```
# --- 1. Install Libraries ---
# We need 'transformers' for the models, 'datasets' for data handling,
# and 'sacrebleu' is a standard library for evaluating "translation" quality (which this is!)
!pip install transformers datasets sacrebleu torch

# --- 2. Import All Dependencies ---
import torch
import re
from datasets import load_dataset, Dataset
from transformers import (
    T5ForConditionalGeneration,
    T5Tokenizer,
    DataCollatorForSeq2Seq,
    Trainer,
    TrainingArguments
)
from google.colab import drive
```

Show hidden output

```
# --- Load Raw Data (Our "Target" Text) ---
print("Loading wikitext dataset...")
full_dataset = load_dataset("wikitext", "wikitext-103-v1", split="train")

# We'll use 50,000 examples as we discussed.
# You can lower this to ~5,000 for a very fast test run.
slice_size = 100000
dataset_slice = full_dataset.select(range(slice_size))

print(f"\nCreated a working slice of {len(dataset_slice)} documents.")
print(f"Example 'good' text: \n'{dataset_slice[5]['text']}'")
```

```
Loading wikitext dataset...

Created a working slice of 100000 documents.
Example 'good' text:
' It met with positive sales in Japan , and was praised by both Japanese and western critics
'
```

```
# --- Define our "Bad" Text -> "Good" Text Preprocessor ---

def create_seq2seq_examples(example):
    text = example['text'].strip()

    # 1. Filter out empty lines and headings
    if not text or text.startswith("=") or len(text.split()) < 5:
        return {"input_text": "", "target_text": ""}

    # 2. Define the Target (Y) - The "good" text
    # This is just the original, correct text.
    target_text = text

    # 3. Define the Input (X) - The "bad" text
    # We make it lowercase and remove all punctuation
    # We use a simple regex to keep only letters, numbers, and whitespace
    broken_text = re.sub(r'[^\w\s]', '', text.lower())

    # 4. Add the T5 Task Prefix
    # This prefix tells the model what "translation" task to perform.
    input_text = "correct: " + broken_text

    return {"input_text": input_text, "target_text": target_text}
```

```
# --- Apply the Function and Filter ---
print("Applying preprocessing to all examples...")

# Apply our function to every example in the dataset
# num_proc=4 uses 4 cores to speed this up.
raw_dataset = dataset_slice.map(
    create_seq2seq_examples,
    num_proc=4,
    remove_columns=['text']  # We don't need the original 'text' column anymore
)

# Filter out the empty examples we created
processed_dataset = raw_dataset.filter(lambda x: len(x['input_text']) > 10)

print(f"\nFinished processing. We have {len(processed_dataset)} valid examples.")
print("\nExample of a training pair:")
print(f"INPUT (X):  '{processed_dataset[5]['input_text']}'")
print(f"TARGET (Y): '{processed_dataset[5]['target_text']}'")
```

```
Applying preprocessing to all examples...
Map (num_proc=4): 100%                    100000/100000 [00:12<00:00, 3366.01 examples/s]

Filter: 100%                              100000/100000 [00:01<00:00, 67036.80 examples/s]


Finished processing. We have 46033 valid examples.

Example of a training pair:
INPUT (X):  'correct: troops are divided into five classes  scouts  unk  engineers  lancers a
TARGET (Y): 'Troops are divided into five classes : Scouts , <unk> , Engineers , Lancers and
```

```
# --- Load Model and Tokenizer ---
model_name = "t5-small"

print(f"Loading '{model_name}' tokenizer and model...")
tokenizer = T5Tokenizer.from_pretrained(model_name)
model = T5ForConditionalGeneration.from_pretrained(model_name)

# Check for GPU
device = "cuda" if torch.cuda.is_available() else "cpu"
model.to(device)
print(f"\nModel loaded and moved to {device}.")
```

```
Loading 't5-small' tokenizer and model...

Model loaded and moved to cuda.
```

```
# --- Define Tokenization Function ---
# We'll truncate sequences to 128 tokens.
# T5 is efficient, but this keeps training fast.
MAX_LENGTH = 128

def tokenize_function(examples):
    # Tokenize the "inputs" (our broken text)
    model_inputs = tokenizer(
        examples["input_text"],
        max_length=MAX_LENGTH,
        truncation=True
    )

    # Tokenize the "targets" (our correct text)
    # We use this 'with' block to ensure the tokenizer knows
    # it's tokenizing the "target" or "label" text.
    with tokenizer.as_target_tokenizer():
        labels = tokenizer(
            examples["target_text"],
            max_length=MAX_LENGTH,
            truncation=True
        )

    # Add the tokenized labels to our model inputs
    model_inputs["labels"] = labels["input_ids"]
    return model_inputs
```

```
# --- Apply Tokenization and Split ---
print("Tokenizing all examples...")

# Apply the tokenization function to all our examples
tokenized_dataset = processed_dataset.map(
    tokenize_function,
    batched=True,
    remove_columns=['input_text', 'target_text'] # Not needed anymore
)

# Split the dataset into 90% train, 10% validation
split_dataset = tokenized_dataset.train_test_split(test_size=0.1, seed=42)

train_dataset = split_dataset['train']
eval_dataset = split_dataset['test']

print("\nData is tokenized and split:")
print(f"Training examples:   {len(train_dataset)}")
print(f"Validation examples: {len(eval_dataset)}")
```

```
Tokenizing all examples...
Map: 100%                                    46033/46033 [01:22<00:00, 844.80 examples/s]
/usr/local/lib/python3.12/dist-packages/transformers/tokenization_utils_base.py:4034: UserWar
  warnings.warn(

Data is tokenized and split:
Training examples:   41429
Validation examples: 4604
```

```
# --- Mount Google Drive ---
print("Mounting Google Drive... Please authorize.")
drive.mount('/content/drive')
print("Google Drive mounted successfully.")
```

```
Mounting Google Drive... Please authorize.
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/c
Google Drive mounted successfully.
```

```
# --- Set Up Trainer (Corrected) ---

# 1. Import the correct Seq2Seq classes
from transformers import (
    Seq2SeqTrainer,
    Seq2SeqTrainingArguments,
    DataCollatorForSeq2Seq
)



# 3. Define the directory in your Google Drive to save the model
output_dir = "/content/drive/MyDrive/t5-punctuation-model-100k"

# 4. This special collator correctly pads both inputs and labels
data_collator = DataCollatorForSeq2Seq(tokenizer=tokenizer, model=model)



# 5. *** Use Seq2SeqTrainingArguments ***
# This class IS designed for T5 and DOES accept 'predict_with_generate'
training_args = Seq2SeqTrainingArguments(
    output_dir=output_dir,
    num_train_epochs=6,                     # 3 epochs is a good start
    per_device_train_batch_size=8,          # 8 is safe for 't5-small' on a T4 GPU
    per_device_eval_batch_size=8,
    weight_decay=0.01,                      # Adds regularization

    # Evaluation and Saving
    eval_strategy="epoch",                # Run validation every epoch
    save_strategy="epoch",                # Save a checkpoint every epoch
    load_best_model_at_end=True,          # Keep only the best model

    # This is the critical argument, and it works with this class
    predict_with_generate=True,

    report_to="none"                        # Disables online logging
)

# 6. *** Use Seq2SeqTrainer ***
trainer = Seq2SeqTrainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    tokenizer=tokenizer,
    data_collator=data_collator,
)

print("\nTrainer initialized successfully with Seq2SeqTrainer.")
```

```
Trainer initialized successfully with Seq2SeqTrainer.
/tmp/ipython-input-2027641955.py:39: FutureWarning: `tokenizer` is deprecated and will be rem
  trainer = Seq2SeqTrainer(
```

```
# --- Train the Model! ---
print("Starting training...")
trainer.train()

print("\nTraining complete!")
print(f"The best model has been saved to: {output_dir}")
```

```
Starting training...
```

[31074/31074 1:20:13, Epoch 6/6]

Epoch	Training Loss	Validation Loss
1	0.410300	0.323803
2	0.359500	0.288531
3	0.325600	0.266852
4	0.305900	0.256341
5	0.294600	0.250900
6	0.288800	0.250409

```
There were missing keys in the checkpoint model loaded: ['encoder.embed_tokens.weight', 'deco

Training complete!
The best model has been saved to: /content/drive/MyDrive/t5-punctuation-model-100k
```

```
local_save_path = "./my-local-t5-model"
trainer.save_model(local_save_path)
print(f"A temporary local copy has also been saved to: {local_save_path}")
```

```
A temporary local copy has also been saved to: ./my-local-t5-model
```

```
# --- Test Your Trained Model ---
from transformers import T5ForConditionalGeneration, T5Tokenizer

# 1. Load your saved model from Google Drive
# The Trainer saves the best model in the 'output_dir'
model_path = "/content/drive/MyDrive/t5-punctuation-model-100k/checkpoint-31074"
model = T5ForConditionalGeneration.from_pretrained(model_path)
tokenizer = T5Tokenizer.from_pretrained(model_path)

# 2. Make sure model is on the GPU
device = "cuda" if torch.cuda.is_available() else "cpu"
model.to(device)
model.eval()
print("Loaded fine-tuned model from Google Drive.")

# 3. Create the prediction function
def correct(text):
    # Add the "correct:" prefix, lowercase, and remove punctuation
    input_text = "correct: " + re.sub(r'[^\w\s]', '', text.lower())

    # Tokenize the input
    inputs = tokenizer(
        input_text,
        return_tensors="pt",
        max_length=128,
        truncation=True
    ).to(device)

    # 4. Generate the corrected text
    with torch.no_grad():
        outputs = model.generate(
            inputs["input_ids"],
            max_length=128,  # Generate up to 128 tokens
            num_beams=4,       # Use beam search for better results
            early_stopping=True
        )

    # 5. Decode the output and return it
    return tokenizer.decode(outputs[0], skip_special_tokens=True)

# --- Let's try it! ---
print("\n" + "="*30)
print("--- TESTING THE NEW S2S MODEL ---")
print("="*30 + "\n")

text_1 = "hello my name is shahaan what is yours"
text_2 = "this is a test of the punctuation model i hope it works"
text_3 = "the game was fun but i think it could be better"

# Test 1
print(f"Input:    '{text_1}'")
print(f"Output:   '{correct(text_1)}'")
print("-" * 20)

# Test 2
print(f"Input:    '{text_2}'")
print(f"Output:   '{correct(text_2)}'")
print("-" * 20)

# Test 3
print(f"Input:    '{text_3}'")
print(f"Output:   '{correct(text_3)}'")
print("-" * 20)
```

```
Loaded fine-tuned model from Google Drive.

==============================
--- TESTING THE NEW S2S MODEL ---
==============================

Input:    'hello my name is shahaan what is yours'
Output:   'Hello, my name is Shahaan, what is yours?'
--------------------
Input:    'this is a test of the punctuation model i hope it works'
Output:   'This is a test of the punctuation model. I hope it works.'
--------------------
Input:    'the game was fun but i think it could be better'
Output:   'The game was fun, but I think it could be better.'
```

```
--------------------
```

```
test_string = "the quick brown fox jumps over the lazy dog this is a classic sentence used f

# Now you can run your function
corrected_version = correct(test_string)

print(f"Input:    '{test_string}'")
print(f"Output:   '{corrected_version}'")
```

```
Input:    'the quick brown fox jumps over the lazy dog this is a classic sentence used for ty
Output:   'The quick brown fox jumps over the lazy dog. This is a classic sentence used for t
```