading original correct sentences from WikiSplit files...")
train_df_orig = load_wikisplit_data(os.path.join(DATA_DIR,
"train.tsv"), MAX_TRAIN_SAMPLES_ORIGINAL, "training data")
val_df_orig = load_wikisplit_data(os.path.join(DATA_DIR,
"validation.tsv"), MAX_VAL_SAMPLES_ORIGINAL, "validation data")
test_df_orig = load_wikisplit_data(os.path.join(DATA_DIR, "test.tsv"),
MAX_TEST_SAMPLES_ORIGINAL, "test data")

# Basic check to ensure data was loaded
if train_df_orig.empty:
    raise ValueError(
        "Training data (train_df_orig) is empty. "
        "Please check your DATA_DIR path and ensure 'train.tsv' exists
and is readable. "
        "Current DATA_DIR: " + os.path.abspath(DATA_DIR)
    )
if val_df_orig.empty:
    print("Warning: Validation data (val_df_orig) is empty. Validation
during training will be skipped or fail.")
if test_df_orig.empty:
    print("Warning: Test data (test_df_orig) is empty. Final
evaluation on test set will be skipped.")

print(f"\nNumber of original correct sentences loaded:")
print(f"  Train: {len(train_df_orig)}")
print(f"  Validation: {len(val_df_orig)}")
print(f"  Test: {len(test_df_orig)}")
```

```
Loading original correct sentences from WikiSplit files...
Loading training data from /kaggle/input/transformer-mode-for-
spelling-correction/wiki-split-master/train.tsv...
Sampling 5000 from 989944 available sentences.
Successfully loaded 5000 samples from /kaggle/input/transformer-mode-
for-spelling-correction/wiki-split-master/train.tsv.
Loading validation data from /kaggle/input/transformer-mode-for-
spelling-correction/wiki-split-master/validation.tsv...
Sampling 500 from 5000 available sentences.
Successfully loaded 500 samples from /kaggle/input/transformer-mode-
for-spelling-correction/wiki-split-master/validation.tsv.
Loading test data from /kaggle/input/transformer-mode-for-spelling-
correction/wiki-split-master/test.tsv...
Sampling 500 from 5000 available sentences.
Successfully loaded 500 samples from /kaggle/input/transformer-mode-
for-spelling-correction/wiki-split-master/test.tsv.

Number of original correct sentences loaded:
  Train: 5000
  Validation: 500
  Test: 500
```

```python
# --- 2. Generate Sentences with Spelling Mistakes (Creative Approach)
---

def generate_misspelled_sentence(sentence, error_prob_word=0.25,
char_error_types=None):
    """
    Introduces various character-level spelling errors into a
sentence.
    error_prob_word: Probability that a word will have an error
introduced.
    """
    if char_error_types is None:
        # Common character error types
        char_error_types = ['swap', 'delete', 'insert', 'replace',
'duplicate']

    words = sentence.split(' ') # Simple split, could use
nltk.word_tokenize for more robustness
    misspelled_words = []
    alphabet = "abcdefghijklmnopqrstuvwxyz" # For
insertions/replacements

    for word in words:
        # Only attempt to misspell words with alphabetic characters
and some length
        if random.random() < error_prob_word and len(word) > 1 and
any(c.isalpha() for c in word):
```

```python
            chars = list(word)
            chosen_error_type = random.choice(char_error_types)

            can_apply_error = False
            if chosen_error_type == 'swap' and len(chars) >= 2:
                idx = random.randint(0, len(chars) - 2)
                chars[idx], chars[idx+1] = chars[idx+1], chars[idx]
                can_apply_error = True
            elif chosen_error_type == 'delete' and len(chars) >= 2: #
Avoid deleting the only char
                idx_to_delete = random.randint(0, len(chars) - 1)
                chars.pop(idx_to_delete)
                can_apply_error = True
            elif chosen_error_type == 'insert':
                idx_to_insert = random.randint(0, len(chars))
                chars.insert(idx_to_insert, random.choice(alphabet))
                can_apply_error = True
            elif chosen_error_type == 'replace' and len(chars) >= 1:
                idx_to_replace = random.randint(0, len(chars) - 1)
                original_char = chars[idx_to_replace]
                if original_char.isalpha(): # Only replace alphabetic
chars meaningfully
                    new_char = random.choice(alphabet)
                    while new_char == original_char.lower() and
len(alphabet) > 1: # Avoid replacing with same char
                        new_char = random.choice(alphabet)
                    chars[idx_to_replace] = new_char if
original_char.islower() else new_char.upper()
                    can_apply_error = True
            elif chosen_error_type == 'duplicate' and len(chars) >= 1:
                idx_to_duplicate = random.randint(0, len(chars) - 1)
                chars.insert(idx_to_duplicate,
chars[idx_to_duplicate]) # Duplicate the char at idx
                can_apply_error = True

            if can_apply_error:
                misspelled_words.append("".join(chars))
            else: # If error couldn't be applied (e.g., word too short
for swap/delete)
                misspelled_words.append(word)
        else:
            misspelled_words.append(word)

    return " ".join(misspelled_words)

# Test the generator
test_sentence = "This is a sample sentence for testing the error
generator."
print(f"Original test sentence: {test_sentence}")
```

```
for i in range(3):
    print(f"Misspelled variant {i+1}:
{generate_misspelled_sentence(test_sentence)}")
```

```
Original test sentence: This is a sample sentence for testing the
error generator.
Misspelled variant 1: This his a ssample sentence ofr ttesting the
error generator.
Misspelled variant 2: This is a sample sentence fir tcesting the error
generator.
Misspelled variant 3: This ms a sample sentence for testing hte error
generator.
```

```
# --- 3. Create Train, Validation, and Test Datasets with Misspelled
Pairs ---

def create_spelling_dataset_from_correct(df_correct,
num_variants_per_sentence=1, dataset_label="data"):
    """
    Generates misspelled versions for each correct sentence in the
DataFrame.
    Returns a DataFrame with 'misspelled' and 'correct' columns.
    """
    data_pairs = []
    if df_correct.empty:
        print(f"Warning: Input DataFrame for {dataset_label} is empty.
No misspelled variants will be generated.")
        return pd.DataFrame(columns=['misspelled', 'correct'])

    print(f"\nGenerating {num_variants_per_sentence} misspelled
variant(s) per sentence for {dataset_label}...")
    for _, row in tqdm(df_correct.iterrows(), total=len(df_correct),
desc=f"Generating for {dataset_label}"):
        correct_sentence = row['correct']

        # Basic filter: ensure sentence is a string and not
excessively long
        # (MAX_SOURCE_LENGTH - 10 reserves tokens for the "correct
spelling: " prefix)
        if not isinstance(correct_sentence, str) or
len(correct_sentence.split()) > (MAX_SOURCE_LENGTH - 10):
            continue # Skip this sentence

        for _ in range(num_variants_per_sentence):
            misspelled =
generate_misspelled_sentence(correct_sentence)
            # Ensure the generated misspelled sentence is actually
different and not empty
            if misspelled != correct_sentence and misspelled.strip():
                data_pairs.append({'misspelled': misspelled,
'correct': correct_sentence})
            # else:
                # Optional: Log if no valid misspelling was generated
for a sentence
                # print(f"Could not generate a distinct non-empty
misspelling for: {correct_sentence}")

    return pd.DataFrame(data_pairs)

# Generate misspelled datasets
train_data_df = create_spelling_dataset_from_correct(train_df_orig,
VARIANTS_PER_SENTENCE, "training")
val_data_df = create_spelling_dataset_from_correct(val_df_orig,
VARIANTS_PER_SENTENCE, "validation")
test_data_df = create_spelling_dataset_from_correct(test_df_orig,
VARIANTS_PER_SENTENCE, "test") # Keep for final demo

print(f"\nNumber of generated misspelled/correct pairs:")
print(f"  Train: {len(train_data_df)}")
print(f"  Validation: {len(val_data_df)}")
print(f"  Test: {len(test_data_df)}")


# Display a few examples from the generated training data
if not train_data_df.empty:
    print("\nSample generated training data (first 3 pairs):")
    for i in range(min(3, len(train_data_df))):
        print(f"  Correct   : {train_data_df.iloc[i]['correct']}")
        print(f"  Misspelled: {train_data_df.iloc[i]['misspelled']}")
        print("-" * 30)
else:
    print("Warning: No training data (train_data_df) was generated.
Training will likely fail or be skipped.")
    # Potentially raise an error if training data is essential and
empty
    # raise ValueError("Training data generation resulted in an empty
dataset.")

# Convert pandas DataFrames to Hugging Face Dataset objects
if not train_data_df.empty:
    train_dataset_hf = HFDataset.from_pandas(train_data_df)
else:
    train_dataset_hf = HFDataset.from_dict({'misspelled': [],
'correct': []}) # Empty HF dataset

if not val_data_df.empty:
    val_dataset_hf = HFDataset.from_pandas(val_data_df)
else:
    val_dataset_hf = HFDataset.from_dict({'misspelled': [], 'correct':
[]})
```

```
if not test_data_df.empty:
    test_dataset_hf_for_tokenization =
HFDataset.from_pandas(test_data_df) # For tokenization
else:
    test_dataset_hf_for_tokenization =
HFDataset.from_dict({'misspelled': [], 'correct': []})
# Note: test_data_df (pandas) is kept for the final string-based
demonstration

# Clean up original DataFrames to save memory (if they are large)
del train_df_orig, val_df_orig, test_df_orig
gc.collect()

print(f"\nCreated Hugging Face Datasets:")
print(f"  Train: {train_dataset_hf}")
print(f"  Validation: {val_dataset_hf}")
print(f"  Test (for tokenization):
{test_dataset_hf_for_tokenization}")
```

```
Generating 1 misspelled variant(s) per sentence for training...
```

{"model_id":"6447f0bebdb6494b9c63935406bcb9fd","version_major":2,"version_minor":0}

```
Generating 1 misspelled variant(s) per sentence for validation...
```

{"model_id":"3743872338864ccd92b20ece17182949","version_major":2,"version_minor":0}

```
Generating 1 misspelled variant(s) per sentence for test...
```

{"model_id":"d847197f239844c1b8ef1ce1433adc4e","version_major":2,"version_minor":0}

```
Number of generated misspelled/correct pairs:
  Train: 4987
  Validation: 497
  Test: 500

Sample generated training data (first 3 pairs):
  Correct   : After the very heavy '' Dragontown '' album Alice
decided to return to his roots , his place of birth , Detroit , where
he accidently joined in with a festival together with MC5 & Iggy and
the Stooges .
  Misspelled: After the very heavy '' Dragontown '' album Alice
decided to retun to hus roots , his place fo birth , Detroit , where
hu accidently joined inn with a festival together with YC5 & Igg and
```

```
the Stooges .
------------------------------
  Correct   : The museum runs a library with photographic books and
magazines , and a small museum store that sells postcards , posters
and more .
  Misspelled: The museum runs a libraryy with photographic baoks and
magazines , and a small museum store that sells postcard , posters and
more .
------------------------------
  Correct   : Jakobshavn Isbræ is a major contributor to the mass
balance of the Greenland ice sheet , producing some 10 % of all
Greenland icebergs some 35 billion tonnes of icebergs calved off and
passing out of the fjord every year .
  Misspelled: Jakohbshavn IIsbræ ys a majof contributor to the maass
balance of the Greenland ice sheet , proudcing some 10 % fo all
Greeland icebergs som 35 billion tonnes on icebergs calved fof and
passing out o te fjord every yyear .
------------------------------

Created Hugging Face Datasets:
  Train: Dataset({
    features: ['misspelled', 'correct'],
    num_rows: 4987
})
  Validation: Dataset({
    features: ['misspelled', 'correct'],
    num_rows: 497
})
  Test (for tokenization): Dataset({
    features: ['misspelled', 'correct'],
    num_rows: 500
})

# --- 4. Perform Necessary Data Preparation and Preprocessing ---

# Initialize tokenizer
tokenizer = T5Tokenizer.from_pretrained(MODEL_NAME,
cache_dir=CACHE_DIR)
print(f"Tokenizer loaded: {MODEL_NAME}")
print(f"  Vocabulary size: {tokenizer.vocab_size}")
print(f"  Pad token: '{tokenizer.pad_token}' (ID:
{tokenizer.pad_token_id})")
print(f"  EOS token: '{tokenizer.eos_token}' (ID:
{tokenizer.eos_token_id})")


def preprocess_function(examples):
    """Tokenizes misspelled inputs and correct target sentences for
T5."""
    # For T5, a task-specific prefix is often beneficial.
    inputs = ["correct spelling: " + doc for doc in
examples["misspelled"]]

    # Tokenize the inputs (misspelled sentences)
    model_inputs = tokenizer(
        inputs,
        max_length=MAX_SOURCE_LENGTH,
        padding="max_length", # Pad to max_length
        truncation=True       # Truncate if longer than max_length
    )

    # Tokenize the targets (correct sentences)
    # The 'with tokenizer.as_target_tokenizer():' context manager
ensures
    # that the tokenizer prepares the labels suitable for T5's
decoder.
    with tokenizer.as_target_tokenizer():
        labels = tokenizer(
            examples["correct"],
            max_length=MAX_TARGET_LENGTH,
            padding="max_length", # Pad to max_length
            truncation=True       # Truncate if longer
        )

    # The model expects the target token IDs in the 'labels' field.
    model_inputs["labels"] = labels["input_ids"]
    return model_inputs

print("\nTokenizing datasets...")
if len(train_dataset_hf) > 0:
    tokenized_train_dataset = train_dataset_hf.map(
        preprocess_function,
        batched=True,
        remove_columns=["misspelled", "correct"],
        desc="Preprocessing train dataset"
    )
    print(f"Tokenized train dataset: {tokenized_train_dataset}")
else:
    tokenized_train_dataset = train_dataset_hf # Keep as empty if
source was empty
    print("Skipping tokenization for empty train dataset.")

if len(val_dataset_hf) > 0:
    tokenized_val_dataset = val_dataset_hf.map(
        preprocess_function,
        batched=True,
        remove_columns=["misspelled", "correct"],
        desc="Preprocessing validation dataset"
    )
    print(f"Tokenized validation dataset: {tokenized_val_dataset}")
```

```
else:
    tokenized_val_dataset = val_dataset_hf
    print("Skipping tokenization for empty validation dataset.")

if len(test_dataset_hf_for_tokenization) > 0:
    tokenized_test_dataset_for_eval =
test_dataset_hf_for_tokenization.map(
        preprocess_function,
        batched=True,
        remove_columns=["misspelled", "correct"],
        desc="Preprocessing test dataset for evaluation"
    )
    print(f"Tokenized test dataset (for eval):
{tokenized_test_dataset_for_eval}")
else:
    tokenized_test_dataset_for_eval = test_dataset_hf_for_tokenization
    print("Skipping tokenization for empty test dataset.")

# Verify an example from the tokenized training set
if len(tokenized_train_dataset) > 0:
    print("\nExample of a tokenized training sample:")
    sample = tokenized_train_dataset[0]
    print(f"  Input IDs: {sample['input_ids'][:20]}... (len:
{len(sample['input_ids'])})")
    print(f"  Decoded Input: {tokenizer.decode(sample['input_ids'],
skip_special_tokens=False)}")
    print(f"  Labels: {sample['labels'][:20]}... (len:
{len(sample['labels'])})")
    print(f"  Decoded Labels: {tokenizer.decode(sample['labels'],
skip_special_tokens=False)}")
```

```
Tokenizer loaded: t5-small
  Vocabulary size: 32000
  Pad token: '<pad>' (ID: 0)
  EOS token: '</s>' (ID: 1)

Tokenizing datasets...
```

{"model_id":"a13dc5bd1644494fa588e129fb637cba","version_major":2,"version_minor":0}

```
/usr/local/lib/python3.11/dist-packages/transformers/
tokenization_utils_base.py:3980: UserWarning: `as_target_tokenizer` is
deprecated and will be removed in v5 of Transformers. You can tokenize
your labels by using the argument `text_target` of the regular
`__call__` method (either in the same call as your input texts if you
use the same keyword arguments, or in a separate call.
  warnings.warn(
Tokenized train dataset: Dataset({
    features: ['input_ids', 'attention_mask', 'labels'],
    num_rows: 4987
})
```

{"model_id":"ac59963bc1b64ae3bc910a9f615a1106","version_major":2,"version_minor":0}

```
Tokenized validation dataset: Dataset({
    features: ['input_ids', 'attention_mask', 'labels'],
    num_rows: 497
})
```

{"model_id":"55ae807740eb4b599810ceecb44665ee","version_major":2,"version_minor":0}

```
Tokenized test dataset (for eval): Dataset({
    features: ['input_ids', 'attention_mask', 'labels'],
    num_rows: 500
})
```

```
Example of a tokenized training sample:
  Input IDs: [2024, 19590, 10, 621, 8, 182, 2437, 3, 31, 31, 10282, 3540, 3, 31, 31, 2306, 13390, 1500, 12, 3]... (len: 128)
  Decoded Input: correct spelling: After the very heavy '' Dragontown '' album Alice decided to retun to hus roots, his place fo birth, Detroit, where hu accidently joined inn with a festival together with YC5 & Igg and the Stooges.</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>
  Labels: [621, 8, 182, 2437, 3, 31, 31, 10282, 3540, 3, 31, 31, 2306, 13390, 1500, 12, 1205, 12, 112, 8523]... (len: 128)
  Decoded Labels: After the very heavy '' Dragontown '' album Alice decided to return to his roots, his place of birth, Detroit, where he accidently joined in with a festival together with MC5 & Iggy and the Stooges.</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>
```

```
# --- 5. Choose Transformer (T5), Prepare for Fine-tuning ---
# --- 6. Methods for Overfitting (Callbacks, Weight Decay) ---
# --- 7. Evaluation Metrics Definition ---
# Define Evaluation Metrics (WER and CER)
wer_metric = evaluate.load("wer")
cer_metric = evaluate.load("cer")
print("WER and CER metrics loaded from 'evaluate' library.")

# In Cell 6 (Model Loading, Metrics, and Training Arguments)
# Modify your compute_metrics function

def compute_metrics(eval_preds):
    """Computes WER, CER, and Exact Match Accuracy for evaluation."""
    preds_outputs, labels_ids = eval_preds # eval_preds is a tuple
(predictions, labels)

    if isinstance(preds_outputs, tuple):
        preds_ids = preds_outputs[0]
    else:
        preds_ids = preds_outputs

    # --- TEMP DEBUGGING: PRINT RAW PREDICTED TOKEN IDs ---
    print(f"\n[DEBUG compute_metrics] Raw preds_ids (first example, up
to 50 tokens): {preds_ids[0, :50] if preds_ids.ndim > 1 and
preds_ids.shape[0] > 0 else preds_ids[:50]}")
    print(f"[DEBUG compute_metrics] Shape of preds_ids:
{preds_ids.shape}")
    print(f"[DEBUG compute_metrics] Min ID in preds:
{np.min(preds_ids)}, Max ID in preds: {np.max(preds_ids)}")
    print(f"[DEBUG compute_metrics] Tokenizer vocab size:
{tokenizer.vocab_size}")
    # --- END TEMP DEBUGGING ---

    # It's possible preds_ids are already a numpy array, but ensure it
for np.where
    if isinstance(preds_ids, torch.Tensor):
        preds_ids_np = preds_ids.cpu().numpy()
    else:
        preds_ids_np = np.array(preds_ids)


    # Filter out any potential out-of-bounds IDs *before* decoding as
a safeguard,
    # though the root cause needs to be fixed.
    # This is a temporary workaround to PREVENT the crash for
diagnosis, not a fix.
    vocab_size = tokenizer.vocab_size
    preds_ids_np_clipped = np.clip(preds_ids_np, 0, vocab_size - 1)


    labels_ids_np = np.array(labels_ids) # Ensure labels_ids is also a
numpy array
```

```
    labels_ids_np = np.where(labels_ids_np != -100, labels_ids_np,
tokenizer.pad_token_id)

    # Use the clipped IDs for decoding
    try:
        # decoded_preds = tokenizer.batch_decode(preds_ids,
skip_special_tokens=True) # Original
        decoded_preds = tokenizer.batch_decode(preds_ids_np_clipped,
skip_special_tokens=True) # Using clipped
        decoded_labels = tokenizer.batch_decode(labels_ids_np,
skip_special_tokens=True)
    except IndexError as e:
        print(f"IndexError during batch_decode even after clipping:
{e}")
        print(f"Problematic preds_ids sample (clipped):
{preds_ids_np_clipped[0, :50] if preds_ids_np_clipped.ndim > 1 and
preds_ids_np_clipped.shape[0] > 0 else preds_ids_np_clipped[:50]}")
        # Fallback if decoding still fails
        decoded_preds = [" [DECODING_ERROR] " for _ in
range(preds_ids_np_clipped.shape[0])]
        decoded_labels = tokenizer.batch_decode(labels_ids_np,
skip_special_tokens=True)


    decoded_preds = [pred.strip() for pred in decoded_preds]
    decoded_labels = [label.strip() for label in decoded_labels]

    # ... (rest of your metric calculations: wer, cer, accuracy) ...
    try:
        wer = wer_metric.compute(predictions=decoded_preds,
references=decoded_labels)
        wer_score = wer['wer'] if isinstance(wer, dict) and 'wer' in
wer else wer
    except Exception as e:
        print(f"Warning: Could not compute WER: {e}. Returning high
value.")
        wer_score = 1.0

    try:
        cer = cer_metric.compute(predictions=decoded_preds,
references=decoded_labels)
        cer_score = cer['cer'] if isinstance(cer, dict) and 'cer' in
cer else cer
    except Exception as e:
        print(f"Warning: Could not compute CER: {e}. Returning high
value.")
        cer_score = 1.0

    exact_matches = sum(1 for pred, label in zip(decoded_preds,
decoded_labels) if pred == label)
    accuracy = exact_matches / len(decoded_preds) if
len(decoded_preds) > 0 else 0.0

    return {
        "wer": wer_score if wer_score is not None else 1.0,
        "cer": cer_score if cer_score is not None else 1.0,
        "exact_match_accuracy": accuracy
    }

# Training Arguments

effective_batch_size = 10
per_device_train_bs = 4
per_device_eval_bs = per_device_train_bs * 2

if per_device_train_bs == 0:
    per_device_train_bs = 1
gradient_accumulation_steps = max(1, effective_batch_size //
per_device_train_bs)

print(f"\nTraining Configuration:")
print(f"  Per-device train batch size: {per_device_train_bs}")
print(f"  Per-device eval batch size: {per_device_eval_bs}")
print(f"  Gradient accumulation steps: {gradient_accumulation_steps}")
print(f"  Effective train batch size: {per_device_train_bs *
gradient_accumulation_steps}")

# Ensure Seq2SeqTrainingArguments is imported
from transformers import Seq2SeqTrainingArguments
import os # for os.path.join

training_args = Seq2SeqTrainingArguments(
    output_dir=OUTPUT_DIR,
    eval_strategy="steps",
    eval_steps=250,
    save_strategy="steps",
    save_steps=250,
    logging_dir=os.path.join(OUTPUT_DIR, "logs"),
    logging_steps=50,
    learning_rate=5e-5,
    per_device_train_batch_size=per_device_train_bs,
    per_device_eval_batch_size=per_device_eval_bs,
    gradient_accumulation_steps=gradient_accumulation_steps,
    weight_decay=0.01,
    save_total_limit=2,
    num_train_epochs=2,
    predict_with_generate=True,
    generation_max_length=MAX_TARGET_LENGTH,
    fp16=torch.cuda.is_available(),
    load_best_model_at_end=True,
    metric_for_best_model="cer",
    greater_is_better=False,
    report_to="none",
    dataloader_num_workers=0,
)

# Data Collator
# Ensure DataCollatorForSeq2Seq is imported
from transformers import DataCollatorForSeq2Seq

data_collator = DataCollatorForSeq2Seq(
    tokenizer=tokenizer,
    model=model,
    label_pad_token_id=-100,
    pad_to_multiple_of=8 if training_args.fp16 else None
)
print("TrainingArguments and DataCollator initialized.")
```

```
WER and CER metrics loaded from 'evaluate' library.

Training Configuration:
  Per-device train batch size: 4
  Per-device eval batch size: 8
  Gradient accumulation steps: 2
  Effective train batch size: 8
TrainingArguments and DataCollator initialized.
```

```
# --- Initialize Trainer ---
trainer = Seq2SeqTrainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train_dataset if
len(tokenized_train_dataset) > 0 else None,
    eval_dataset=tokenized_val_dataset if len(tokenized_val_dataset) >
0 else None,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
    callbacks=[EarlyStoppingCallback(early_stopping_patience=3)] #
Stop if metric doesn't improve for 3 evals
)
print("Seq2SeqTrainer initialized.")

# --- Start Training (Requirement 5 cont.) ---
print("\nStarting model fine-tuning...")
# Check if datasets are valid for training
can_train = False
if tokenized_train_dataset and len(tokenized_train_dataset) > 0:
    if tokenized_val_dataset and len(tokenized_val_dataset) > 0:
        can_train = True
        print(f"Training with {len(tokenized_train_dataset)} train
samples and {len(tokenized_val_dataset)} validation samples.")
    else:
        print("Warning: Validation dataset is empty. Training without
validation is not recommended for finding the best model.")
else:
    print("Training dataset is empty. Skipping training.")

if can_train:
    try:
        print("Calling trainer.train()...")
        train_result = trainer.train()
        print("Training finished successfully.")

        # Save training metrics
        trainer.log_metrics("train", train_result.metrics)
        trainer.save_metrics("train", train_result.metrics)
        trainer.save_state() # Saves optimizer, scheduler, etc.

        # Save the best model explicitly (though
load_best_model_at_end should handle it)
        best_model_path = os.path.join(OUTPUT_DIR,
"best_spell_corrector_model")
        trainer.save_model(best_model_path)
        tokenizer.save_pretrained(best_model_path) # Save tokenizer
with the model
        print(f"Best model and tokenizer saved to {best_model_path}")

    except Exception as e:
        print(f"An error occurred during training: {e}")
        import traceback
        traceback.print_exc()
else:
    if not (tokenized_train_dataset and len(tokenized_train_dataset) >
0) :
        print("Skipping training because the training dataset is empty
or invalid.")
    elif not (tokenized_val_dataset and len(tokenized_val_dataset) >
0) :
         print("Skipping training because the validation dataset is
empty and training_args expect validation.")


# Clean up GPU memory after training
gc.collect()
if torch.cuda.is_available():
    torch.cuda.empty_cache()
    print("GPU memory cache cleared after training.")
```

```
/tmp/ipykernel_35/1036528892.py:2: FutureWarning: `tokenizer` is
deprecated and will be removed in version 5.0.0 for
`Seq2SeqTrainer.__init__`. Use `processing_class` instead.
  trainer = Seq2SeqTrainer(

Seq2SeqTrainer initialized.

Starting model fine-tuning...
Training with 4987 train samples and 497 validation samples.
Calling trainer.train()...

<IPython.core.display.HTML object>


[DEBUG compute_metrics] Raw preds_ids (first example, up to 50
tokens): [    0     3    15   566     3  8637  5973    44     8 20052
1121    13
     8   868 10639    13 12263    41   262   189 24362   157    32
37
    32   144    52    32     3    61    11   263   112     3     7
2408
    63  5695    16 25745     3     6 16069    38     8 11595    16
4599
   451  1639]
[DEBUG compute_metrics] Shape of preds_ids: (497, 128)
[DEBUG compute_metrics] Min ID in preds: -100, Max ID in preds: 31968
[DEBUG compute_metrics] Tokenizer vocab size: 32000

[DEBUG compute_metrics] Raw preds_ids (first example, up to 50
tokens): [    0   454     3  8637  5973    44     8 20052  1121    13
8   868
 10639    13 12263    41   262   189 24362   157    32    37    32
144
    52    32     3    61    11   263   112  1415  5695    16 25745
3
     6 16069    38     8 11595    16  4599   451  1639     7   855
355
     3    31]
[DEBUG compute_metrics] Shape of preds_ids: (497, 97)
[DEBUG compute_metrics] Min ID in preds: -100, Max ID in preds: 31968
[DEBUG compute_metrics] Tokenizer vocab size: 32000

[DEBUG compute_metrics] Raw preds_ids (first example, up to 50
tokens): [    0   216     3  8637  5973    44     8 20052  1121    13
8   868
 10639    13 12263    41   262   189 24362   157    32    37    32
144
    52    32     3    61    11   263   112  1415  5695    16 25745
3
     6 16069    38     8 11595    16  4599   451  1639     7   855
355
     3    31]
[DEBUG compute_metrics] Shape of preds_ids: (497, 95)
[DEBUG compute_metrics] Min ID in preds: -100, Max ID in preds: 31968
[DEBUG compute_metrics] Tokenizer vocab size: 32000

[DEBUG compute_metrics] Raw preds_ids (first example, up to 50
tokens): [    0   216     3  8637  5973    44     8 20052  1121    13
8   868
 10639    13 12263    41   262   189 24362   157    32    37    32
144
    52    32     3    61    11   263   112  1415  5695    16 25745
3
     6 16069    38     8 11595    16  4599   451  1639     7   855
355
     3    31]
[DEBUG compute_metrics] Shape of preds_ids: (497, 96)
[DEBUG compute_metrics] Min ID in preds: -100, Max ID in preds: 31968
[DEBUG compute_metrics] Tokenizer vocab size: 32000

There were missing keys in the checkpoint model loaded:
['encoder.embed_tokens.weight', 'decoder.embed_tokens.weight',
'lm_head.weight'].

Training finished successfully.
***** train metrics *****
  epoch                    =     1.9976
  total_flos               =   313951GF
  train_loss               =     0.2168
  train_runtime            = 2:14:32.63
  train_samples_per_second =      1.236
  train_steps_per_second   =      0.154
Best model and tokenizer saved to
/kaggle/working/spell_correction_outputs/best_spell_corrector_model
```

```
# --- 7. Evaluate the Performance on Test Dataset ---
if 'can_train' in locals() and can_train and 'best_model_path' in
locals() : # Check if training was attempted and successful
    print(f"\nEvaluating the fine-tuned model on the test set...")
    if tokenized_test_dataset_for_eval and
len(tokenized_test_dataset_for_eval) > 0:
        try:
            test_results = trainer.evaluate(
                eval_dataset=tokenized_test_dataset_for_eval,
                metric_key_prefix="test" # Adds "test_" prefix to
metric names
            )
            print("\nTest Set Evaluation Results:")
            for key, value in test_results.items():
                # Ensure value is float for consistent formatting,
handle potential None
                value_to_print = value if isinstance(value, (int,
float)) else 0.0
                print(f"  {key}: {value_to_print:.4f}")

            # Save test metrics
            trainer.log_metrics("test", test_results)
            trainer.save_metrics("test", test_results)
        except Exception as e:
            print(f"An error occurred during test set evaluation:
{e}")
            import traceback
            traceback.print_exc()
    else:
        print("Test dataset (tokenized_test_dataset_for_eval) is
empty. Skipping test set evaluation.")
else:
    print("\nSkipping test set evaluation as model training was not
performed or did not complete successfully.")
```

```
Evaluating the fine-tuned model on the test set...
```

```
<IPython.core.display.HTML object>
```

```
[DEBUG compute_metrics] Raw preds_ids (first example, up to 50
tokens): [    0   216 17785    12 20134     7  3763    16  2038    11
1632 12130
 17418    21     8 25491     7    16  1421     3     5     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]
[DEBUG compute_metrics] Shape of preds_ids: (500, 107)
[DEBUG compute_metrics] Min ID in preds: -100, Max ID in preds: 31993
[DEBUG compute_metrics] Tokenizer vocab size: 32000
```

```
early stopping required metric_for_best_model, but did not find
eval_cer so early stopping is disabled
```

```
Test Set Evaluation Results:
  test_loss: 0.1753
  test_wer: 0.1396
  test_cer: 0.0643
  test_exact_match_accuracy: 0.0240
  test_runtime: 330.4428
  test_samples_per_second: 1.5130
  test_steps_per_second: 0.1910
  epoch: 1.9976
***** test metrics *****
  epoch                      =     1.9976
  test_cer                   =     0.0643
  test_exact_match_accuracy =      0.024
  test_loss                  =     0.1753
  test_runtime               = 0:05:30.44
  test_samples_per_second    =      1.513
  test_steps_per_second      =      0.191
  test_wer                   =     0.1396

# --- Inference Function & Demonstration ---

# Determine which model and tokenizer to use for inference
model_for_inference = None
tokenizer_for_inference = None
best_model_path_to_load = os.path.join(OUTPUT_DIR,
"best_spell_corrector_model") # Path where model was saved

if 'can_train' in locals() and can_train and
os.path.exists(best_model_path_to_load):
    print(f"\nLoading best fine-tuned model from
{best_model_path_to_load} for inference...")
    try:
        model_for_inference =
T5ForConditionalGeneration.from_pretrained(best_model_path_to_load)
        tokenizer_for_inference =
T5Tokenizer.from_pretrained(best_model_path_to_load)
        print("Successfully loaded fine-tuned model and tokenizer.")
    except Exception as e:
        print(f"Error loading fine-tuned model from
{best_model_path_to_load}: {e}. Falling back to base model.")
        model_for_inference = None # Ensure fallback

if model_for_inference is None: # Fallback if fine-tuned model loading
failed or training was skipped
    print(f"\nFine-tuned model not available. Using base {MODEL_NAME}
for inference (results may be indicative only).")
    model_for_inference =
T5ForConditionalGeneration.from_pretrained(MODEL_NAME,
cache_dir=CACHE_DIR)
    tokenizer_for_inference = T5Tokenizer.from_pretrained(MODEL_NAME,
cache_dir=CACHE_DIR)

model_for_inference.to(device)
model_for_inference.eval() # Set to evaluation mode (disables dropout
etc.)
print(f"Model for inference is on device:
{model_for_inference.device}")
```

```
def correct_spelling_batch(text_list,
max_correction_len=MAX_TARGET_LENGTH):
    """Corrects spelling for a batch of input texts."""
    if not isinstance(text_list, list):
        text_list = [str(text_list)] # Ensure it's a list of strings
    else:
        text_list = [str(text) for text in text_list]

    if not text_list:
        return []

    # Add T5 prefix
    prefixed_texts = ["correct spelling: " + text.strip().replace("\
n"," ") for text in text_list]

    # Tokenize
    inputs = tokenizer_for_inference(
        prefixed_texts,
        return_tensors="pt",    # PyTorch tensors
        padding=True,           # Pad to longest in batch
        truncation=True,        # Truncate if too long
        max_length=MAX_SOURCE_LENGTH
    ).to(device) # Move inputs to the same device as the model

    # Generate corrected sequences
    with torch.no_grad(): # Disable gradient calculations for
inference
        summary_ids = model_for_inference.generate(
            inputs['input_ids'],
            attention_mask=inputs['attention_mask'], # Pass attention
mask
            num_beams=4,                             # Beam search for
potentially better quality
            no_repeat_ngram_size=2,                  # Helps prevent
repetitive phrases
            min_length=max(1, int(MAX_TARGET_LENGTH * 0.1)), # Avoid
overly short trivial outputs
            max_length=max_correction_len,
            early_stopping=True                      # Stop when EOS
token is generated by all beams
        )

    # Decode generated token IDs back to text
    corrected_texts =
tokenizer_for_inference.batch_decode(summary_ids,
skip_special_tokens=True)
    return corrected_texts
```

```
# --- Demonstration using examples from the original test_data_df ---
print("\n--- Demonstrating Spelling Correction on Test Examples ---")

# test_data_df is the pandas DataFrame with 'misspelled' and 'correct'
columns created in Cell 4
if 'test_data_df' in locals() and not test_data_df.empty:
    # Take a few random samples for demonstration
    num_demo_samples = min(5, len(test_data_df))
    demo_samples_df = test_data_df.sample(n=num_demo_samples,
random_state=77) # Use a fixed seed for demo

    for _, row in demo_samples_df.iterrows():
        misspelled_text = row['misspelled']
        original_correct_text = row['correct']

        print(f"\nOriginal Misspelled : {misspelled_text}")
        print(f"Ground Truth Correct: {original_correct_text}")

        # Correct spelling (pass as a list, even if it's one sentence)
        corrected_output_list =
correct_spelling_batch([misspelled_text])

        if corrected_output_list:
            model_corrected_text = corrected_output_list[0]
            print(f"Model Corrected     : {model_corrected_text}")

            # Calculate edit distance (case-insensitive comparison)
            if original_correct_text and model_corrected_text:
                try:
                    dist =
edit_distance(model_corrected_text.lower().strip(),
original_correct_text.lower().strip())
                    print(f"Edit Distance (Model vs Truth, lower is
better): {dist}")
                except Exception as e_dist:
                    print(f"Could not calculate edit distance:
{e_dist}")
        else:
            print(f"Model Corrected     : [NO OUTPUT OR ERROR]")

else:
    print("Original test data (test_data_df) not available or empty.
Skipping demonstration on test examples.")

# Example of correcting a new, custom sentence
print("\n--- Correcting a New Custom Sentence ---")
custom_misspelled_sentence = "Ths is a sentance wth sme speling errrs
and I hope it gets fxed."
print(f"Input Misspelled: {custom_misspelled_sentence}")
```

```python
corrected_custom_list =
correct_spelling_batch([custom_misspelled_sentence])
if corrected_custom_list:
    print(f"Model Corrected : {corrected_custom_list[0]}")
else:
    print(f"Model Corrected : [NO OUTPUT OR ERROR]")
```

```
Loading best fine-tuned model from
/kaggle/working/spell_correction_outputs/best_spell_corrector_model
for inference...
Successfully loaded fine-tuned model and tokenizer.
Model for inference is on device: cpu

--- Demonstrating Spelling Correction on Test Examples ---

Original Misspelled : Produced by Warner Beros. Pictures , the film
premiered in New York Cityy on December 13 , 2016 amd had a limited
release oq Decembepr 25 , 2016 , and had a wide release on January
13 , 2017 .
Ground Truth Correct: Produced by Warner Bros. Pictures , the film
premiered in New York City on December 13 , 2016 and had a limited
release on December 25 , 2016 , and had a wide release on January 13 ,
2017 .
Model Corrected     : Produced by Warner Bros. Pictures, the film
premiered in New York City on December 13. 2016 and had a limited
release on Decembepr 25 - 2016 ; and was able to open on January 13
and 2017!
Edit Distance (Model vs Truth, lower is better): 24

Original Misspelled : Various elements and obstacles are introduced
aas one moves on to new levesl , which means that the complexity acd
level of puzzle solving rquired ogradually increases as they progress
throughh the gme .
Ground Truth Correct: Various elements and obstacles are introduced as
one moves on to new levels , which means that the complexity and level
of puzzle solving required gradually increases as they progress
through the game .
Model Corrected     : Various elements and obstacles are introduced as
one moves on to new areas, which means that the complexity of the
level of puzzle solving required gradually increases as they progress
through the process.
Edit Distance (Model vs Truth, lower is better): 17

Original Misspelled : Hoowever , thhe original ylag hazd a brownish
oclour instead of green , atnd there ae horizontal variants of the
flag aas well .
Ground Truth Correct: However , the original flag had a brownish
colour instead of green , and there are horizontal variants of the
```

```
flag as well .
Model Corrected     : Hoowever, the original ylag hazd a brownish
clour instead of green and there were horizontal variants of the flag
as well.
Edit Distance (Model vs Truth, lower is better): 10

Original Misspelled : Khurramm tries to return back to the camp from
tthe battle but takes longv tmie to reach as he forgets his way and
Mumatz Mahal diesy while giving birth to her nineteenth hild .
Ground Truth Correct: Khurram tries to return back to the camp from
the battle but takes long time to reach as he forgets his way and
Mumtaz Mahal dies while giving birth to her nineteenth child .
Model Corrected     : Khurramm tries to return back to the camp from
the battle but takes long to reach as he forgets his way and Mumatz
Mahal dies while giving birth to her nineteenth birthday.
Edit Distance (Model vs Truth, lower is better): 15

Original Misspelled : '' Bellringer '' was in fact a derivativqe fo ''
Hellbringer , '' a nickname giyen to him vy fellow musician Dan Massie
rn rfeerence to hhis unquenchable thirst for debaucheyr and outlandish
clsthing .
Ground Truth Correct: '' Bellringer '' was in fact a derivative of ''
Hellbringer , '' a nickname given to him by fellow musician Dan Massie
in reference to his unquenchable thirst for debauchery and outlandish
clothing .
Model Corrected     : "' Bellringer '' was in fact a derivativist for
" " Hellbringer, "' an nickname given to him by fellow musician Dan
Massie to his unquenchable thirst for debauchey and outlandishthing.
Edit Distance (Model vs Truth, lower is better): 31

--- Correcting a New Custom Sentence ---
Input Misspelled: Ths is a sentance wth sme speling errrs and I hope
it gets fxed.
Model Corrected : Ths is a sentance for the speling errors and I hope
it gets done.
```