```
import zipfile
import os

zip_path = "/content/handwritten500.zip"
extract_path = "/content/handwritten500"

# Ensure clean extraction directory
if not os.path.exists(extract_path):
    os.makedirs(extract_path)

# Extract
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print("✅ Successfully unzipped to:", extract_path)
```

```
✅ Successfully unzipped to: /content/handwritten500
```

```
import os

for root, dirs, files in os.walk(extract_path):
    print(f"{root} - {len(files)} files")
    for f in files[:5]:  # Show first 5 files in each folder
        print(f"  - {f}")
```

```
/content/handwritten500 - 0 files
/content/handwritten500/handwritten500 - 0 files
/content/handwritten500/handwritten500/english_pages - 60 files
  - a03-071.png
  - a05-108.png
  - a01-007u.png
  - a01-072u.png
  - a01-003.png
/content/handwritten500/handwritten500/english_pages/ground_truth - 60 files
  - a05-053.txt
  - a01-122u.txt
  - a01-003u.txt
  - a05-089.txt
  - a01-102u.txt
/content/handwritten500/handwritten500/hindi_synthetic - 164 files
  - 02-p1-93.png
  - 02-p2-9.png
  - 01_p2.jpg
  - 02-p1-90.png
  - 02-p1-7.png
/content/handwritten500/handwritten500/hindi_synthetic/ground_truth - 164 files
  - 02_p7.txt
  - 02-p2-10.txt
  - 02-p1-40.txt
  - 02-p2-37.txt
  - 02-p2-52.txt
/content/handwritten500/handwritten500/english_lines - 270 files
  - selfMade_177.png
  - selfMade_180.png
  - selfMade_44.png
  - selfMade_276.png
  - selfMade_138.png
/content/handwritten500/handwritten500/english_lines/ground_truth - 270 files
  - selfMade_176.txt
  - selfMade_3.txt
  - selfMade_157.txt
  - selfMade_210.txt
  - selfMade_256.txt
```

Prediction of Handwritten Text Image Using TrOCR

```
# ✅ Step 1: Install Required Libraries (if not done yet)
!pip install -U -q transformers torch torchvision torchaudio

# ✅ Step 2: Imports
import os
import glob
from PIL import Image
```

```
import torch
from transformers import TrOCRProcessor, VisionEncoderDecoderModel

# ✅ Step 3: Define Paths
image_dir = "/content/handwritten500/handwritten500/english_lines"

# ✅ Step 4: Load TrOCR Model
processor = TrOCRProcessor.from_pretrained("microsoft/trocr-base-handwritten")
model = VisionEncoderDecoderModel.from_pretrained("microsoft/trocr-base-handwritten")
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# ✅ Step 5: Prediction Function
def predict_text(image_path):
    image = Image.open(image_path).convert("RGB")
    inputs = processor(images=image, return_tensors="pt").to(device)
    generated_ids = model.generate(inputs.pixel_values)
    pred_text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
    return pred_text.strip()

# ✅ Step 6: Get First 5 Image Paths
image_paths = sorted(glob.glob(os.path.join(image_dir, "*.png")))[:5]

# ✅ Step 7: Predict and Print Text for Each Image
print("🖼️  OCR Predictions on First 5 English Handwritten Line Images:\n")
for path in image_paths:
    pred = predict_text(path)
    print(f"{os.path.basename(path)} → {pred}")
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 40.9/40.9 kB 1.8 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.8/10.8 MB 43.7 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 821.2/821.2 MB 747.6 kB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 393.1/393.1 MB 4.5 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.9/8.9 MB 101.0 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 23.7/23.7 MB 72.8 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 897.7/897.7 kB 46.3 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 571.0/571.0 MB 895.0 kB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 200.2/200.2 MB 2.5 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 47.3 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 14.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 158.2/158.2 MB 8.2 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 216.6/216.6 MB 1.3 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 156.8/156.8 MB 6.4 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 201.3/201.3 MB 6.8 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 19.7/19.7 MB 44.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 89.3/89.3 kB 6.2 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 155.7/155.7 MB 8.0 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.5/7.5 MB 82.8 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.5/3.5 MB 82.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.3/6.3 MB 91.6 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the sc
fastai 2.7.19 requires torch<2.7,>=1.10, but you have torch 2.7.1 which is incompatible.
/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
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(
preprocessor_config.json: 100%  224/224 [00:00<00:00, 14.0kB/s]
Using a slow image processor as `use_fast` is unset and a slow processor was saved with this model. `use_fast=True` will be the def
tokenizer_config.json:  1.12k/? [00:00<00:00, 98.6kB/s]
vocab.json:  899k/? [00:00<00:00, 21.7MB/s]
merges.txt:  456k/? [00:00<00:00, 25.8MB/s]
special_tokens_map.json: 100%  772/772 [00:00<00:00, 84.2kB/s]
config.json:  4.17k/? [00:00<00:00, 389kB/s]
model.safetensors: 100%  1.33G/1.33G [00:59<00:00, 51.6MB/s]
Some weights of VisionEncoderDecoderModel were not initialized from the model checkpoint at microsoft/trocr-base-handwritten and ar
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
generation_config.json: 100%  190/190 [00:00<00:00, 16.8kB/s]
🖼️ OCR Predictions on First 5 English Handwritten Line Images:

selfMade_0.png → Machine learning is on idea to learn from examples .
selfMade_1.png → and experience , without being explicitly programmed .
selfMade_10.png → Machine learning is a field which is raised out .
selfMade_100.png → when be applied to understructured data , giving
selfMade_102.png → access to much more input than machine-learning
```

```
# ✅ Step 1: Install Required Libraries (if not already installed)
!pip install -U -q transformers torch torchvision torchaudio

# ✅ Step 2: Imports
import os
import glob
from PIL import Image
import torch
from transformers import TrOCRProcessor, VisionEncoderDecoderModel

# ✅ Step 3: Define Paths
image_dir = "/content/handwritten500/handwritten500/english_lines"

# ✅ Step 4: Load TrOCR Model and Processor
processor = TrOCRProcessor.from_pretrained("microsoft/trocr-base-handwritten")
model = VisionEncoderDecoderModel.from_pretrained("microsoft/trocr-base-handwritten")

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# ✅ Step 5: Prediction Function
def predict_text(image_path):
    try:
        image = Image.open(image_path).convert("RGB")
        inputs = processor(images=image, return_tensors="pt").to(device)
```

```
        generated_ids = model.generate(inputs.pixel_values)
        pred_text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
        return pred_text.strip()
    except Exception as e:
        return f"[ERROR] {e}"

# ✅ Step 6: Get First 100 Image Paths
image_paths = sorted(glob.glob(os.path.join(image_dir, "*.png")))[:100]

# ✅ Step 7: Predict and Print Text for Each Image
print("🔍 OCR Predictions on First 100 English Handwritten Line Images:\n")
for idx, path in enumerate(image_paths, 1):
    pred = predict_text(path)
    print(f"{idx:03d}. {os.path.basename(path)} → {pred}")
```

```
044. selfMade_140.png → without being explicitly programmed . It is an
045. selfMade_141.png → application of AT that provide system the
046. selfMade_142.png → ability to automatically learn and improve .
047. selfMade_143.png → trans experience .
048. selfMade_145.png → Machine learning are the Appointment that passed data , learn from
049. selfMade_146.png → that dalaxand this apply that they've learned to make
050. selfMade_147.png → informed decisions . In addition example of a machine learning
051. selfMade_148.png → algorithm is an on-demand music stracing service . For the
052. selfMade_149.png → service to make a decision about which " now songs or .
053. selfMade_15.png → were unable to program more complex and constantly ... .
054. selfMade_150.png → attacks to recommend to a listener , machine learning algorithms
055. selfMade_151.png → associate the Victoria's preferences with other listeners who
056. selfMade_152.png → have similar musical taste .
057. selfMade_153.png → This sounds similar to child learning from
058. selfMade_154.png → its self . So machine learning was developed .
059. selfMade_155.png → as a new capability of computers . And now
060. selfMade_156.png → machine learning is present in so many segments .
061. selfMade_157.png → of technology , that we don't even " realise :
062. selfMade_158.png → it , while using it . Finding patterns in .
063. selfMade_159.png → data on planet earth is " possible only for
064. selfMade_16.png → evalving challenges . There was a realization that the #
065. selfMade_160.png → humans brains . The data being very massive
066. selfMade_165.png → Machine learning is an idea to learn from examples and
067. selfMade_166.png → " experience , without being explicitly programmed . Instead of
068. selfMade_167.png → writing code , you feed data to the generic algorithms ,
069. selfMade_168.png → and it builds logic based on the data given ...
070. selfMade_169.png → for example , one kind of algorithm is a classification
071. selfMade_170.png → algorithm . It can put data into different groups ... "
072. selfMade_171.png → The classification algorithm used to detect handwritten
073. selfMade_172.png → alphabets could also be used to classify emails .
074. selfMade_174.png → into spam and not-spam .
075. selfMade_176.png → Andrew No , the chief scientist of China major search engine Butler and
076. selfMade_177.png → one of the leaders of the benefit Brain Project , showed a great
077. selfMade_178.png → analogy for deep farming with wind Magazine ? " I think it ? "
078. selfMade_179.png → is ' akin ' to building a ' market shop , You need a huge engine .
079. selfMade_18.png → only way to be able to achieve this task to let ...
080. selfMade_180.png → and a lot of feet ! " The analogy to chap learning is . Most . "
081. selfMade_181.png → the rocket engine is the drop training models and the fuel ...
082. selfMade_182.png → is the huge amounts of data we can feel to those algorithms .
083. selfMade_185.png → Machine learning is an idea to learn from
084. selfMade_186.png → examples and experience , without being
085. selfMade_187.png → explicitly programmed . Instead of writing
086. selfMade_188.png → code , you feed data to the generic algorithm
087. selfMade_189.png → and it builds large based on the date .
088. selfMade_19.png → machine learn from itself .
089. selfMade_190.png → given . For example , one kind of similar
090. selfMade_196.2.png → " experience , without being explicitly programmed . Instead of
091. selfMade_196.png → Machine learning is an idea to learn from examples and
092. selfMade_197.png → is a classification algorithm .
093. selfMade_198.png → writing code , you feed data to the generic algorithms ,
094. selfMade_199.png → and it builds logic based on the data given ...
095. selfMade_2.png → Instead of writing code , you feed data to the
096. selfMade_200.png → for example , one kind of algorithm is a classification
097. selfMade_201.png → algorithm . It can put data into different groups ... "
098. selfMade_202.png → The classification algorithm used to detect handwritten
099. selfMade_203.png → alphabets could also be used to classify emails .
100. selfMade_204.png → into spam and not-spam .
```

Accuracy of all Hanwritten English Line Images Using TrOCR

```
# ✅ Step 1: Install Required Libraries
!pip install -q transformers torch torchvision torchaudio jiwer opencv-python

# ✅ Step 2: Import Libraries
```

```
import os
import glob
import cv2
import numpy as np
from PIL import Image
import torch
from tqdm import tqdm
from transformers import TrOCRProcessor, VisionEncoderDecoderModel
from jiwer import cer, wer

# ✅ Step 3: Set Dataset Paths
image_dir = "/content/handwritten500/handwritten500/english_lines"
gt_dir = os.path.join(image_dir, "ground_truth")

# ✅ Step 4: Load Improved TrOCR Model (LARGE)
processor = TrOCRProcessor.from_pretrained("microsoft/trocr-large-handwritten")
model = VisionEncoderDecoderModel.from_pretrained("microsoft/trocr-large-handwritten")
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# ✅ Step 5: Enhanced Preprocessing Function (OpenCV)
def preprocess_image(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        raise ValueError(f"Image not loaded: {image_path}")

    # Resize for consistency
    h, w = img.shape
    if h < 32:
        scale = 32 / h
        img = cv2.resize(img, (int(w * scale), 32), interpolation=cv2.INTER_AREA)

    # Binarize using adaptive threshold
    img = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                cv2.THRESH_BINARY, 11, 2)

    # Convert to 3-channel RGB for model
    img_rgb = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
    return Image.fromarray(img_rgb)

# ✅ Step 6: Prediction Function
def predict_text(image_path):
    try:
        image = preprocess_image(image_path)
        inputs = processor(images=image, return_tensors="pt").to(device)
        generated_ids = model.generate(inputs.pixel_values)
        pred_text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]

        # Clean output
        return pred_text.strip().replace("\n", " ").replace("  ", " ")
    except Exception as e:
        return f"[ERROR] {e}"

# ✅ Step 7: Run Evaluation on All 270 Images
image_paths = sorted(glob.glob(os.path.join(image_dir, "*.png")))
predictions, ground_truths = [], []

print("📊 Evaluating with Improved Accuracy...\n")
for path in tqdm(image_paths, desc="🔍 Predicting"):
    base_name = os.path.basename(path).replace(".png", ".txt")
    gt_path = os.path.join(gt_dir, base_name)

    if not os.path.exists(gt_path):
        print(f"⚠️ Missing GT for {base_name}, skipping.")
        continue

    # Ground truth
    with open(gt_path, "r", encoding="utf-8") as f:
        gt_text = f.read().strip().lower()

    # Prediction
    pred_text = predict_text(path).lower()

    ground_truths.append(gt_text)
    predictions.append(pred_text)

# ✅ Step 8: Compute Metrics
```

```
character_error = cer(ground_truths, predictions)
word_error = wer(ground_truths, predictions)

character_accuracy = 100 * (1 - character_error)
word_accuracy = 100 * (1 - word_error)

# ✅ Step 9: Print Results
print("\n📈 Improved Evaluation Summary:")
print(f"🧾 Character Error Rate (CER):   {character_error:.4f}")
print(f"✅ Character-Level Accuracy:     {character_accuracy:.2f}%")
print(f"🧾 Word Error Rate (WER):        {word_error:.4f}")
print(f"✅ Word-Level Accuracy:          {word_accuracy:.2f}%")
```

```
/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
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(
preprocessor_config.json: 100%                                  224/224 [00:00<00:00, 17.2kB/s]
Using a slow image processor as `use_fast` is unset and a slow processor was saved with this model. `use_fast=True` will be the def
tokenizer_config.json:        1.12k/? [00:00<00:00, 106kB/s]
vocab.json:        899k/? [00:00<00:00, 16.9MB/s]
merges.txt:        456k/? [00:00<00:00, 21.2MB/s]
special_tokens_map.json: 100%                                   772/772 [00:00<00:00, 70.7kB/s]
config.json:        4.13k/? [00:00<00:00, 238kB/s]
pytorch_model.bin: 100%                                 2.23G/2.23G [00:53<00:00, 80.4MB/s]
model.safetensors: 100%                                 2.23G/2.23G [01:09<00:00, 82.3MB/s]
Some weights of VisionEncoderDecoderModel were not initialized from the model checkpoint at microsoft/trocr-large-handwritten and a
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
generation_config.json: 100%                                    190/190 [00:00<00:00, 11.2kB/s]
📊 Evaluating with Improved Accuracy...


🔍 Predicting:   0%|          | 0/270 [00:00<?, ?it/s]
🔍 Predicting:   0%|          | 1/270 [00:25<1:55:26, 25.75s/it]
🔍 Predicting:   1%|          | 2/270 [00:51<1:55:39, 25.90s/it]
🔍 Predicting:   1%|          | 3/270 [01:03<1:27:00, 19.55s/it]
🔍 Predicting:   1%|▏         | 4/270 [01:16<1:14:23, 16.78s/it]
🔍 Predicting:   2%|▏         | 5/270 [01:28<1:07:08, 15.20s/it]
🔍 Predicting:   2%|▏         | 6/270 [01:41<1:03:31, 14.44s/it]
🔍 Predicting:   3%|▎         | 7/270 [01:59<1:07:59, 15.51s/it]
🔍 Predicting:   3%|▎         | 8/270 [02:13<1:05:27, 14.99s/it]
🔍 Predicting:   3%|▎         | 9/270 [02:25<1:01:44, 14.19s/it]
🔍 Predicting:   4%|▎         | 10/270 [02:37<58:22, 13.47s/it]
🔍 Predicting:   4%|▍         | 11/270 [02:51<58:27, 13.54s/it]
🔍 Predicting:   4%|▍         | 12/270 [03:03<56:44, 13.20s/it]
🔍 Predicting:   5%|▍         | 13/270 [03:18<58:14, 13.60s/it]
🔍 Predicting:   5%|▌         | 14/270 [03:32<59:08, 13.86s/it]
🔍 Predicting:   6%|▌         | 15/270 [03:46<59:02, 13.89s/it]
🔍 Predicting:   6%|▌         | 16/270 [04:03<1:02:12, 14.70s/it]
🔍 Predicting:   6%|▋         | 17/270 [04:20<1:05:01, 15.42s/it]
🔍 Predicting:   7%|▋         | 18/270 [04:30<58:08, 13.85s/it]
🔍 Predicting:   7%|▋         | 19/270 [04:44<58:44, 14.04s/it]
🔍 Predicting:   7%|▋         | 20/270 [04:58<58:10, 13.96s/it]
🔍 Predicting:   8%|▊         | 21/270 [05:15<1:01:16, 14.77s/it]
🔍 Predicting:   8%|▊         | 22/270 [05:27<57:26, 13.90s/it]
🔍 Predicting:   9%|▊         | 23/270 [05:43<59:45, 14.52s/it]
🔍 Predicting:   9%|▉         | 24/270 [05:55<57:18, 13.98s/it]
🔍 Predicting:   9%|▉         | 25/270 [06:10<57:27, 14.07s/it]
🔍 Predicting:  10%|▉         | 26/270 [06:20<52:29, 12.91s/it]
🔍 Predicting:  10%|▉         | 27/270 [06:32<51:05, 12.62s/it]
🔍 Predicting:  10%|▉         | 28/270 [06:45<51:32, 12.78s/it]
🔍 Predicting:  11%|▉         | 29/270 [06:57<50:02, 12.46s/it]
🔍 Predicting:  11%|▉         | 30/270 [07:10<50:30, 12.63s/it]
🔍 Predicting:  11%|█         | 31/270 [07:22<50:08, 12.59s/it]
🔍 Predicting:  12%|█         | 32/270 [07:34<49:08, 12.39s/it]
🔍 Predicting:  12%|█         | 33/270 [07:47<49:00, 12.41s/it]
🔍 Predicting:  13%|█▏        | 34/270 [07:57<46:38, 11.86s/it]
🔍 Predicting:  13%|█▏        | 35/270 [08:08<45:29, 11.62s/it]
🔍 Predicting:  13%|█▏        | 36/270 [08:17<42:25, 10.88s/it]
🔍 Predicting:  14%|█▏        | 37/270 [08:29<43:19, 11.16s/it]
🔍 Predicting:  14%|█▎        | 38/270 [08:44<47:43, 12.34s/it]
🔍 Predicting:  14%|█▎        | 39/270 [08:57<48:03, 12.48s/it]
🔍 Predicting:  15%|█▍        | 40/270 [09:10<48:30, 12.65s/it]
🔍 Predicting:  15%|█▍        | 41/270 [09:21<45:51, 12.02s/it]
🔍 Predicting:  16%|█▍        | 42/270 [09:34<47:26, 12.49s/it]
🔍 Predicting:  16%|█▌        | 43/270 [09:47<47:09, 12.46s/it]
🔍 Predicting:  16%|█▋        | 44/270 [09:58<45:25, 12.06s/it]
🔍 Predicting:  17%|█▋        | 45/270 [10:08<43:37, 11.63s/it]
🔍 Predicting:  17%|█▋        | 46/270 [10:20<43:35, 11.68s/it]
🔍 Predicting:  17%|█▋        | 47/270 [10:29<40:38, 10.94s/it]
🔍 Predicting:  18%|█▊        | 48/270 [10:44<44:02, 11.90s/it]
🔍 Predicting:  18%|█▊        | 49/270 [10:58<46:26, 12.61s/it]
🔍 Predicting:  19%|█▊        | 50/270 [11:11<46:32, 12.69s/it]
🔍 Predicting:  19%|█▉        | 51/270 [11:28<51:46, 14.19s/it]
🔍 Predicting:  19%|█▉        | 52/270 [11:42<50:56, 14.02s/it]
🔍 Predicting:  20%|█▉        | 53/270 [11:53<47:24, 13.11s/it]
🔍 Predicting:  20%|██        | 54/270 [12:09<49:55, 13.87s/it]
🔍 Predicting:  20%|██        | 55/270 [12:21<48:10, 13.44s/it]
```

```
Predicting:  21%|██        | 56/270 [12:30<43:33, 12.21s/it]
Predicting:  21%|██        | 57/270 [12:41<41:34, 11.71s/it]
Predicting:  21%|██        | 58/270 [12:53<42:01, 11.89s/it]
Predicting:  22%|██        | 59/270 [13:05<41:35, 11.82s/it]
Predicting:  22%|██        | 60/270 [13:17<41:15, 11.79s/it]
Predicting:  23%|██        | 61/270 [13:32<44:33, 12.79s/it]
Predicting:  23%|██        | 62/270 [13:43<42:27, 12.25s/it]
Predicting:  23%|██        | 63/270 [13:54<41:13, 11.95s/it]
Predicting:  24%|██        | 64/270 [14:07<41:50, 12.19s/it]
Predicting:  24%|██        | 65/270 [14:18<40:38, 11.89s/it]
Predicting:  24%|██        | 66/270 [14:31<41:48, 12.30s/it]
Predicting:  25%|██        | 67/270 [14:44<41:46, 12.35s/it]
Predicting:  25%|██        | 68/270 [14:57<42:27, 12.61s/it]
Predicting:  26%|██        | 69/270 [15:09<42:03, 12.56s/it]
Predicting:  26%|██        | 70/270 [15:22<42:17, 12.69s/it]
Predicting:  26%|██▏       | 71/270 [15:36<42:54, 12.94s/it]
Predicting:  27%|██▏       | 72/270 [15:46<40:14, 12.19s/it]
Predicting:  27%|██▏       | 73/270 [16:00<41:07, 12.52s/it]
Predicting:  27%|██▏       | 74/270 [16:11<40:01, 12.25s/it]
Predicting:  28%|██▏       | 75/270 [16:27<43:03, 13.25s/it]
Predicting:  28%|██▏       | 76/270 [16:46<48:15, 14.93s/it]
Predicting:  29%|██▏       | 77/270 [17:03<50:28, 15.69s/it]
Predicting:  29%|██▎       | 78/270 [17:21<52:02, 16.27s/it]
Predicting:  29%|██▎       | 79/270 [17:35<49:58, 15.70s/it]
Predicting:  30%|██▎       | 80/270 [17:51<49:59, 15.79s/it]
Predicting:  30%|██▎       | 81/270 [18:08<51:12, 16.26s/it]
Predicting:  30%|██▍       | 82/270 [18:23<49:15, 15.72s/it]
Predicting:  31%|██▍       | 83/270 [18:34<45:04, 14.46s/it]
Predicting:  31%|██▍       | 84/270 [18:46<41:42, 13.45s/it]
Predicting:  31%|██▍       | 85/270 [18:57<39:29, 12.81s/it]
Predicting:  32%|██▍       | 86/270 [19:09<39:04, 12.74s/it]
Predicting:  32%|██▍       | 87/270 [19:21<38:11, 12.52s/it]
Predicting:  33%|██▌       | 88/270 [19:31<35:15, 11.62s/it]
Predicting:  33%|██▌       | 89/270 [19:43<35:43, 11.84s/it]
Predicting:  33%|██▌       | 90/270 [19:56<36:05, 12.03s/it]
Predicting:  34%|██▌       | 91/270 [20:09<36:50, 12.35s/it]
Predicting:  34%|██▌       | 92/270 [20:18<33:55, 11.44s/it]
Predicting:  34%|██▌       | 93/270 [20:31<35:10, 11.92s/it]
Predicting:  35%|██▌       | 94/270 [20:44<35:28, 12.10s/it]
Predicting:  35%|██▋       | 95/270 [20:57<36:10, 12.40s/it]
Predicting:  36%|██▋       | 96/270 [21:10<36:26, 12.57s/it]
Predicting:  36%|██▋       | 97/270 [21:23<36:53, 12.80s/it]
Predicting:  36%|██▋       | 98/270 [21:33<34:31, 12.04s/it]
Predicting:  37%|██▋       | 99/270 [21:47<35:24, 12.42s/it]
Predicting:  37%|██▊       | 100/270 [21:58<34:28, 12.17s/it]
Predicting:  37%|██▊       | 101/270 [22:11<34:20, 12.19s/it]
Predicting:  38%|██▊       | 102/270 [22:21<33:04, 11.81s/it]
Predicting:  38%|██▊       | 103/270 [22:34<33:46, 12.14s/it]
Predicting:  39%|██▊       | 104/270 [22:47<33:56, 12.27s/it]
Predicting:  39%|██▉       | 105/270 [22:59<33:54, 12.33s/it]
Predicting:  39%|██▉       | 106/270 [23:12<33:46, 12.36s/it]
Predicting:  40%|██▉       | 107/270 [23:23<32:27, 11.95s/it]
Predicting:  40%|██▉       | 108/270 [23:35<32:06, 11.89s/it]
Predicting:  40%|██▉       | 109/270 [23:48<33:10, 12.36s/it]
Predicting:  41%|███       | 110/270 [24:01<33:25, 12.54s/it]
Predicting:  41%|███       | 111/270 [24:13<32:44, 12.35s/it]
Predicting:  41%|███       | 112/270 [24:26<32:54, 12.50s/it]
Predicting:  42%|███       | 113/270 [24:37<31:44, 12.13s/it]
Predicting:  42%|███       | 114/270 [24:48<30:49, 11.85s/it]
Predicting:  43%|███       | 115/270 [25:00<30:28, 11.80s/it]
Predicting:  43%|███▏      | 116/270 [25:13<31:23, 12.23s/it]
Predicting:  43%|███▏      | 117/270 [25:25<31:08, 12.21s/it]
Predicting:  44%|███▏      | 118/270 [25:33<27:29, 10.85s/it]
Predicting:  44%|███▏      | 119/270 [25:46<28:34, 11.35s/it]
Predicting:  44%|███▏      | 120/270 [25:58<29:13, 11.69s/it]
Predicting:  45%|███▏      | 121/270 [26:09<28:33, 11.50s/it]
Predicting:  45%|███▎      | 122/270 [26:21<28:27, 11.54s/it]
Predicting:  46%|███▎      | 123/270 [26:33<28:46, 11.75s/it]
Predicting:  46%|███▎      | 124/270 [26:46<29:30, 12.13s/it]
Predicting:  46%|███▎      | 125/270 [26:56<28:04, 11.62s/it]
Predicting:  47%|███▎      | 126/270 [27:07<26:58, 11.24s/it]
Predicting:  47%|███▍      | 127/270 [27:17<25:56, 10.88s/it]
Predicting:  47%|███▍      | 128/270 [27:27<25:13, 10.66s/it]
Predicting:  48%|███▍      | 129/270 [27:37<24:49, 10.56s/it]
Predicting:  48%|███▍      | 130/270 [27:48<24:37, 10.55s/it]
Predicting:  49%|███▍      | 131/270 [27:58<24:17, 10.48s/it]
Predicting:  49%|███▍      | 132/270 [28:13<27:10, 11.81s/it]
Predicting:  49%|███▌      | 133/270 [28:23<25:58, 11.38s/it]
Predicting:  50%|███▌      | 134/270 [28:34<25:35, 11.29s/it]
Predicting:  50%|███▌      | 135/270 [28:47<26:12, 11.65s/it]
Predicting:  50%|███▌      | 136/270 [28:59<26:25, 11.83s/it]
Predicting:  51%|███▌      | 137/270 [29:11<26:00, 11.73s/it]
Predicting:  51%|███▌      | 138/270 [29:21<24:51, 11.30s/it]
Predicting:  51%|███▋      | 139/270 [29:30<23:27, 10.74s/it]
```

```
Predicting:  52%|          | 140/270 [29:43<24:16, 11.21s/it]
Predicting:  52%|          | 141/270 [29:52<22:46, 10.59s/it]
Predicting:  53%|          | 142/270 [30:03<22:47, 10.69s/it]
Predicting:  53%|          | 143/270 [30:15<23:38, 11.17s/it]
Predicting:  53%|          | 144/270 [30:30<25:53, 12.33s/it]
Predicting:  54%|          | 145/270 [30:45<27:07, 13.02s/it]
Predicting:  54%|          | 146/270 [30:58<26:59, 13.06s/it]
Predicting:  54%|          | 147/270 [31:10<26:08, 12.75s/it]
Predicting:  55%|          | 148/270 [31:23<26:20, 12.96s/it]
Predicting:  55%|          | 149/270 [31:38<27:19, 13.55s/it]
Predicting:  56%|          | 150/270 [31:54<28:33, 14.28s/it]
Predicting:  56%|          | 151/270 [32:09<28:29, 14.36s/it]
Predicting:  56%|          | 152/270 [32:17<24:31, 12.47s/it]
Predicting:  57%|          | 153/270 [32:29<24:15, 12.44s/it]
Predicting:  57%|          | 154/270 [32:43<24:52, 12.87s/it]
Predicting:  57%|          | 155/270 [32:59<26:10, 13.65s/it]
Predicting:  58%|          | 156/270 [33:12<25:55, 13.64s/it]
Predicting:  58%|          | 157/270 [33:27<26:15, 13.94s/it]
Predicting:  59%|          | 158/270 [33:38<24:34, 13.17s/it]
Predicting:  59%|          | 159/270 [33:51<23:58, 12.96s/it]
Predicting:  59%|          | 160/270 [34:05<24:15, 13.23s/it]
Predicting:  60%|          | 161/270 [34:18<24:04, 13.25s/it]
Predicting:  60%|          | 162/270 [34:34<25:37, 14.23s/it]
Predicting:  60%|          | 163/270 [34:46<24:04, 13.50s/it]
Predicting:  61%|          | 164/270 [34:59<23:26, 13.27s/it]
Predicting:  61%|          | 165/270 [35:11<22:29, 12.85s/it]
Predicting:  61%|          | 166/270 [35:23<21:51, 12.61s/it]
Predicting:  62%|          | 167/270 [35:36<21:59, 12.81s/it]
Predicting:  62%|          | 168/270 [35:49<21:38, 12.73s/it]
Predicting:  63%|          | 169/270 [36:01<21:03, 12.51s/it]
Predicting:  63%|          | 170/270 [36:11<19:38, 11.79s/it]
Predicting:  63%|          | 171/270 [36:25<20:36, 12.49s/it]
Predicting:  64%|          | 172/270 [36:39<21:14, 13.00s/it]
Predicting:  64%|          | 173/270 [36:52<20:48, 12.87s/it]
Predicting:  64%|          | 174/270 [37:05<20:54, 13.07s/it]
Predicting:  65%|          | 175/270 [37:20<21:28, 13.56s/it]
Predicting:  65%|          | 176/270 [37:32<20:40, 13.20s/it]
Predicting:  66%|          | 177/270 [37:45<20:07, 12.99s/it]
Predicting:  66%|          | 178/270 [37:57<19:25, 12.67s/it]
Predicting:  66%|          | 179/270 [38:05<17:22, 11.46s/it]
Predicting:  67%|          | 180/270 [38:18<17:39, 11.77s/it]
Predicting:  67%|          | 181/270 [38:30<17:33, 11.84s/it]
Predicting:  67%|          | 182/270 [38:43<17:57, 12.25s/it]
Predicting:  68%|          | 183/270 [38:56<18:03, 12.45s/it]
Predicting:  68%|          | 184/270 [39:09<18:16, 12.75s/it]
Predicting:  69%|          | 185/270 [39:21<17:26, 12.31s/it]
Predicting:  69%|          | 186/270 [39:33<17:21, 12.40s/it]
Predicting:  69%|          | 187/270 [39:42<15:36, 11.28s/it]
Predicting:  70%|          | 188/270 [39:56<16:40, 12.21s/it]
Predicting:  70%|          | 189/270 [40:09<16:38, 12.33s/it]
Predicting:  70%|          | 190/270 [40:22<16:48, 12.61s/it]
Predicting:  71%|          | 191/270 [40:36<17:09, 13.03s/it]
Predicting:  71%|          | 192/270 [40:52<18:06, 13.93s/it]
Predicting:  71%|          | 193/270 [41:07<18:08, 14.14s/it]
Predicting:  72%|          | 194/270 [41:21<17:50, 14.08s/it]
Predicting:  72%|          | 195/270 [41:31<16:07, 12.89s/it]
Predicting:  73%|          | 196/270 [41:46<16:33, 13.42s/it]
Predicting:  73%|          | 197/270 [42:01<16:56, 13.93s/it]
Predicting:  73%|          | 198/270 [42:17<17:32, 14.62s/it]
Predicting:  74%|          | 199/270 [42:31<17:16, 14.59s/it]
Predicting:  74%|          | 200/270 [42:47<17:26, 14.95s/it]
Predicting:  74%|          | 201/270 [43:02<17:17, 15.04s/it]
Predicting:  75%|          | 202/270 [43:17<16:49, 14.84s/it]
Predicting:  75%|          | 203/270 [43:30<16:06, 14.42s/it]
Predicting:  76%|          | 204/270 [43:38<13:42, 12.46s/it]
Predicting:  76%|          | 205/270 [43:46<11:54, 10.99s/it]
Predicting:  76%|          | 206/270 [43:59<12:24, 11.63s/it]
Predicting:  77%|          | 207/270 [44:11<12:27, 11.87s/it]
Predicting:  77%|          | 208/270 [44:23<12:20, 11.94s/it]
Predicting:  77%|          | 209/270 [44:37<12:34, 12.37s/it]
Predicting:  78%|          | 210/270 [44:48<12:03, 12.06s/it]
Predicting:  78%|          | 211/270 [45:00<11:42, 11.90s/it]
Predicting:  79%|          | 212/270 [45:12<11:41, 12.09s/it]
Predicting:  79%|          | 213/270 [45:23<11:05, 11.68s/it]
Predicting:  79%|          | 214/270 [45:34<10:50, 11.61s/it]
Predicting:  80%|          | 215/270 [45:46<10:38, 11.61s/it]
Predicting:  80%|          | 216/270 [45:56<10:04, 11.19s/it]
Predicting:  80%|          | 217/270 [46:06<09:38, 10.91s/it]
Predicting:  81%|          | 218/270 [46:17<09:19, 10.75s/it]
Predicting:  81%|          | 219/270 [46:29<09:25, 11.10s/it]
Predicting:  81%|          | 220/270 [46:40<09:18, 11.17s/it]
Predicting:  82%|          | 221/270 [46:51<09:05, 11.13s/it]
Predicting:  82%|          | 222/270 [47:02<08:48, 11.01s/it]
Predicting:  83%|          | 223/270 [47:14<08:59, 11.47s/it]
Predicting:  83%|          | 224/270 [47:30<09:45, 12.72s/it]
```

Prediction of Printed & Handwritten Text Image Using EasyOCR

```
!pip install easyocr
!pip install opencv-python matplotlib
```

```
Using cached pyclipper-1.3.0.post6-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (969 kB)
Downloading python_bidi-0.6.6-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (292 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 292.9/292.9 kB 17.7 MB/s eta 0:00:00
Installing collected packages: python-bidi, pyclipper, nvidia-nvjitlink-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-
  Attempting uninstall: nvidia-nvjitlink-cu12
    Found existing installation: nvidia-nvjitlink-cu12 12.5.82
    Uninstalling nvidia-nvjitlink-cu12-12.5.82:
      Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
  Attempting uninstall: nvidia-curand-cu12
    Found existing installation: nvidia-curand-cu12 10.3.6.82
    Uninstalling nvidia-curand-cu12-10.3.6.82:
      Successfully uninstalled nvidia-curand-cu12-10.3.6.82
  Attempting uninstall: nvidia-cufft-cu12
    Found existing installation: nvidia-cufft-cu12 11.2.3.61
    Uninstalling nvidia-cufft-cu12-11.2.3.61:
      Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
  Attempting uninstall: nvidia-cuda-runtime-cu12
    Found existing installation: nvidia-cuda-runtime-cu12 12.5.82
    Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-nvrtc-cu12
    Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
    Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-nvrtc-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-cupti-cu12
    Found existing installation: nvidia-cuda-cupti-cu12 12.5.82
    Uninstalling nvidia-cuda-cupti-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
  Attempting uninstall: nvidia-cublas-cu12
    Found existing installation: nvidia-cublas-cu12 12.5.3.2
    Uninstalling nvidia-cublas-cu12-12.5.3.2:
      Successfully uninstalled nvidia-cublas-cu12-12.5.3.2
  Attempting uninstall: nvidia-cusparse-cu12
    Found existing installation: nvidia-cusparse-cu12 12.5.1.3
    Uninstalling nvidia-cusparse-cu12-12.5.1.3:
      Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
  Attempting uninstall: nvidia-cudnn-cu12
    Found existing installation: nvidia-cudnn-cu12 9.3.0.75
    Uninstalling nvidia-cudnn-cu12-9.3.0.75:
      Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
  Attempting uninstall: nvidia-cusolver-cu12
    Found existing installation: nvidia-cusolver-cu12 11.6.3.83
    Uninstalling nvidia-cusolver-cu12-11.6.3.83:
      Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
Successfully installed easyocr-1.7.2 ninja-1.11.1.4 nvidia-cublas-cu12-12.4.5.8 nvidia-cuda-cupti-cu12-12.4.127 nvidia-cuda-nvrtc
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/dist-packages (4.11.0.86)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.11/dist-packages (from opencv-python) (2.0.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.58.4)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib) (1.17.
```

```
import easyocr
import matplotlib.pyplot as plt
import cv2
from PIL import Image

# Load image
img_path = "/content/whatsapp.jpg"
image = cv2.imread(img_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Initialize OCR reader
reader = easyocr.Reader(['en'], gpu=False)  # CPU-only

# Perform OCR
```

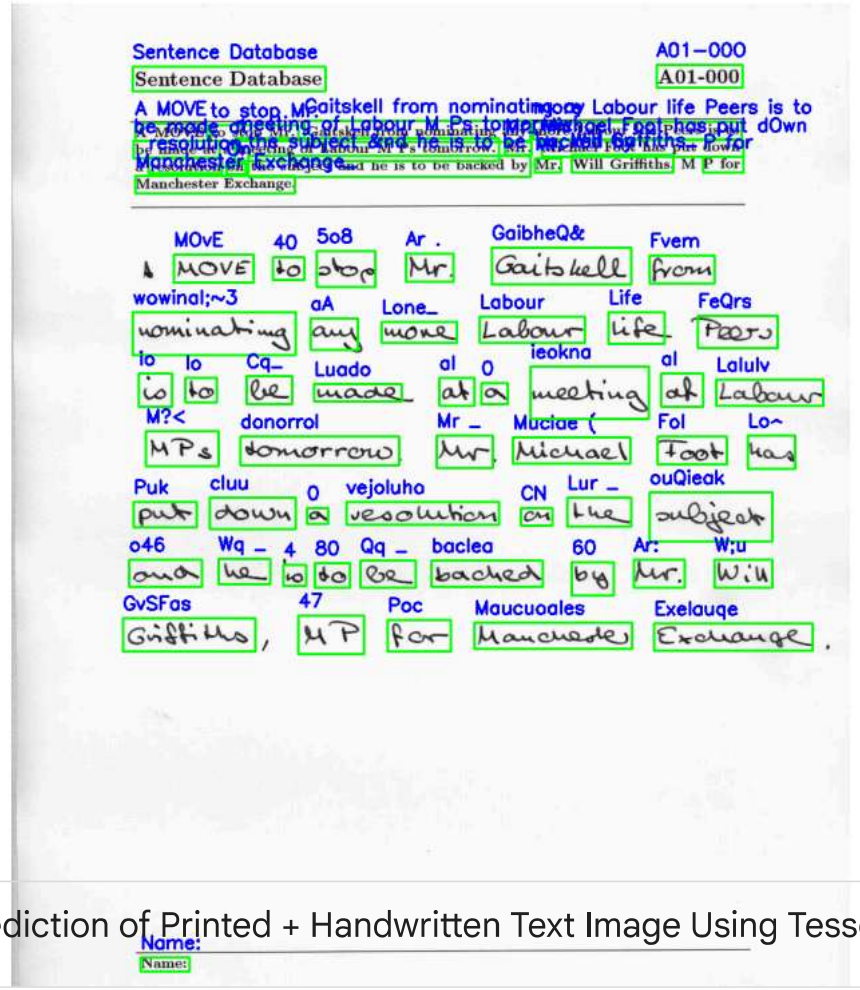
```
results = reader.readtext(img_path)

# Show image with boxes
for (bbox, text, prob) in results:
    (top_left, top_right, bottom_right, bottom_left) = bbox
    top_left = tuple(map(int, top_left))
    bottom_right = tuple(map(int, bottom_right))
    cv2.rectangle(image, top_left, bottom_right, (0, 255, 0), 2)
    cv2.putText(image, text, (top_left[0], top_left[1] - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

# Show image
plt.figure(figsize=(12, 12))
plt.imshow(image)
plt.axis('off')
plt.show()

# Print text
print("📌 Extracted Text:")
for (_, text, _) in results:
    print(text)
```

```
WARNING:easyocr.easyocr:Using CPU. Note: This module is much faster with a GPU.
WARNING:easyocr.easyocr:Downloading detection model, please wait. This may take several minutes depending upon your network conne
Progress: |                                        | 100.0% CompleteWARNING:easyocr.easyocr:Downloading recognition mod
Progress: |                                        | 100.0% Complete
```



Prediction of Printed + Handwritten Text Image Using Tesseract

```
# STEP 1: Install Tesseract and pytesseract
!apt install tesseract-ocr
!apt install tesseract-ocr-hin
!pip install pytesseract
!pip install pillow

# STEP 2: Import Libraries
import pytesseract
from PIL import Image
import matplotlib.pyplot as plt

# STEP 3: Load Your Image
image_path = "/content/whatsapp.jpg"  # Replace with your image
img = Image.open(image_path)

# STEP 4: OCR Text Extraction (English + Hindi)
text = pytesseract.image_to_string(img, lang='eng+hin')
print("📌 Extracted Text:")
print(text)

# STEP 5: Display the Image
plt.imshow(img)
plt.axis('off')
plt.title("Input Image")
plt.show()
```

```
MOvE
40
Ar .
GaibheQ&
Fvem
wowinal;~3
Lone_
Labour
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
tesseract-ocr is already the newest version (4.1.1-2.1build1).
0 upgraded, 0 newly installed, 0 to remove and 35 not upgraded.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
tesseract-ocr-hin is already the newest version (1:4.00~git30-7274cfa-1.1).
0 upgraded, 0 newly installed, 0 to remove and 35 not upgraded.
Requirement already satisfied: pytesseract in /usr/local/lib/python3.11/dist-packages (0.3.13)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.11/dist-packages (from pytesseract) (24.2)
Requirement already satisfied: Pillow>=8.0.0 in /usr/local/lib/python3.11/dist-packages (from pytesseract) (11.2.1)
Requirement already satisfied: pillow in /usr/local/lib/python3.11/dist-packages (11.2.1)

📌 Extracted Text:
Sentence Database A01-000

A MOVE to stop Mr. Gaitskell from nominating any more Labour life Peers is to
be made at a meeting of Labour M Ps tomorrow. Mr. Michael Foot has put down
a resolution on the subject and he is to be backed by Mr. Will Griffiths, M P for
Manchester Exchange.

k MOVE to sloe Mr. Galbkell from

MOWAL MOI wach OMA ORR Lobowe Ufe_ Poors

wo to G2 wade aka medhiua af Lado

MPs Youcrrono, My, Michael Fook hoy

pre Aown a vesotution on bha sudjede

On Lo to 6o bLackeR by Mr. Wit

Gules , WP Por Moarcrades Cxroronge ,

Name:
```

Input Image

Prediction of Printed + Handwritten Text Image Using TrOCR

```
from google.colab import files
uploaded = files.upload()  # Upload a full handwritten page image
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
from transformers import TrOCRProcessor, VisionEncoderDecoderModel
import torch

# Load model and processor
processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-handwritten')
model = VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-base-handwritten')
```

```
/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
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(
preprocessor_config.json: 100%  224/224 [00:00<00:00, 14.1kB/s]
Using a slow image processor as `use_fast` is unset and a slow processor was saved with this model. `use_fast=True` will be the def
tokenizer_config.json:  1.12k/? [00:00<00:00, 79.9kB/s]
vocab.json:  899k/? [00:00<00:00, 13.3MB/s]
merges.txt:  456k/? [00:00<00:00, 15.2MB/s]
special_tokens_map.json: 100%  772/772 [00:00<00:00, 89.0kB/s]
config.json:  4.17k/? [00:00<00:00, 414kB/s]
model.safetensors: 100%  1.33G/1.33G [00:20<00:00, 106MB/s]
Some weights of VisionEncoderDecoderModel were not initialized from the model checkpoint at microsoft/trocr-base-handwritten and ar
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
generation_config.json: 100%  190/190 [00:00<00:00, 22.2kB/s]
```

```
import cv2
import numpy as np
from PIL import Image

# Load and preprocess image
img_path = list(uploaded.keys())[0]
image = cv2.imread(img_path)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]

# Horizontal line projection to find line regions
horizontal_sum = np.sum(thresh, axis=1)
lines = []
start = None

for i, val in enumerate(horizontal_sum):
    if val > 0 and start is None:
        start = i
    elif val == 0 and start is not None:
        end = i
        if end - start > 15:  # filter short noise
            lines.append((start, end))
        start = None
```

```
from PIL import Image
results = []

for y1, y2 in lines:
    line_img = image[y1:y2, :]
    pil_img = Image.fromarray(cv2.cvtColor(line_img, cv2.COLOR_BGR2RGB))
    pixel_values = processor(images=pil_img, return_tensors="pt").pixel_values

    # Beam decoding for accuracy
    generated_ids = model.generate(pixel_values, max_length=128, num_beams=5, early_stopping=True)
    predicted_text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
    results.append(predicted_text)
```

```
print("📝 Final Predicted Text:")
for line in results:
    print(line)
```

```
📝 Final Predicted Text:
```

Prediction of all the Handwritten Paragraph Images using TrOCR

```
import zipfile
import os

zip_path = "/content/english_paragraph.zip"
extract_path = "/content/english_paragraph"

# Ensure clean extraction directory
if not os.path.exists(extract_path):
    os.makedirs(extract_path)

# Extract
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print("✅ Successfully unzipped to:", extract_path)
```

```
✅ Successfully unzipped to: /content/english_paragraph
```

```
!pip install jiwer
```

```
Collecting jiwer
  Downloading jiwer-4.0.0-py3-none-any.whl.metadata (3.3 kB)
Requirement already satisfied: click>=8.1.8 in /usr/local/lib/python3.11/dist-packages (from jiwer) (8.2.1)
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-4.0.0-py3-none-any.whl (23 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 43.4 MB/s eta 0:00:00
Installing collected packages: rapidfuzz, jiwer
Successfully installed jiwer-4.0.0 rapidfuzz-3.13.0
```

```
from transformers import TrOCRProcessor, VisionEncoderDecoderModel
import torch

# Load model and processor
processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-handwritten')
model = VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-base-handwritten')
```

```
/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
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(
preprocessor_config.json: 100%  224/224 [00:00<00:00, 18.0kB/s]
Using a slow image processor as `use_fast` is unset and a slow processor was saved with this model. `use_fast=True` will be the def
tokenizer_config.json:  1.12k/? [00:00<00:00, 106kB/s]
vocab.json:  899k/? [00:00<00:00, 14.9MB/s]
merges.txt:  456k/? [00:00<00:00, 15.5MB/s]
special_tokens_map.json: 100%  772/772 [00:00<00:00, 81.9kB/s]
config.json:  4.17k/? [00:00<00:00, 292kB/s]
model.safetensors: 100%  1.33G/1.33G [00:44<00:00, 33.5MB/s]
Some weights of VisionEncoderDecoderModel were not initialized from the model checkpoint at microsoft/trocr-base-handwritten and ar
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
generation_config.json: 100%  190/190 [00:00<00:00, 20.1kB/s]
```

```
import os
import glob
import cv2
import numpy as np
from PIL import Image
import torch
from tqdm import tqdm
from jiwer import cer, wer

# Define paths (using the extraction path from the unzip step)
```

```
image_dir = "/content/english_paragraph/english_paragraph"
gt_dir = os.path.join(image_dir, "ground_truth")

# Ensure model and processor are loaded and model is on the correct device
# Assuming 'model' and 'processor' are already loaded from a previous cell
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# Preprocessing function (reusing the one from previous analysis)
def preprocess_image_for_lines(image_path):
    img = cv2.imread(image_path)
    if img is None:
        raise ValueError(f"Image not loaded: {image_path}")

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]

    # Horizontal line projection to find line regions
    horizontal_sum = np.sum(thresh, axis=1)
    lines = []
    start = None

    for i, val in enumerate(horizontal_sum):
        if val > 0 and start is None:
            start = i
        elif val == 0 and start is not None:
            end = i
            if end - start > 15:  # filter short noise
                lines.append((start, end))
            start = None
    # Add the last line if the image doesn't end with a zero sum
    if start is not None:
        lines.append((start, len(horizontal_sum)))

    return img, lines


# Prediction function for lines
def predict_text_from_lines(image, lines):
    predicted_lines = []
    for y1, y2 in lines:
        line_img = image[y1:y2, :]
        # Ensure the line image is valid before processing
        if line_img.shape[0] > 0 and line_img.shape[1] > 0:
            pil_img = Image.fromarray(cv2.cvtColor(line_img, cv2.COLOR_BGR2RGB))
            try:
                pixel_values = processor(images=pil_img, return_tensors="pt").pixel_values.to(device)
                generated_ids = model.generate(pixel_values, max_length=128, num_beams=5, early_stopping=True)
                predicted_text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
                predicted_lines.append(predicted_text.strip())
            except Exception as e:
                predicted_lines.append(f"[OCR ERROR] {e}")
        else:
            predicted_lines.append("") # Append empty string for invalid line images

    return " ".join(predicted_lines).replace("  ", " ") # Join lines with space and clean up extra spaces


# Get all image paths (including .jpg and .png)
image_paths = sorted(glob.glob(os.path.join(image_dir, "*.png")) + glob.glob(os.path.join(image_dir, "*.jpg")))

if len(image_paths) == 0:
    print(f"No images found in the directory: {image_dir}")
else:
    predictions, ground_truths = [], []

    print(f"📊 Evaluating TrOCR Line-by-Line OCR on {len(image_paths)} images...\n")

    # Wrap image_paths with tqdm for a progress bar
    for image_path in tqdm(image_paths, desc="🔍 Processing Images"):
        base_name = os.path.basename(image_path).replace(".png", ".txt").replace(".jpg", ".txt")
        gt_path = os.path.join(gt_dir, base_name)

        if not os.path.exists(gt_path):
            print(f"\n⚠️ Missing GT for {base_name}, skipping.")
            continue
```

```
        # Process image and get lines
        try:
            image, lines = preprocess_image_for_lines(image_path)
            # Predict text from detected lines
            pred_text = predict_text_from_lines(image, lines).lower()

            # Get ground truth text
            with open(gt_path, "r", encoding="utf-8") as f:
                gt_text = f.read().strip().lower()

            ground_truths.append(gt_text)
            predictions.append(pred_text)

        except Exception as e:
            print(f"\n❌ Error processing {os.path.basename(image_path)}: {e}")
            # Append empty strings to maintain list length for metrics calculation
            ground_truths.append("")
            predictions.append("")


    # Compute Metrics
    if len(ground_truths) > 0:
        character_error = cer(ground_truths, predictions)
        word_error = wer(ground_truths, predictions)

        character_accuracy = 100 * (1 - character_error)
        word_accuracy = 100 * (1 - word_error)

        # Print Results
        print("\n📈 Evaluation Summary:")
        print(f"🧾 Character Error Rate (CER):   {character_error:.4f}")
        print(f"✅ Character-Level Accuracy:     {character_accuracy:.2f}%")
        print(f"🧾 Word Error Rate (WER):        {word_error:.4f}")
        print(f"✅ Word-Level Accuracy:          {word_accuracy:.2f}%")
    else:
        print("\n⚠️ No images were processed for evaluation.")
```

```
📊 Evaluating TrOCR Line-by-Line OCR on 72 images...

🔍 Processing Images:  60%|██████    | 43/72 [01:24<01:28,  3.04s/it]The channel dimension is ambiguous. Got image shape (1, 549,
🔍 Processing Images:  69%|██████▉   | 50/72 [01:41<00:55,  2.55s/it]The channel dimension is ambiguous. Got image shape (1, 344,
🔍 Processing Images:  76%|███████▋  | 55/72 [01:54<00:42,  2.48s/it]The channel dimension is ambiguous. Got image shape (1, 349,
🔍 Processing Images:  94%|█████████▍| 68/72 [02:28<00:09,  2.34s/it]The channel dimension is ambiguous. Got image shape (3, 363,
🔍 Processing Images: 100%|██████████| 72/72 [02:38<00:00,  2.20s/it]
📈 Evaluation Summary:
🧾 Character Error Rate (CER):   0.4151
✅ Character-Level Accuracy:     58.49%
🧾 Word Error Rate (WER):        0.8743
✅ Word-Level Accuracy:          12.57%
```

Start coding or generate with AI.

Start coding or generate with AI.

```
# Add your code here and tell me what you want to achieve.
```

```
from google.colab import files
uploaded = files.upload()  # Upload a full handwritten page image
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
from transformers import TrOCRProcessor, VisionEncoderDecoderModel
import torch

# Load model and processor
processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-handwritten')
model = VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-base-handwritten')
```

```
Some weights of VisionEncoderDecoderModel were not initialized from the model checkpoint at microsoft/trocr-base-handwritten and ar
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
import cv2
import numpy as np
from PIL import Image

# Load and preprocess image
img_path = list(uploaded.keys())[0]
image = cv2.imread(img_path)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]

# Horizontal line projection to find line regions
horizontal_sum = np.sum(thresh, axis=1)
lines = []
start = None

for i, val in enumerate(horizontal_sum):
    if val > 0 and start is None:
        start = i
    elif val == 0 and start is not None:
        end = i
        if end - start > 15:  # filter short noise
            lines.append((start, end))
        start = None
```

```
from PIL import Image
results = []

for y1, y2 in lines:
    line_img = image[y1:y2, :]
    pil_img = Image.fromarray(cv2.cvtColor(line_img, cv2.COLOR_BGR2RGB))
    pixel_values = processor(images=pil_img, return_tensors="pt").pixel_values

    # Beam decoding for accuracy
    generated_ids = model.generate(pixel_values, max_length=128, num_beams=5, early_stopping=True)
    predicted_text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
    results.append(predicted_text)
```

```
print("📝 Final Predicted Text:")
for line in results:
    print(line)
```

```
📝 Final Predicted Text:
2 1 1 Blinds 2 3
I love to observe birds .
I enjoy their songs .
and colorful feathers .
6 Birdwatching brings .
peace and happiness .
```

```
from google.colab import files
uploaded = files.upload()  # Upload a full handwritten page image
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
from transformers import TrOCRProcessor, VisionEncoderDecoderModel
import torch

# Load model and processor
processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-handwritten')
model = VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-base-handwritten')
```

```
Some weights of VisionEncoderDecoderModel were not initialized from the model checkpoint at microsoft/trocr-base-handwritten and ar
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
import cv2
import numpy as np
from PIL import Image

# Load and preprocess image
img_path = list(uploaded.keys())[0]
image = cv2.imread(img_path)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]
```

```
# Horizontal line projection to find line regions
horizontal_sum = np.sum(thresh, axis=1)
lines = []
start = None

for i, val in enumerate(horizontal_sum):
    if val > 0 and start is None:
        start = i
    elif val == 0 and start is not None:
        end = i
        if end - start > 15:  # filter short noise
            lines.append((start, end))
        start = None
```

```
from PIL import Image
results = []

for y1, y2 in lines:
    line_img = image[y1:y2, :]
    pil_img = Image.fromarray(cv2.cvtColor(line_img, cv2.COLOR_BGR2RGB))
    pixel_values = processor(images=pil_img, return_tensors="pt").pixel_values

    # Beam decoding for accuracy
    generated_ids = model.generate(pixel_values, max_length=128, num_beams=5, early_stopping=True)
    predicted_text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
    results.append(predicted_text)
```

```
print("📝 Final Predicted Text:")
for line in results:
    print(line)
```

```
📝 Final Predicted Text:
MR. Sentence Database.000520005. A.1-132
# 200 " That cannot continue without either development being limited or an adjustment of 1.000
# been being made in financing . " The Government decided to adjust the financing - which
# ... Mr. Powell claimed was underpinning - not undermining - the service . Answering ... ...
# that the attack on " economic charges " for welfare foods , Mr. Powell said that all these
0 0000 000foods would still be free in families receiving regular National Assistance grants .0003000
# that cannot continue without either ...
of development being limited or an 8-
# adjusting being made in financing . "
# The Government decided to adjust
# the financing - which Mr. Powell
accompanied was underpinning - not
# undermining the service . Answering the
# attack on " economic charges " for welfare
0 0
Categories 1921 births2021 deaths20th century American lawyersAmerican lawyers of the United States Supreme Court of Representative
Categories 1882 births1953 deathsAmerican male sailors of the United States Navy officersAmerican male sailors of the United States
1957 British film director and director of the British Parliament of the South Carolina Housewives of the Order of
1999 2000 2001 2002 2003 2003 20042005 2006 2007 2008 2009 2010 2011 2012 2013
1942 1943 1944 1946 1947 1948 1949 1951 1951 1952 1953 1954 1954 19541965 66
```

```
Start coding or generate with AI.
```

```
from google.colab import files
import cv2
import numpy as np
from PIL import Image
import torch
from transformers import TrOCRProcessor, VisionEncoderDecoderModel
import os

# Load model and processor (reusing if already loaded)
try:
    processor
    model
except NameError:
    print("Loading TrOCR model and processor...")
    processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-handwritten')
    model = VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-base-handwritten')

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

```
print("✅ Model loaded and on device.")

# Preprocessing function (reusing the one from previous analysis)
def preprocess_image_for_lines(image_path):
    img = cv2.imread(image_path)
    if img is None:
        raise ValueError(f"Image not loaded: {image_path}")

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Using OTSU's thresholding
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]

    # Horizontal line projection to find line regions
    horizontal_sum = np.sum(thresh, axis=1)
    lines = []
    start = None

    for i, val in enumerate(horizontal_sum):
        if val > 0 and start is None:
            start = i
        elif val == 0 and start is not None:
            end = i
            # Filter short noise lines, adjust threshold (e.g., 15) if needed
            if end - start > 15:
                lines.append((start, end))
            start = None
    # Add the last line if the image doesn't end with a zero sum
    if start is not None:
        lines.append((start, len(horizontal_sum)))

    return img, lines

# Prediction function for lines
def predict_text_from_lines(image, lines):
    predicted_lines = []
    for y1, y2 in lines:
        line_img = image[y1:y2, :]
        # Ensure the line image is valid before processing
        if line_img.shape[0] > 0 and line_img.shape[1] > 0:
            pil_img = Image.fromarray(cv2.cvtColor(line_img, cv2.COLOR_BGR2RGB))
            try:
                pixel_values = processor(images=pil_img, return_tensors="pt").pixel_values.to(device)
                # Beam decoding for accuracy
                generated_ids = model.generate(pixel_values, max_length=128, num_beams=5, early_stopping=True)
                predicted_text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
                predicted_lines.append(predicted_text.strip())
            except Exception as e:
                predicted_lines.append(f"[OCR ERROR] {e}")
        else:
            predicted_lines.append("") # Append empty string for invalid line images


    return "\n".join(predicted_lines) # Join lines with newline for better readability

# Upload the 4 images
print("Please upload the 4 images:")
uploaded_files = files.upload()

if len(uploaded_files) == 0:
    print("❌ No files uploaded.")
elif len(uploaded_files) != 4:
     print(f"⚠️ You uploaded {len(uploaded_files)} files. Expected 4 images. Processing the uploaded files anyway.")


# Process each uploaded image
print("\n▶️ Processing uploaded images...")
for filename in uploaded_files.keys():
    print(f"\n--- Processing: {filename} ---")
    try:
        # Save the uploaded file temporarily to process with OpenCV
        with open(filename, 'wb') as f:
            f.write(uploaded_files[filename])

        image, lines = preprocess_image_for_lines(filename)
        if lines:
            predicted_text = predict_text_from_lines(image, lines)
            print(predicted_text)
```

```
        else:
            print("No lines detected in this image.")

        # Clean up the temporary file
        os.remove(filename)

    except Exception as e:
        print(f"❌ Error processing {filename}: {e}")

print("\n✅ Finished processing uploaded images.")
```

```
✅ Model loaded and on device.
Please upload the 4 images:
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving a01-000x.png to a01-000x.png
⚠️ You uploaded 1 files. Expected 4 images. Processing the uploaded files anyway.

▶️ Processing uploaded images...

--- Processing: a01-000x.png ---
The channel dimension is ambiguous. Got image shape (1, 2479, 3). Assuming channels are the first dimension. Use the [input_data_fo
10, Sentence Database C02-0005.0005000,000,0005000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,
# ... A MOVE to stop Mr. Gaitskell from nominating any more Labour life Peers is to
1942 1944 he made at a meeting of Labour 0MPs tomorrow . Mr. Michael Foot has put down 1950 1951
# a resolution on the subject and he is to be backed by Mr. Will Griffiths . MP for 200,000
1962 Manchester Exchange , 1982 1983 1985 1986 1987
# A MOVE to stop . Mr. Gaitskell from nominating
" any more Labour life Peers is to be made at a
# meeting of Labour MPs tomorrow . Mr. Michael ,
# Foot has put down a resolution on the subject ...
# and he is to be backed by Mr. Will Griffith ,
# UMP for Manchester Exchange .
a member of the National Register of Historic Places listings in the National Register of Historic Places is a
[OCR ERROR] mean must have 1 elements if it is an iterable, got 3

✅ Finished processing uploaded images.
```

```
from google.colab import files
import cv2
import numpy as np
from PIL import Image
import torch
from transformers import TrOCRProcessor, VisionEncoderDecoderModel
import os

# Load model and processor (reusing if already loaded)
try:
    processor
    model
except NameError:
    print("Loading TrOCR model and processor...")
    processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-handwritten')
    model = VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-base-handwritten')

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
print("✅ Model loaded and on device.")

# Preprocessing function (reusing the one from previous analysis)
def preprocess_image_for_lines(image_path):
    img = cv2.imread(image_path)
    if img is None:
        raise ValueError(f"Image not loaded: {image_path}")

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Using OTSU's thresholding
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]

    # Horizontal line projection to find line regions
    horizontal_sum = np.sum(thresh, axis=1)
    lines = []
    start = None

    for i, val in enumerate(horizontal_sum):
        if val > 0 and start is None:
            start = i
        elif val == 0 and start is not None:
```

```
            end = i
            # Filter short noise lines, adjust threshold (e.g., 15) if needed
            if end - start > 15:
                lines.append((start, end))
            start = None
    # Add the last line if the image doesn't end with a zero sum
    if start is not None:
        lines.append((start, len(horizontal_sum)))

    return img, lines

# Prediction function for lines
def predict_text_from_lines(image, lines):
    predicted_lines = []
    for y1, y2 in lines:
        line_img = image[y1:y2, :]
        # Ensure the line image is valid before processing
        if line_img.shape[0] > 0 and line_img.shape[1] > 0:
            pil_img = Image.fromarray(cv2.cvtColor(line_img, cv2.COLOR_BGR2RGB))
            try:
                pixel_values = processor(images=pil_img, return_tensors="pt").pixel_values.to(device)
                # Beam decoding for accuracy
                generated_ids = model.generate(pixel_values, max_length=128, num_beams=5, early_stopping=True)
                predicted_text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
                predicted_lines.append(predicted_text.strip())
            except Exception as e:
                predicted_lines.append(f"[OCR ERROR] {e}")
        else:
            predicted_lines.append("") # Append empty string for invalid line images


    return "\n".join(predicted_lines) # Join lines with newline for better readability

# Upload a single image
print("Please upload a single image:")
uploaded = files.upload()

if len(uploaded) == 0:
    print("❌ No file uploaded.")
elif len(uploaded) > 1:
     print(f"⚠️ You uploaded {len(uploaded)} files. Expected 1 image. Processing the first uploaded file.")
     filename = list(uploaded.keys())[0]
else:
    filename = list(uploaded.keys())[0]


# Process the uploaded image
print(f"\n▶️ Processing: {filename}...")
try:
    # Save the uploaded file temporarily to process with OpenCV
    with open(filename, 'wb') as f:
        f.write(uploaded[filename])

    image, lines = preprocess_image_for_lines(filename)
    if lines:
        predicted_text = predict_text_from_lines(image, lines)
        print("\n📝 Predicted Text:")
        print(predicted_text)
    else:
        print("No lines detected in this image.")

    # Clean up the temporary file
    os.remove(filename)

except Exception as e:
    print(f"❌ Error processing {filename}: {e}")

print("\n✅ Finished processing the image.")
```

```
✅ Model loaded and on device.
Please upload a single image:
```
Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
```
Saving a01-003.png to a01-003.png

▶️ Processing: a01-003.png...

📝 Predicted Text:
# Sentence Database .0005.0005.0005.0005.0005.0005.
# ... . Though they gather some Left-wing support , a large majority of Labour MPs ___
# # are likely to turn down the Foot-Griffiths resolution . Mr. Foot's line will be that as
# ... Labour MPs opposed the Government Bill which brought life peers into existence ,
# that they should not now put forward nominees . He believes that the House of Lords ___
# # should be abolished and that Labour should not take any steps which would appear ...
" 2 0 to " prop up " an out-dated institution , 2000 2001 2002 2003 2004
Although they may gather some Left-wing support , a
# large majority of Labour MPs are likely to
# turn down the Foot-Griffiths resolution . Mr.
# Foot's line will be that as Labour MPs .
to opposed the Government Bill which brought
# life peers into existence , they should not
# now put forward nominees . He believes
# that the House of Lords should be abolished
# and that Labour should not take any steps ...
# which would appear to " prop up " an out-dated
0 institutions listed here . See also known as well as well as well as
What links here has multiple issues Please help improve it or discuss these issues Please help improve it by
2 Nicola Beck 1953 1954
What links here has multiple issues Please help improve it or discuss these issues Please help improve it by
What links here has multiple issues Please help improve it or discuss these issues Please help
1944 American film posterDirected byGeorge AllenWritten byGeorge AllenProduced byGeorge AllenProduced byGeorge AllenProduced byGeor
What links here has multiple issues Please help improve it or discuss these issues on
What links here has multiple issues Please help improve it or discuss these issues on
MRDOT Report for the Year of the Year of the Year of the Year of the Year of the
1944 American film posterDirected byWilliam BurdettScreenplay byWilliam BurdettScreenplay by William BurdettScreenplay by Robert Bu
What links hereRelated changesUpload fileSpecial pagesPermanent linkPage informationCite this pageWikidata item
```

```python
from google.colab import files
import cv2
import numpy as np
from PIL import Image
import torch
from transformers import TrOCRProcessor, VisionEncoderDecoderModel
import os

# Load model and processor (reusing if already loaded)
try:
    processor
    model
except NameError:
    print("Loading TrOCR model and processor...")
    processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-handwritten')
    model = VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-base-handwritten')

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
print("✅ Model loaded and on device.")

# Preprocessing function (reusing the one from previous analysis)
def preprocess_image_for_lines(image_path):
    img = cv2.imread(image_path)
    if img is None:
        raise ValueError(f"Image not loaded: {image_path}")

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Using OTSU's thresholding
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]

    # Horizontal line projection to find line regions
    horizontal_sum = np.sum(thresh, axis=1)
    lines = []
    start = None

    for i, val in enumerate(horizontal_sum):
        if val > 0 and start is None:
            start = i
        elif val == 0 and start is not None:
```

```
            end = i
            # Filter short noise lines, adjust threshold (e.g., 15) if needed
            if end - start > 15:
                lines.append((start, end))
            start = None
    # Add the last line if the image doesn't end with a zero sum
    if start is not None:
        lines.append((start, len(horizontal_sum)))

    return img, lines

# Prediction function for lines
def predict_text_from_lines(image, lines):
    predicted_lines = []
    for y1, y2 in lines:
        line_img = image[y1:y2, :]
        # Ensure the line image is valid before processing
        if line_img.shape[0] > 0 and line_img.shape[1] > 0:
            pil_img = Image.fromarray(cv2.cvtColor(line_img, cv2.COLOR_BGR2RGB))
            try:
                pixel_values = processor(images=pil_img, return_tensors="pt").pixel_values.to(device)
                # Beam decoding for accuracy
                generated_ids = model.generate(pixel_values, max_length=128, num_beams=5, early_stopping=True)
                predicted_text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
                predicted_lines.append(predicted_text.strip())
            except Exception as e:
                predicted_lines.append(f"[OCR ERROR] {e}")
        else:
            predicted_lines.append("") # Append empty string for invalid line images


    return "\n".join(predicted_lines) # Join lines with newline for better readability

# Upload a single image
print("Please upload a single image:")
uploaded = files.upload()

if len(uploaded) == 0:
    print("❌ No file uploaded.")
elif len(uploaded) > 1:
     print(f"⚠️ You uploaded {len(uploaded)} files. Expected 1 image. Processing the first uploaded file.")
     filename = list(uploaded.keys())[0]
else:
    filename = list(uploaded.keys())[0]


# Process the uploaded image
print(f"\n▶️ Processing: {filename}...")
try:
    # Save the uploaded file temporarily to process with OpenCV
    with open(filename, 'wb') as f:
        f.write(uploaded[filename])

    image, lines = preprocess_image_for_lines(filename)
    if lines:
        predicted_text = predict_text_from_lines(image, lines)
        print("\n📝 Predicted Text:")
        print(predicted_text)
    else:
        print("No lines detected in this image.")

    # Clean up the temporary file
    os.remove(filename)

except Exception as e:
    print(f"❌ Error processing {filename}: {e}")

print("\n✅ Finished processing the image.")
```

```
Loading TrOCR model and processor...
/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
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(
Using a slow image processor as `use_fast` is unset and a slow processor was saved with this model. `use_fast=True` will be the def
Some weights of VisionEncoderDecoderModel were not initialized from the model checkpoint at microsoft/trocr-base-handwritten and ar
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
✅ Model loaded and on device.
Please upload a single image:
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving a01-132.png to a01-132 (1).png

▶️ Processing: a01-132 (1).png...

📝 Predicted Text:
MR. Sentence Database.000520005. A.1-132
# 200 " That cannot continue without either development being limited or an adjustment of 1.000
# been being made in financing . " The Government decided to adjust the financing - which
# ... Mr. Powell claimed was underpinning - not undermining - the service . Answering ... ...
# that the attack on " economic charges " for welfare foods , Mr. Powell said that all these
0 0000 000foods would still be free in families receiving regular National Assistance grants .0003000
# that cannot continue without either ...
of development being limited or an 8-
# adjusting being made in financing . "
# The Government decided to adjust
# the financing - which Mr. Powell
accompanied was underpinning - not
# undermining the service . Answering the
# attack on " economic charges " for welfare
0 0
Categories 1921 births2021 deaths20th century American lawyersAmerican lawyers of the United States Supreme Court of Representative
Categories 1882 births1953 deathsAmerican male sailors of the United States Navy officersAmerican male sailors of the United States
1957 British film director and director of the British Parliament of the South Carolina Housewives of the Order of
1999 2000 2001 2002 2003 2003 20042005 2006 2007 2008 2009 2010 2011 2012 2013
1942 1943 1944 1946 1947 1948 1949 1951 1951 1952 1953 1954 1954 19541965 66
1944 British film posterDirected byWilliam BurdettScreenplay byWilliam BurdettScreenplay byWilliam BurdettScreenplay byWilliam Burd

✅ Finished processing the image.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```python
from google.colab import files
import cv2
import numpy as np
from PIL import Image
import torch
from transformers import TrOCRProcessor, VisionEncoderDecoderModel
import os

# Load model and processor (reusing if already loaded)
try:
    processor
    model
except NameError:
    print("Loading TrOCR model and processor...")
    processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-handwritten')
    model = VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-base-handwritten')

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
print("✅ Model loaded and on device.")

# Preprocessing function to detect lines
def preprocess_image_for_lines(image_path):
    img = cv2.imread(image_path)
    if img is None:
        raise ValueError(f"Image not loaded: {image_path}")

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
        # Using OTSU's thresholding
        thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]

        # Horizontal line projection to find line regions
        horizontal_sum = np.sum(thresh, axis=1)
        lines = []
        start = None

        for i, val in enumerate(horizontal_sum):
            if val > 0 and start is None:
                start = i
            elif val == 0 and start is not None:
                end = i
                # Filter short noise lines, adjust threshold (e.g., 15) if needed
                if end - start > 15: # You can adjust this threshold
                    lines.append((start, end))
                start = None
        # Add the last line if the image doesn't end with a zero sum
        if start is not None:
            if len(horizontal_sum) - start > 15: # Check last segment length
                 lines.append((start, len(horizontal_sum)))


        return img, lines

    # Prediction function for lines
    def predict_text_from_lines(image, lines, processor, model, device):
        predicted_lines = []
        for y1, y2 in lines:
            line_img = image[y1:y2, :]
            # Ensure the line image is valid before processing
            if line_img.shape[0] > 0 and line_img.shape[1] > 0:
                # Convert to grayscale and add a channel dimension
                gray_line_img = cv2.cvtColor(line_img, cv2.COLOR_BGR2GRAY)
                # Add a channel dimension to the grayscale image
                gray_line_img_3d = np.expand_dims(gray_line_img, axis=-1)
                # Convert to RGB format with 3 identical channels for the processor
                pil_img = Image.fromarray(cv2.cvtColor(gray_line_img_3d, cv2.COLOR_GRAY2RGB))
                try:
                    pixel_values = processor(images=pil_img, return_tensors="pt").pixel_values.to(device)
                    # Beam decoding for accuracy
                    generated_ids = model.generate(pixel_values, max_length=128, num_beams=5, early_stopping=True)
                    predicted_text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
                    predicted_lines.append(predicted_text.strip())
                except Exception as e:
                    predicted_lines.append(f"[OCR ERROR] {e}")
            else:
                predicted_lines.append("") # Append empty string for invalid line images


        return "\n".join(predicted_lines) # Join lines with newline for better readability

    # Upload a single image
    print("Please upload a single image:")
    uploaded = files.upload()

    if len(uploaded) == 0:
        print("❌ No file uploaded.")
    elif len(uploaded) > 1:
         print(f"⚠️ You uploaded {len(uploaded)} files. Expected 1 image. Processing the first uploaded file.")
         filename = list(uploaded.keys())[0]
    else:
        filename = list(uploaded.keys())[0]


    # Process the uploaded image
    if filename:
        print(f"\n▶️ Processing: {filename}...")
        try:
            # Save the uploaded file temporarily to process with OpenCV
            with open(filename, 'wb') as f:
                f.write(uploaded[filename])

            image, lines = preprocess_image_for_lines(filename)
            print(f"✅ Detected {len(lines)} lines.")
            if lines:
                # Pass model and processor to the prediction function
```

```
            predicted_text = predict_text_from_lines(image, lines, processor, model, device)
            print("\n📝 Predicted Text:")
            print(predicted_text)
        else:
            print("No lines detected in this image that meet the size threshold.")

        # Clean up the temporary file
        os.remove(filename)

    except Exception as e:
        print(f"❌ Error processing {filename}: {e}")

print("\n✅ Finished processing the image.")
```

```
✅ Model loaded and on device.
Please upload a single image:
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving a01-000x.png to a01-000x (2).png

▶️ Processing: a01-000x (2).png...
✅ Detected 12 lines.

📝 Predicted Text:
10, Sentence Database C02-0005.0005000,000,0005000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,
# ... A MOVE to stop Mr. Gaitskell from nominating any more Labour life Peers is to
1942 1944 he made at a meeting of Labour 0MPs tomorrow . Mr. Michael Foot has put down 1950 1951
# a resolution on the subject and he is to be backed by Mr. Will Griffiths . MP for 200,000
1962 Manchester Exchange , 1982 1983 1985 1986 1987
# A MOVE to stop . Mr. Gaitskell from nominating
" any more Labour life Peers is to be made at a
# meeting of Labour MPs tomorrow . Mr. Michael ,
# Foot has put down a resolution on the subject ...
# and he is to be backed by Mr. Will Griffith ,
# UMP for Manchester Exchange .
a member of the National Register of Historic Places listings in the National Register of Historic Places is a
```

```
from google.colab import files
import cv2
import numpy as np
from PIL import Image
import torch
from transformers import TrOCRProcessor, VisionEncoderDecoderModel
import os

# Load model and processor (reusing if already loaded)
try:
    processor
    model
except NameError:
    print("Loading TrOCR model and processor...")
    processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-handwritten')
    model = VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-base-handwritten')

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
print("✅ Model loaded and on device.")

# Preprocessing function to detect lines
def preprocess_image_for_lines(image_path):
    img = cv2.imread(image_path)
    if img is None:
        raise ValueError(f"Image not loaded: {image_path}")

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Using OTSU's thresholding
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]

    # Horizontal line projection to find line regions
    horizontal_sum = np.sum(thresh, axis=1)
    lines = []
    start = None

    for i, val in enumerate(horizontal_sum):
        if val > 0 and start is None:
            start = i
        elif val == 0 and start is not None:
```

```
                end = i
                # Filter short noise lines, adjust threshold (e.g., 15) if needed
                if end - start > 15: # You can adjust this threshold
                    lines.append((start, end))
                start = None
        # Add the last line if the image doesn't end with a zero sum
        if start is not None:
            if len(horizontal_sum) - start > 15: # Check last segment length
                 lines.append((start, len(horizontal_sum)))


        return img, lines

    # Prediction function for lines
    def predict_text_from_lines(image, lines, processor, model, device):
        predicted_lines = []
        for y1, y2 in lines:
            line_img = image[y1:y2, :]
            # Ensure the line image is valid before processing
            if line_img.shape[0] > 0 and line_img.shape[1] > 0:
                # Convert to grayscale and add a channel dimension
                gray_line_img = cv2.cvtColor(line_img, cv2.COLOR_BGR2GRAY)
                # Add a channel dimension to the grayscale image
                gray_line_img_3d = np.expand_dims(gray_line_img, axis=-1)
                # Convert to RGB format with 3 identical channels for the processor
                pil_img = Image.fromarray(cv2.cvtColor(gray_line_img_3d, cv2.COLOR_GRAY2RGB))
                try:
                    pixel_values = processor(images=pil_img, return_tensors="pt").pixel_values.to(device)
                    # Beam decoding for accuracy
                    generated_ids = model.generate(pixel_values, max_length=128, num_beams=5, early_stopping=True)
                    predicted_text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
                    predicted_lines.append(predicted_text.strip())
                except Exception as e:
                    predicted_lines.append(f"[OCR ERROR] {e}")
            else:
                predicted_lines.append("") # Append empty string for invalid line images


        return "\n".join(predicted_lines) # Join lines with newline for better readability

    # Upload a single image
    print("Please upload a single image:")
    uploaded = files.upload()

    if len(uploaded) == 0:
        print("❌ No file uploaded.")
    elif len(uploaded) > 1:
         print(f"⚠️ You uploaded {len(uploaded)} files. Expected 1 image. Processing the first uploaded file.")
         filename = list(uploaded.keys())[0]
    else:
        filename = list(uploaded.keys())[0]


    # Process the uploaded image
    if filename:
        print(f"\n▶️ Processing: {filename}...")
        try:
            # Save the uploaded file temporarily to process with OpenCV
            with open(filename, 'wb') as f:
                f.write(uploaded[filename])

            image, lines = preprocess_image_for_lines(filename)
            print(f"✅ Detected {len(lines)} lines.")
            if lines:
                # Pass model and processor to the prediction function
                predicted_text = predict_text_from_lines(image, lines, processor, model, device)
                print("\n📝 Predicted Text:")
                print(predicted_text)
            else:
                print("No lines detected in this image that meet the size threshold.")

            # Clean up the temporary file
            os.remove(filename)

        except Exception as e:
            print(f"❌ Error processing {filename}: {e}")
```

```
print("\n✅ Finished processing the image.")
```

```
✅ Model loaded and on device.
Please upload a single image:
Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving a01-132x.png to a01-132x.png

▶️ Processing: a01-132x.png...
✅ Detected 16 lines.

📝 Predicted Text:
60- Sentence Database.0005.0005. AOL-1320005
# that continue without either development being limited on adjustment ...
# been being made in financing . " The Government decided to adjust the financing - which
# ... Mr. Powell claimed was underpinning - not undermining - the service . Answering ... ...
# that the attack on " economic charges " for welfare foods , Mr. Powell said that all these
0 0000 000foods would still be free in families receiving regular National Assistance grants .00030003000
" That cannot continue without either development ...
# being limited or an adjustment being made ...
# in financing . " The Government decided to adjust
# the financing - which Mr. Powell claimed was
# underpinning - not undermining - the service ,
# Answering the attack on " economic charges "
# for welfare , Mr. Powell said that all these
# foods would still be free in families receiving
# regular National Assistance grants ,
a b c d e f. Nature of the South Carolina Department of the American Government of Australia from the
```

```
from google.colab import files
import cv2
import numpy as np
from PIL import Image
import torch
from transformers import TrOCRProcessor, VisionEncoderDecoderModel
import os

# Load model and processor (reusing if already loaded)
try:
    processor
    model
except NameError:
    print("Loading TrOCR model and processor...")
    processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-handwritten')
    model = VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-base-handwritten')

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
print("✅ Model loaded and on device.")

# Preprocessing function to detect lines
def preprocess_image_for_lines(image_path):
    img = cv2.imread(image_path)
    if img is None:
        raise ValueError(f"Image not loaded: {image_path}")

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Using OTSU's thresholding
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]

    # Horizontal line projection to find line regions
    horizontal_sum = np.sum(thresh, axis=1)
    lines = []
    start = None

    for i, val in enumerate(horizontal_sum):
        if val > 0 and start is None:
            start = i
        elif val == 0 and start is not None:
            end = i
            # Filter short noise lines, adjust threshold (e.g., 15) if needed
            if end - start > 15: # You can adjust this threshold
                lines.append((start, end))
            start = None
    # Add the last line if the image doesn't end with a zero sum
    if start is not None:
```

```
            if len(horizontal_sum) - start > 15: # Check last segment length
                 lines.append((start, len(horizontal_sum)))


        return img, lines

    # Prediction function for lines
    def predict_text_from_lines(image, lines, processor, model, device):
        predicted_lines = []
        for y1, y2 in lines:
            line_img = image[y1:y2, :]
            # Ensure the line image is valid before processing
            if line_img.shape[0] > 0 and line_img.shape[1] > 0:
                # Convert to grayscale and add a channel dimension
                gray_line_img = cv2.cvtColor(line_img, cv2.COLOR_BGR2GRAY)
                # Add a channel dimension to the grayscale image
                gray_line_img_3d = np.expand_dims(gray_line_img, axis=-1)
                # Convert to RGB format with 3 identical channels for the processor
                pil_img = Image.fromarray(cv2.cvtColor(gray_line_img_3d, cv2.COLOR_GRAY2RGB))
                try:
                    pixel_values = processor(images=pil_img, return_tensors="pt").pixel_values.to(device)
                    # Beam decoding for accuracy
                    generated_ids = model.generate(pixel_values, max_length=128, num_beams=5, early_stopping=True)
                    predicted_text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
                    predicted_lines.append(predicted_text.strip())
                except Exception as e:
                    predicted_lines.append(f"[OCR ERROR] {e}")
            else:
                predicted_lines.append("") # Append empty string for invalid line images


        return "\n".join(predicted_lines) # Join lines with newline for better readability

    # Upload a single image
    print("Please upload a single image:")
    uploaded = files.upload()

    if len(uploaded) == 0:
        print("❌ No file uploaded.")
    elif len(uploaded) > 1:
         print(f"⚠️ You uploaded {len(uploaded)} files. Expected 1 image. Processing the first uploaded file.")
         filename = list(uploaded.keys())[0]
    else:
        filename = list(uploaded.keys())[0]


    # Process the uploaded image
    if filename:
        print(f"\n▶️ Processing: {filename}...")
        try:
            # Save the uploaded file temporarily to process with OpenCV
            with open(filename, 'wb') as f:
                f.write(uploaded[filename])

            image, lines = preprocess_image_for_lines(filename)
            print(f"✅ Detected {len(lines)} lines.")
            if lines:
                # Pass model and processor to the prediction function
                predicted_text = predict_text_from_lines(image, lines, processor, model, device)
                print("\n📝 Predicted Text:")
                print(predicted_text)
            else:
                print("No lines detected in this image that meet the size threshold.")

            # Clean up the temporary file
            os.remove(filename)

        except Exception as e:
            print(f"❌ Error processing {filename}: {e}")

    print("\n✅ Finished processing the image.")
```

```
✅ Model loaded and on device.
Please upload a single image:
```
Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
```
Saving a01-072x.png to a01-072x.png

▶️ Processing: a01-072x.png...
✅ Detected 15 lines.

📝 Predicted Text:
2 . Sentence Database.0005.0005. AOL-072.
1942 1953 These support costs are a big drain on America's dollar reserves . Dr. Adenauer's 1953 1956
1800 1899 answer is the once-and-all cash offer of 35million . President Kennedy's rejection 1900 1900
# 100 of it is a painful blow to the West German Government . It will now have to pay ... .
# more - and increase taxation to do so - or run the obvious risks in upsetting the new
0 0000 American administration .0003000500050005000
" There support costs are a big draw on America's
# better revenues . Dr. Adenauer's answer in the 1960s .
# once - and - for - all cash offer of 357million . President #
# Kennedy's rejection of it is a painful blow to the 1/2
# that German Government . It will now have to pay 1 000
# more - and increase taxation to do so - or # ...
# even the obvious risks in upsetting the war ...
t. American administration , 2000 2001
a b c d e f g h.
```

```python
from google.colab import files
import cv2
import numpy as np
from PIL import Image
import torch
from transformers import TrOCRProcessor, VisionEncoderDecoderModel
import os

# Load model and processor (reusing if already loaded)
try:
    processor
    model
except NameError:
    print("Loading TrOCR model and processor...")
    processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-handwritten')
    model = VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-base-handwritten')

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
print("✅ Model loaded and on device.")

# Preprocessing function to detect lines
def preprocess_image_for_lines(image_path):
    img = cv2.imread(image_path)
    if img is None:
        raise ValueError(f"Image not loaded: {image_path}")

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Using OTSU's thresholding
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]

    # Horizontal line projection to find line regions
    horizontal_sum = np.sum(thresh, axis=1)
    lines = []
    start = None

    for i, val in enumerate(horizontal_sum):
        if val > 0 and start is None:
            start = i
        elif val == 0 and start is not None:
            end = i
            # Filter short noise lines, adjust threshold (e.g., 15) if needed
            if end - start > 15: # You can adjust this threshold
                lines.append((start, end))
            start = None
    # Add the last line if the image doesn't end with a zero sum
    if start is not None:
        if len(horizontal_sum) - start > 15: # Check last segment length
             lines.append((start, len(horizontal_sum)))
```

```
    return img, lines

# Prediction function for lines
def predict_text_from_lines(image, lines, processor, model, device):
    predicted_lines = []
    for y1, y2 in lines:
        line_img = image[y1:y2, :]
        # Ensure the line image is valid before processing
        if line_img.shape[0] > 0 and line_img.shape[1] > 0:
            # Convert to grayscale and add a channel dimension
            gray_line_img = cv2.cvtColor(line_img, cv2.COLOR_BGR2GRAY)
            # Add a channel dimension to the grayscale image
            gray_line_img_3d = np.expand_dims(gray_line_img, axis=-1)
            # Convert to RGB format with 3 identical channels for the processor
            pil_img = Image.fromarray(cv2.cvtColor(gray_line_img_3d, cv2.COLOR_GRAY2RGB))
            try:
                pixel_values = processor(images=pil_img, return_tensors="pt").pixel_values.to(device)
                # Beam decoding for accuracy
                generated_ids = model.generate(pixel_values, max_length=128, num_beams=5, early_stopping=True)
                predicted_text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
                predicted_lines.append(predicted_text.strip())
            except Exception as e:
                predicted_lines.append(f"[OCR ERROR] {e}")
        else:
            predicted_lines.append("") # Append empty string for invalid line images


    return "\n".join(predicted_lines) # Join lines with newline for better readability

# Upload a single image
print("Please upload a single image:")
uploaded = files.upload()

if len(uploaded) == 0:
    print("❌ No file uploaded.")
elif len(uploaded) > 1:
     print(f"⚠️ You uploaded {len(uploaded)} files. Expected 1 image. Processing the first uploaded file.")
     filename = list(uploaded.keys())[0]
else:
    filename = list(uploaded.keys())[0]


# Process the uploaded image
if filename:
    print(f"\n▶️ Processing: {filename}...")
    try:
        # Save the uploaded file temporarily to process with OpenCV
        with open(filename, 'wb') as f:
            f.write(uploaded[filename])

        image, lines = preprocess_image_for_lines(filename)
        print(f"✅ Detected {len(lines)} lines.")
        if lines:
            # Pass model and processor to the prediction function
            predicted_text = predict_text_from_lines(image, lines, processor, model, device)
            print("\n📝 Predicted Text:")
            print(predicted_text)
        else:
            print("No lines detected in this image that meet the size threshold.")

        # Clean up the temporary file
        os.remove(filename)

    except Exception as e:
        print(f"❌ Error processing {filename}: {e}")

print("\n✅ Finished processing the image.")
```

```
✅ Model loaded and on device.
Please upload a single image:
```
Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
```
Saving p100.jpg to p100.jpg

▶️ Processing: p100.jpg...
✅ Detected 2 lines.

📝 Predicted Text:
dispersed in the Yellow Belt
trourellerine and for stock
```

```python
from google.colab import files
import cv2
import numpy as np
from PIL import Image
import torch
from transformers import TrOCRProcessor, VisionEncoderDecoderModel
import os

# Load model and processor (reusing if already loaded)
try:
    processor
    model
except NameError:
    print("Loading TrOCR model and processor...")
    processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-handwritten')
    model = VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-base-handwritten')

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
print("✅ Model loaded and on device.")

# Preprocessing function to detect lines
def preprocess_image_for_lines(image_path):
    img = cv2.imread(image_path)
    if img is None:
        raise ValueError(f"Image not loaded: {image_path}")

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Using OTSU's thresholding
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]

    # Horizontal line projection to find line regions
    horizontal_sum = np.sum(thresh, axis=1)
    lines = []
    start = None

    for i, val in enumerate(horizontal_sum):
        if val > 0 and start is None:
            start = i
        elif val == 0 and start is not None:
            end = i
            # Filter short noise lines, adjust threshold (e.g., 15) if needed
            if end - start > 15: # You can adjust this threshold
                lines.append((start, end))
            start = None
    # Add the last line if the image doesn't end with a zero sum
    if start is not None:
        if len(horizontal_sum) - start > 15: # Check last segment length
             lines.append((start, len(horizontal_sum)))


    return img, lines

# Prediction function for lines
def predict_text_from_lines(image, lines, processor, model, device):
    predicted_lines = []
    for y1, y2 in lines:
        line_img = image[y1:y2, :]
        # Ensure the line image is valid before processing
        if line_img.shape[0] > 0 and line_img.shape[1] > 0:
            # Convert to grayscale and add a channel dimension
            gray_line_img = cv2.cvtColor(line_img, cv2.COLOR_BGR2GRAY)
            # Add a channel dimension to the grayscale image
```

```
            gray_line_img_3d = np.expand_dims(gray_line_img, axis=-1)
            # Convert to RGB format with 3 identical channels for the processor
            pil_img = Image.fromarray(cv2.cvtColor(gray_line_img_3d, cv2.COLOR_GRAY2RGB))
            try:
                pixel_values = processor(images=pil_img, return_tensors="pt").pixel_values.to(device)
                # Beam decoding for accuracy
                generated_ids = model.generate(pixel_values, max_length=128, num_beams=5, early_stopping=True)
                predicted_text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
                predicted_lines.append(predicted_text.strip())
            except Exception as e:
                predicted_lines.append(f"[OCR ERROR] {e}")
        else:
            predicted_lines.append("") # Append empty string for invalid line images


    return "\n".join(predicted_lines) # Join lines with newline for better readability

# Upload a single image
print("Please upload a single image:")
uploaded = files.upload()

if len(uploaded) == 0:
    print("❌ No file uploaded.")
elif len(uploaded) > 1:
     print(f"⚠️ You uploaded {len(uploaded)} files. Expected 1 image. Processing the first uploaded file.")
     filename = list(uploaded.keys())[0]
else:
    filename = list(uploaded.keys())[0]


# Process the uploaded image
if filename:
    print(f"\n▶️ Processing: {filename}...")
    try:
        # Save the uploaded file temporarily to process with OpenCV
        with open(filename, 'wb') as f:
            f.write(uploaded[filename])

        image, lines = preprocess_image_for_lines(filename)
        print(f"✅ Detected {len(lines)} lines.")
        if lines:
            # Pass model and processor to the prediction function
            predicted_text = predict_text_from_lines(image, lines, processor, model, device)
            print("\n📝 Predicted Text:")
            print(predicted_text)
        else:
            print("No lines detected in this image that meet the size threshold.")

        # Clean up the temporary file
        os.remove(filename)

    except Exception as e:
        print(f"❌ Error processing {filename}: {e}")

print("\n✅ Finished processing the image.")
```

```
✅ Model loaded and on device.
Please upload a single image:
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving p71.jpg to p71.jpg

▶️ Processing: p71.jpg...
✅ Detected 8 lines.

📝 Predicted Text:
2 1 Travel -
I enjoy travelling .
. Visiting new places .
experiencing different .
cultures , and try
local cuisine . The
excitement of future .
trips .000
```

```
from google.colab import files
import cv2
```

```
import numpy as np
from PIL import Image
import torch
from transformers import TrOCRProcessor, VisionEncoderDecoderModel
import os

# Load model and processor (reusing if already loaded)
try:
    processor
    model
except NameError:
    print("Loading TrOCR model and processor...")
    processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-handwritten')
    model = VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-base-handwritten')

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
print("✅ Model loaded and on device.")

# Preprocessing function to detect lines
def preprocess_image_for_lines(image_path):
    img = cv2.imread(image_path)
    if img is None:
        raise ValueError(f"Image not loaded: {image_path}")

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Using OTSU's thresholding
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]

    # Horizontal line projection to find line regions
    horizontal_sum = np.sum(thresh, axis=1)
    lines = []
    start = None

    for i, val in enumerate(horizontal_sum):
        if val > 0 and start is None:
            start = i
        elif val == 0 and start is not None:
            end = i
            # Filter short noise lines, adjust threshold (e.g., 15) if needed
            if end - start > 15: # You can adjust this threshold
                lines.append((start, end))
            start = None
    # Add the last line if the image doesn't end with a zero sum
    if start is not None:
        if len(horizontal_sum) - start > 15: # Check last segment length
             lines.append((start, len(horizontal_sum)))


    return img, lines

# Prediction function for lines
def predict_text_from_lines(image, lines, processor, model, device):
    predicted_lines = []
    for y1, y2 in lines:
        line_img = image[y1:y2, :]
        # Ensure the line image is valid before processing
        if line_img.shape[0] > 0 and line_img.shape[1] > 0:
            # Convert to grayscale and add a channel dimension
            gray_line_img = cv2.cvtColor(line_img, cv2.COLOR_BGR2GRAY)
            # Add a channel dimension to the grayscale image
            gray_line_img_3d = np.expand_dims(gray_line_img, axis=-1)
            # Convert to RGB format with 3 identical channels for the processor
            pil_img = Image.fromarray(cv2.cvtColor(gray_line_img_3d, cv2.COLOR_GRAY2RGB))
            try:
                pixel_values = processor(images=pil_img, return_tensors="pt").pixel_values.to(device)
                # Beam decoding for accuracy
                generated_ids = model.generate(pixel_values, max_length=128, num_beams=5, early_stopping=True)
                predicted_text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
                predicted_lines.append(predicted_text.strip())
            except Exception as e:
                predicted_lines.append(f"[OCR ERROR] {e}")
        else:
            predicted_lines.append("") # Append empty string for invalid line images


    return "\n".join(predicted_lines) # Join lines with newline for better readability
```

```
# Upload a single image
print("Please upload a single image:")
uploaded = files.upload()

if len(uploaded) == 0:
    print("❌ No file uploaded.")
elif len(uploaded) > 1:
     print(f"⚠️ You uploaded {len(uploaded)} files. Expected 1 image. Processing the first uploaded file.")
     filename = list(uploaded.keys())[0]
else:
    filename = list(uploaded.keys())[0]


# Process the uploaded image
if filename:
    print(f"\n▶️ Processing: {filename}...")
    try:
        # Save the uploaded file temporarily to process with OpenCV
        with open(filename, 'wb') as f:
            f.write(uploaded[filename])

        image, lines = preprocess_image_for_lines(filename)
        print(f"✅ Detected {len(lines)} lines.")
        if lines:
            # Pass model and processor to the prediction function
            predicted_text = predict_text_from_lines(image, lines, processor, model, device)
            print("\n📝 Predicted Text:")
            print(predicted_text)
        else:
            print("No lines detected in this image that meet the size threshold.")

        # Clean up the temporary file
        os.remove(filename)

    except Exception as e:
        print(f"❌ Error processing {filename}: {e}")

print("\n✅ Finished processing the image.")
```

```
✅ Model loaded and on device.
Please upload a single image:
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving p32.jpg to p32.jpg

▶️ Processing: p32.jpg...
✅ Detected 4 lines.

📝 Predicted Text:
# Never rush for a position .
6 Further information
# your knowledge and politeness ,
# not for your designation .
```

```
from google.colab import files
import cv2
import numpy as np
from PIL import Image
import torch
from transformers import TrOCRProcessor, VisionEncoderDecoderModel
import os

# Load model and processor (reusing if already loaded)
try:
    processor
    model
except NameError:
    print("Loading TrOCR model and processor...")
    processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-handwritten')
    model = VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-base-handwritten')

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
print("✅ Model loaded and on device.")
```

```
# Preprocessing function to detect lines
def preprocess_image_for_lines(image_path):
    img = cv2.imread(image_path)
    if img is None:
        raise ValueError(f"Image not loaded: {image_path}")

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Using OTSU's thresholding
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]

    # Horizontal line projection to find line regions
    horizontal_sum = np.sum(thresh, axis=1)
    lines = []
    start = None

    for i, val in enumerate(horizontal_sum):
        if val > 0 and start is None:
            start = i
        elif val == 0 and start is not None:
            end = i
            # Filter short noise lines, adjust threshold (e.g., 15) if needed
            if end - start > 15: # You can adjust this threshold
                lines.append((start, end))
            start = None
    # Add the last line if the image doesn't end with a zero sum
    if start is not None:
        if len(horizontal_sum) - start > 15: # Check last segment length
             lines.append((start, len(horizontal_sum)))


    return img, lines

# Prediction function for lines
def predict_text_from_lines(image, lines, processor, model, device):
    predicted_lines = []
    for y1, y2 in lines:
        line_img = image[y1:y2, :]
        # Ensure the line image is valid before processing
        if line_img.shape[0] > 0 and line_img.shape[1] > 0:
            # Convert to grayscale and add a channel dimension
            gray_line_img = cv2.cvtColor(line_img, cv2.COLOR_BGR2GRAY)
            # Add a channel dimension to the grayscale image
            gray_line_img_3d = np.expand_dims(gray_line_img, axis=-1)
            # Convert to RGB format with 3 identical channels for the processor
            pil_img = Image.fromarray(cv2.cvtColor(gray_line_img_3d, cv2.COLOR_GRAY2RGB))
            try:
                pixel_values = processor(images=pil_img, return_tensors="pt").pixel_values.to(device)
                # Beam decoding for accuracy
                generated_ids = model.generate(pixel_values, max_length=128, num_beams=5, early_stopping=True)
                predicted_text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
                predicted_lines.append(predicted_text.strip())
            except Exception as e:
                predicted_lines.append(f"[OCR ERROR] {e}")
        else:
            predicted_lines.append("") # Append empty string for invalid line images


    return "\n".join(predicted_lines) # Join lines with newline for better readability

# Upload a single image
print("Please upload a single image:")
uploaded = files.upload()

if len(uploaded) == 0:
    print("❌ No file uploaded.")
elif len(uploaded) > 1:
     print(f"⚠️ You uploaded {len(uploaded)} files. Expected 1 image. Processing the first uploaded file.")
     filename = list(uploaded.keys())[0]
else:
    filename = list(uploaded.keys())[0]


# Process the uploaded image
if filename:
    print(f"\n▶️ Processing: {filename}...")
    try:
        # Save the uploaded file temporarily to process with OpenCV
```

```
            with open(filename, 'wb') as f:
                f.write(uploaded[filename])

            image, lines = preprocess_image_for_lines(filename)
            print(f"✅ Detected {len(lines)} lines.")
            if lines:
                # Pass model and processor to the prediction function
                predicted_text = predict_text_from_lines(image, lines, processor, model, device)
                print("\n📝 Predicted Text:")
                print(predicted_text)
            else:
                print("No lines detected in this image that meet the size threshold.")

            # Clean up the temporary file
            os.remove(filename)

        except Exception as e:
            print(f"❌ Error processing {filename}: {e}")

    print("\n✅ Finished processing the image.")
```

```
✅ Model loaded and on device.
Please upload a single image:
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving p29.jpg to p29.jpg

▶️ Processing: p29.jpg...
✅ Detected 2 lines.

📝 Predicted Text:
" Fashion is also related to the
probably fashion . Orlando land-disclosed
```

```
from google.colab import files
import cv2
import numpy as np
from PIL import Image
import torch
from transformers import TrOCRProcessor, VisionEncoderDecoderModel
import os

# Load model and processor (reusing if already loaded)
try:
    processor
    model
except NameError:
    print("Loading TrOCR model and processor...")
    processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-handwritten')
    model = VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-base-handwritten')

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
print("✅ Model loaded and on device.")

# Preprocessing function to detect lines
def preprocess_image_for_lines(image_path):
    img = cv2.imread(image_path)
    if img is None:
        raise ValueError(f"Image not loaded: {image_path}")

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Using OTSU's thresholding
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]

    # Horizontal line projection to find line regions
    horizontal_sum = np.sum(thresh, axis=1)
    lines = []
    start = None

    for i, val in enumerate(horizontal_sum):
        if val > 0 and start is None:
            start = i
        elif val == 0 and start is not None:
            end = i
            # Filter short noise lines, adjust threshold (e.g., 15) if needed
```

```
            if end - start > 15: # You can adjust this threshold
                lines.append((start, end))
            start = None
    # Add the last line if the image doesn't end with a zero sum
    if start is not None:
        if len(horizontal_sum) - start > 15: # Check last segment length
             lines.append((start, len(horizontal_sum)))


    return img, lines

# Prediction function for lines
def predict_text_from_lines(image, lines, processor, model, device):
    predicted_lines = []
    for y1, y2 in lines:
        line_img = image[y1:y2, :]
        # Ensure the line image is valid before processing
        if line_img.shape[0] > 0 and line_img.shape[1] > 0:
            # Convert to grayscale and add a channel dimension
            gray_line_img = cv2.cvtColor(line_img, cv2.COLOR_BGR2GRAY)
            # Add a channel dimension to the grayscale image
            gray_line_img_3d = np.expand_dims(gray_line_img, axis=-1)
            # Convert to RGB format with 3 identical channels for the processor
            pil_img = Image.fromarray(cv2.cvtColor(gray_line_img_3d, cv2.COLOR_GRAY2RGB))
            try:
                pixel_values = processor(images=pil_img, return_tensors="pt").pixel_values.to(device)
                # Beam decoding for accuracy
                generated_ids = model.generate(pixel_values, max_length=128, num_beams=5, early_stopping=True)
                predicted_text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
                predicted_lines.append(predicted_text.strip())
            except Exception as e:
                predicted_lines.append(f"[OCR ERROR] {e}")
        else:
            predicted_lines.append("") # Append empty string for invalid line images


    return "\n".join(predicted_lines) # Join lines with newline for better readability

# Upload a single image
print("Please upload a single image:")
uploaded = files.upload()

if len(uploaded) == 0:
    print("❌ No file uploaded.")
elif len(uploaded) > 1:
     print(f"⚠️ You uploaded {len(uploaded)} files. Expected 1 image. Processing the first uploaded file.")
     filename = list(uploaded.keys())[0]
else:
    filename = list(uploaded.keys())[0]


# Process the uploaded image
if filename:
    print(f"\n▶️ Processing: {filename}...")
    try:
        # Save the uploaded file temporarily to process with OpenCV
        with open(filename, 'wb') as f:
            f.write(uploaded[filename])

        image, lines = preprocess_image_for_lines(filename)
        print(f"✅ Detected {len(lines)} lines.")
        if lines:
            # Pass model and processor to the prediction function
            predicted_text = predict_text_from_lines(image, lines, processor, model, device)
            print("\n📝 Predicted Text:")
            print(predicted_text)
        else:
            print("No lines detected in this image that meet the size threshold.")

        # Clean up the temporary file
        os.remove(filename)

    except Exception as e:
        print(f"❌ Error processing {filename}: {e}")

print("\n✅ Finished processing the image.")
```

```
✅ Model loaded and on device.
Please upload a single image:
```
Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
```
Saving p37.jpg to p37.jpg

▶️ Processing: p37.jpg...
✅ Detected 8 lines.

📝 Predicted Text:
Not logged inTalkContributionsCreate accountLog in
What links here has multiple issues Please help improve it or discuss these issues on
Sometimes all you can do is not think .
# not wonder , not imagine , not obsess ...
must breathe , and have faith that
everything will work out for the best-
0 1
0 2
```

```python
from google.colab import files
import cv2
import numpy as np
from PIL import Image
import torch
from transformers import TrOCRProcessor, VisionEncoderDecoderModel
import os

# Load model and processor (reusing if already loaded)
try:
    processor
    model
except NameError:
    print("Loading TrOCR model and processor...")
    processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-handwritten')
    model = VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-base-handwritten')

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
print("✅ Model loaded and on device.")

# Preprocessing function to detect lines
def preprocess_image_for_lines(image_path):
    img = cv2.imread(image_path)
    if img is None:
        raise ValueError(f"Image not loaded: {image_path}")

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Using OTSU's thresholding
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]

    # Horizontal line projection to find line regions
    horizontal_sum = np.sum(thresh, axis=1)
    lines = []
    start = None

    for i, val in enumerate(horizontal_sum):
        if val > 0 and start is None:
            start = i
        elif val == 0 and start is not None:
            end = i
            # Filter short noise lines, adjust threshold (e.g., 15) if needed
            if end - start > 15: # You can adjust this threshold
                lines.append((start, end))
            start = None
    # Add the last line if the image doesn't end with a zero sum
    if start is not None:
        if len(horizontal_sum) - start > 15: # Check last segment length
             lines.append((start, len(horizontal_sum)))


    return img, lines

# Prediction function for lines
def predict_text_from_lines(image, lines, processor, model, device):
    predicted_lines = []
    for y1, y2 in lines:
```

```
            line_img = image[y1:y2, :]
            # Ensure the line image is valid before processing
            if line_img.shape[0] > 0 and line_img.shape[1] > 0:
                # Convert to grayscale and add a channel dimension
                gray_line_img = cv2.cvtColor(line_img, cv2.COLOR_BGR2GRAY)
                # Add a channel dimension to the grayscale image
                gray_line_img_3d = np.expand_dims(gray_line_img, axis=-1)
                # Convert to RGB format with 3 identical channels for the processor
                pil_img = Image.fromarray(cv2.cvtColor(gray_line_img_3d, cv2.COLOR_GRAY2RGB))
                try:
                    pixel_values = processor(images=pil_img, return_tensors="pt").pixel_values.to(device)
                    # Beam decoding for accuracy
                    generated_ids = model.generate(pixel_values, max_length=128, num_beams=5, early_stopping=True)
                    predicted_text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
                    predicted_lines.append(predicted_text.strip())
                except Exception as e:
                    predicted_lines.append(f"[OCR ERROR] {e}")
            else:
                predicted_lines.append("") # Append empty string for invalid line images


    return "\n".join(predicted_lines) # Join lines with newline for better readability

# Upload a single image
print("Please upload a single image:")
uploaded = files.upload()

if len(uploaded) == 0:
    print("❌ No file uploaded.")
elif len(uploaded) > 1:
     print(f"⚠️ You uploaded {len(uploaded)} files. Expected 1 image. Processing the first uploaded file.")
     filename = list(uploaded.keys())[0]
else:
    filename = list(uploaded.keys())[0]


# Process the uploaded image
if filename:
    print(f"\n▶️ Processing: {filename}...")
    try:
        # Save the uploaded file temporarily to process with OpenCV
        with open(filename, 'wb') as f:
            f.write(uploaded[filename])

        image, lines = preprocess_image_for_lines(filename)
        print(f"✅ Detected {len(lines)} lines.")
        if lines:
            # Pass model and processor to the prediction function
            predicted_text = predict_text_from_lines(image, lines, processor, model, device)
            print("\n📝 Predicted Text:")
            print(predicted_text)
        else:
            print("No lines detected in this image that meet the size threshold.")

        # Clean up the temporary file
        os.remove(filename)

    except Exception as e:
        print(f"❌ Error processing {filename}: {e}")

print("\n✅ Finished processing the image.")
```

```
✅ Model loaded and on device.
Please upload a single image:
```
Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
```
Saving a05-062.png to a05-062.png

▶️ Processing: a05-062.png...
✅ Detected 17 lines.

📝 Predicted Text:
2 . Sentence Database.0005.0005.0005.0005.0005.
```

```
from google.colab import files
import cv2
import numpy as np
from PIL import Image
import torch
from transformers import TrOCRProcessor, VisionEncoderDecoderModel
import os

# Load model and processor (reusing if already loaded)
try:
    processor
    model
except NameError:
    print("Loading TrOCR model and processor...")
    processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-handwritten')
    model = VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-base-handwritten')

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
print("✅ Model loaded and on device.")

# Preprocessing function to detect lines
def preprocess_image_for_lines(image_path):
    img = cv2.imread(image_path)
    if img is None:
        raise ValueError(f"Image not loaded: {image_path}")

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Using OTSU's thresholding
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]

    # Horizontal line projection to find line regions
    horizontal_sum = np.sum(thresh, axis=1)
    lines = []
    start = None

    for i, val in enumerate(horizontal_sum):
        if val > 0 and start is None:
            start = i
        elif val == 0 and start is not None:
            end = i
            # Filter short noise lines, adjust threshold (e.g., 15) if needed
            if end - start > 15: # You can adjust this threshold
                lines.append((start, end))
            start = None
    # Add the last line if the image doesn't end with a zero sum
    if start is not None:
        if len(horizontal_sum) - start > 15: # Check last segment length
             lines.append((start, len(horizontal_sum)))


    return img, lines

# Prediction function for lines
def predict_text_from_lines(image, lines, processor, model, device):
    predicted_lines = []
    for y1, y2 in lines:
        line_img = image[y1:y2, :]
        # Ensure the line image is valid before processing
        if line_img.shape[0] > 0 and line_img.shape[1] > 0:
            # Convert to grayscale and add a channel dimension
            gray_line_img = cv2.cvtColor(line_img, cv2.COLOR_BGR2GRAY)
            # Add a channel dimension to the grayscale image
            gray_line_img_3d = np.expand_dims(gray_line_img, axis=-1)
```