3. Source Code Latin Aksara

```
import evaluate
import numpy as np
import huggingface_hub
import torch
import re
import json

from tokenizers import Tokenizer
from tokenizers.models import BPE

from datasets import load_dataset
from tokenizers import Tokenizer
from transformers import AutoTokenizer,
DataCollatorForSeq2Seq
from transformers import AutoModelForSeq2SeqLM,
Seq2SeqTrainingArguments, Seq2SeqTrainer
from accelerate import init_empty_weights,
load_checkpoint_and_dispatch

# Load dataset
books = load_dataset("dataset")

# Split dataset into train and test sets
books =
books["train"].train_test_split(test_size=0.2)

tokenizer =
AutoTokenizer.from_pretrained("./Model_v2")
model =
AutoModelForSeq2SeqLM.from_pretrained("./Model_v2")

# Define source and target languages
source_lang = "latin"
target_lang = "aksara"

# Define prefix for model input
prefix = "translate latin to aksara: "

def preprocess_function(examples):
    inputs = [prefix + example[source_lang] for
example in examples["translation"]]
    targets = [example[target_lang] for example in
examples["translation"]]
    model_inputs = tokenizer(inputs,
text_target=targets, max_length=128, truncation=True)
    return model_inputs
```

```
tokenized_books = books.map(preprocess_function,
batched=True)
data_collator =
DataCollatorForSeq2Seq(tokenizer=tokenizer,
model=model)

# Load evaluation metric
metric = evaluate.load("sacrebleu")

def postprocess_text(preds, labels):
    preds = [pred.strip() for pred in preds]
    labels = [[label.strip()] for label in labels]

    return preds, labels


def compute_metrics(eval_preds):
    preds, labels = eval_preds
    if isinstance(preds, tuple):
        preds = preds[0]
    decoded_preds = tokenizer.batch_decode(preds,
skip_special_tokens=True)

    labels = np.where(labels != -100, labels,
tokenizer.pad_token_id)
    decoded_labels = tokenizer.batch_decode(labels,
skip_special_tokens=True)

    decoded_preds, decoded_labels =
postprocess_text(decoded_preds, decoded_labels)

    result =
metric.compute(predictions=decoded_preds,
references=decoded_labels)
    result = {"bleu": result["score"]}

    prediction_lens = [np.count_nonzero(pred !=
tokenizer.pad_token_id) for pred in preds]
    result["gen_len"] = np.mean(prediction_lens)
    result = {k: round(v, 4) for k, v in
result.items()}
    return result

# Define training arguments
training_args = Seq2SeqTrainingArguments(
    output_dir="Model",
    group_by_length=True,
    evaluation_strategy="epoch",
```

```
    learning_rate=2e-5,
    per_device_train_batch_size=2,
    weight_decay=0.01,
    save_total_limit=1,
    warmup_steps=100,
    num_train_epochs=20,
    predict_with_generate=True,
    fp16=True,
    optim="adamw_torch"
)

trainer = Seq2SeqTrainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_books["train"],
    eval_dataset=tokenized_books["test"],
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

trainer.train()
trainer.save_model()
trainer.save_state()
```

4. Source Code Speech To Text

```
     import os
     import re
     import time
     import json
     import jiwer
     import torch
     import random
     import librosa
     import evaluate
     import datasets
     import torchaudio
     import numpy as np
     import pandas as pd
     import lightning as L
     import IPython.display as ipd

     from pathlib import Path
     from datetime import datetime
     from IPython.display import Audio
     from nltk.translate import bleu_score
```

```
    from dataclasses import dataclass, field
    from IPython.display import display, HTML
    from torch.utils.data import Dataset, DataLoader
    from typing import Any, Dict, List, Optional,
    Union
    from sklearn.model_selection import
    train_test_split
    from datasets.utils.download_manager import
    DownloadManager
    from datasets import Dataset, load_dataset,
    load_metric, ClassLabel
    from sklearn.metrics import precision_score,
    recall_score, f1_score
    from transformers import TrainingArguments,
    Trainer, TrainerCallback, Wav2Vec2ForCTC
from transformers import Wav2Vec2CTCTokenizer,
Wav2Vec2FeatureExtractor, Wav2Vec2Processor

    # Fungsi untuk memuat dataset bahasa Sunda
    def load_dataset_sunda():
        # Direktori dan nama file data
        data_dirs = [
            Path("Dataset_Update/wav")
        ]
        filenames = [
            Path("Dataset_Update/line_index.tsv")
        ]

        # Membaca data dari file tsv menggunakan
    Pandas dan menyimpannya dalam DataFrame
        dfs = []
        dfs.append(pd.read_csv(filenames[0],
    sep='\t', names=["path", "sentence"],
    encoding='latin-1'))

        # Mengubah path file dalam DataFrame menjadi
    path lengkap
        for i, dir in enumerate(data_dirs):
            dfs[i]["path"] = dfs[i].apply(lambda
    row: str(data_dirs[i]) + "/" + row['path'] +
    ".wav", axis=1)

        # Mengonversi DataFrame menjadi objek
    Dataset
        dataset = Dataset.from_pandas(dfs[0])

        return
    dataset.train_test_split(test_size=0.1, seed=1)
```

```
dataset = load_dataset_sunda()

     # Mengambil subset "train" dari dataset
     common_voice_train = dataset['train']

     # Mengambil subset "test" dari dataset
     common_voice_test = dataset['test']

     # Fungsi untuk menampilkan contoh-contoh acak
     dari dataset
     def show_random_elements(dataset,
     num_examples=10):
         # Memastikan jumlah contoh yang diminta
     tidak melebihi jumlah total contoh dalam dataset
         assert num_examples <= len(dataset), "Tidak
     dapat memilih elemen lebih banyak daripada yang
     ada di kumpulan data."
         picks = []

         # Memilih contoh acak sebanyak num_examples
         for _ in range(num_examples):
             pick = random.randint(0, len(dataset)-1)
             while pick in picks:
                 pick = random.randint(0,
     len(dataset)-1)
             picks.append(pick)

         # Membuat DataFrame dari contoh-contoh yang
     terpilih dan menampilkannya
         df = pd.DataFrame(dataset[picks])
    display(HTML(df.to_html()))

     # Menampilkan contoh-contoh acak dari subset
     "train" dataset common_voice_train setelah
     menghapus kolom "path"
show_random_elements(common_voice_train.remove_column
s(["path"]), num_examples=20)

     # Pola regex untuk karakter khusus yang akan
     dihapus
     chars_to_ignore_regex = '[\,\?\.\!\-
     \;\:\"\“\%\‘\'\”_\\x1a\½\¿\ï]'

     # Fungsi untuk menghapus karakter khusus dari
     batch
     def remove_special_characters(batch):
         # Menghapus karakter khusus dan mengubah
     teks menjadi huruf kecil
```

```
        batch["sentence"] =
    re.sub(chars_to_ignore_regex, '',
    batch["sentence"]).lower() + " "
        # Menghapus karakter khusus tambahan
        batch["sentence"] =
    batch["sentence"].replace('! ', '')
        batch["sentence"] =
    batch["sentence"].replace(', ', '')
        batch["sentence"] =
    batch["sentence"].replace('é', 'e')
        batch["sentence"] =
    batch["sentence"].replace('è', 'e')
        return batch

    # Mengaplikasikan fungsi
    remove_special_characters pada subset "train"
    dan "test" dari dataset
    common_voice_train =
    common_voice_train.map(remove_special_characters
    )
common_voice_test =
common_voice_test.map(remove_special_characters)

    # Menampilkan contoh-contoh acak dari subset
    "train" dataset common_voice_train setelah
    menghapus kolom "path"
show_random_elements(common_voice_train.remove_column
s(["path"]), num_examples=10)

    # Fungsi untuk mengekstrak semua karakter yang
    muncul dalam batch
    def extract_all_chars(batch):
      # Menggabungkan semua teks dalam batch menjadi
    satu teks panjang
      all_text = " ".join(batch["sentence"])
      # Membuat vocab dari setiap karakter unik yang
    muncul dalam teks
      vocab = list(set(all_text))
      # Mengembalikan vocab dan teks utuh
      return {"vocab": [vocab], "all_text":
    [all_text]}

    # Mengekstrak vocab dari dataset latih dengan
    menggunakan fungsi extract_all_chars
    vocab_train = common_voice_train.map(
        extract_all_chars,
        batched=True,
        batch_size=-1,
        keep_in_memory=True,
```

```
    remove_columns=common_voice_train.column_names
    )

    # Mengekstrak vocab dari dataset uji dengan
    menggunakan fungsi extract_all_chars
    vocab_test = common_voice_test.map(
        extract_all_chars,
        batched=True,
        batch_size=-1,
        keep_in_memory=True,

    remove_columns=common_voice_test.column_names
)

    # Menggabungkan vocab dari dataset latih dan
    dataset uji
    vocab_list = list(set(vocab_train["vocab"][0]) |
    set(vocab_test["vocab"][0]))

    # Membuat kamus (dictionary) di mana setiap
    karakter dari vocab_list memiliki indeksnya
    sendiri
    vocab_dict = {v: k for k, v in
    enumerate(vocab_list)}

    # Mengembalikan kamus yang berisi indeks untuk
    setiap karakter dalam vocab_list
vocab_dict

    # Menetapkan indeks untuk karakter "|" sama
    dengan indeks untuk spasi kosong (" ")
    vocab_dict["|"] = vocab_dict[" "]

    # Menghapus entri untuk spasi kosong (" ") dari
    kamus
del vocab_dict[" "]

    vocab_dict["[UNK]"] = len(vocab_dict)

    # Menambahkan entri untuk token [PAD] (token
    untuk padding) dengan indeks yang baru
    vocab_dict["[PAD]"] = len(vocab_dict)

    # Menghitung jumlah total token dalam kamus
len(vocab_dict)

    # Menyimpan kamus vocab_dict ke dalam file JSON
    with open('vocab-su.json', 'w') as vocab_file:
```

```
        json.dump(vocab_dict, vocab_file)

    # Membaca isi dari file vocab-su.json
    with open('vocab-su.json', 'r') as vocab_file:
        vocab_content = json.load(vocab_file)

    # Menampilkan isi vocab-su.json
    print("Isi vocab-su.json:")
    print(vocab_content)

    # Membuat tokenizer menggunakan kamus vocab_dict
    yang telah disimpan dalam file JSON
    tokenizer = Wav2Vec2CTCTokenizer("./vocab-
    su.json", unk_token=None, pad_token="[PAD]",
    word_delimiter_token="|")

    # Menampilkan vocab dari tokenizer
    print("\nVocab dari tokenizer:")
print(tokenizer.get_vocab())

    # Membuat objek Wav2Vec2FeatureExtractor dengan
    konfigurasi tertentu
    feature_extractor = Wav2Vec2FeatureExtractor(
        feature_size=1,  # Ukuran fitur
        sampling_rate=16000,  # Frekuensi sampel
        padding_value=0.0,  # Nilai untuk padding
        do_normalize=True,  # Normalisasi fitur
        return_attention_mask=True  # Mengembalikan
    masker perhatian
    )

    # Membuat objek Wav2Vec2Processor dengan
    menggunakan objek feature_extractor dan
    tokenizer yang sudah ada
    processor = Wav2Vec2Processor(
        feature_extractor=feature_extractor,  #
    Pengolah fitur
        tokenizer=tokenizer  # Tokenizer
    )

    # Mencetak informasi tentang objek processor
print(processor)

processor.save_pretrained("./wav2vec2-sunda-V1.1")

    # Fungsi untuk mengonversi file audio menjadi
    larik numpy
    def speech_file_to_array_fn(batch):
        # Memuat file audio menggunakan torchaudio
```

```
        speech_array, sampling_rate =
    torchaudio.load(batch["path"])
        # Mengonversi tensor audio menjadi numpy
    array
        batch["speech"] = speech_array[0].numpy()
        # Menyimpan sampling rate
        batch["sampling_rate"] = sampling_rate
        # Menyalin teks target ke kolom baru
    "target_text"
        batch["target_text"] = batch["sentence"]
        return batch

    # Mengaplikasikan fungsi speech_file_to_array_fn
    pada dataset latih dan uji
    common_voice_train =
    common_voice_train.map(speech_file_to_array_fn,
    remove_columns=common_voice_train.column_names)
common_voice_test =
common_voice_test.map(speech_file_to_array_fn,
remove_columns=common_voice_test.column_names)

    # Memilih secara acak contoh dari dataset latih
    dan mencetak informasi tentangnya
    rand_int = random.randint(0,
    len(common_voice_train))
    print("Target text:",
    common_voice_train[rand_int]["target_text"])  #
    Mencetak teks target
    print("Input array shape:",
    np.asarray(common_voice_train[rand_int]["speech"
    ]).shape)  # Mencetak bentuk larik input
print("Sampling rate:",
common_voice_train[rand_int]["sampling_rate"])  #
Mencetak sampling rate

    # Memilih secara acak audio dari dataset latih
    dan memainkannya
    rand_int = random.randint(0,
    len(common_voice_train))
ipd.Audio(data=np.asarray(common_voice_train[rand_int
]["speech"]), autoplay=False, rate=16000)

    # Memilih secara acak audio dari dataset uji dan
    memainkannya
    rand_int = random.randint(0,
    len(common_voice_test))
ipd.Audio(data=np.asarray(common_voice_test[rand_int]
["speech"]), autoplay=False, rate=16000)
```

```
# Fungsi untuk mempersiapkan dataset dengan
pengolahan dan tokenisasi data
def prepare_dataset(batch):
    # Impor yang diperlukan
    from transformers import
Wav2Vec2FeatureExtractor, Wav2Vec2Processor,
Wav2Vec2CTCTokenizer

    # Membuat tokenizer
    tokenizer = Wav2Vec2CTCTokenizer("./vocab-
su.json", pad_token="[PAD]",
word_delimiter_token="|")

    # Membuat pengolah fitur
    feature_extractor =
Wav2Vec2FeatureExtractor(feature_size=1,
sampling_rate=16000, padding_value=0.0,
do_normalize=True, return_attention_mask=True)

    # Membuat pengolah
    processor =
Wav2Vec2Processor(feature_extractor=feature_extr
actor, tokenizer=tokenizer)

    # Memastikan semua input memiliki sampling
rate yang sama
    assert (
        len(set(batch["sampling_rate"])) == 1
    ), f"Make sure all inputs have the same
sampling rate of
{processor.feature_extractor.sampling_rate}."

    # Memproses input_values
    batch["input_values"] =
processor(batch["speech"],
sampling_rate=batch["sampling_rate"][0]).input_v
alues

    # Memproses label
    with processor.as_target_processor():
        batch["labels"] =
processor(batch["target_text"]).input_ids

    return batch

# Memproses dataset latih menggunakan fungsi
prepare_dataset
common_voice_train =
common_voice_train.map(prepare_dataset,
```

```
    remove_columns=common_voice_train.column_names,
    batch_size=8, num_proc=4, batched=True)

    # Memproses dataset uji menggunakan fungsi
    prepare_dataset
common_voice_test =
common_voice_test.map(prepare_dataset,
remove_columns=common_voice_test.column_names,
batch_size=8, num_proc=4, batched=True)

    @dataclass
    class DataCollatorCTCWithPadding:
        processor: Wav2Vec2Processor
        padding: Union[bool, str] = True
        max_length: Optional[int] = None
        max_length_labels: Optional[int] = None
        pad_to_multiple_of: Optional[int] = None
        pad_to_multiple_of_labels: Optional[int] =
    None

        def __call__(self, features: List[Dict[str,
    Union[List[int], torch.Tensor]]]) -> Dict[str,
    torch.Tensor]:
            # memisahkan input dan label karena
    mereka harus memiliki panjang yang berbeda dan
    membutuhkan
            # metode padding yang berbeda
            input_features = [{"input_values":
    feature["input_values"]} for feature in
    features]
            label_features = [{"input_ids":
    feature["labels"]} for feature in features]

            # melakukan padding pada input
            batch = self.processor.pad(
                input_features,
                padding=self.padding,
                max_length=self.max_length,

    pad_to_multiple_of=self.pad_to_multiple_of,
                return_tensors="pt",
            )
            # melakukan padding pada label
            with
    self.processor.as_target_processor():
                labels_batch = self.processor.pad(
                    label_features,
                    padding=self.padding,
```

```
        max_length=self.max_length_labels,

        pad_to_multiple_of=self.pad_to_multiple_of_label
        s,
                        return_tensors="pt",
                    )

                # mengganti padding dengan -100 agar
        dapat diabaikan dengan benar saat menghitung
        loss
                labels =
        labels_batch["input_ids"].masked_fill(labels_bat
        ch.attention_mask.ne(1), -100)

                batch["labels"] = labels

                return batch

        # Membuat objek DataCollatorCTCWithPadding
data_collator =
DataCollatorCTCWithPadding(processor=processor,
padding=True)

        # Memuat metrik WER (Word Error Rate)
        wer_metric = load_metric("wer")

        # Fungsi untuk menghitung metrik
        def compute_metrics_on(pred):
            pred_logits = pred.predictions
            pred_ids = np.argmax(pred_logits, axis=-1)

            # mengabaikan token padding (-100) agar
        metrik dihitung dengan benar
            pred.label_ids[pred.label_ids == -100] =
        processor.tokenizer.pad_token_id

            # mendekode hasil prediksi dan label
            pred_str = processor.batch_decode(pred_ids)
            label_str =
        processor.batch_decode(pred.label_ids,
        group_tokens=False)

            # menghitung WER
            wer =
        wer_metric.compute(predictions=pred_str,
        references=label_str)

            # Menghitung Character Error Rate (CER)
```

```
        cer = jiwer.wer(label_str, pred_str)

        # Menghitung BLEU Score dengan unigram (N=1)
        bleu = bleu_score.corpus_bleu([[ref.split()]
    for ref in label_str], [pred.split() for pred in
    pred_str], weights=(1, 0, 0, 0))

        # Menghitung WER per Word
        wer_per_word = wer / sum(len(ref.split())
    for ref in label_str)

        # Menghitung CER per Character
        cer_per_char = cer / sum(len(ref) for ref in
    label_str)

        # Menghitung waktu sekarang
        current_time = time.time()

        # Menghitung waktu yang telah berlalu sejak
    terakhir logging
        elapsed_time = current_time -
    compute_metrics_on.last_logging_time

        # Menghitung Runtime per logging_steps
        Runtime = elapsed_time /
    trainer.args.logging_steps

        # Menghitung jumlah sampel yang telah
    diproses sejak terakhir logging
        samples_processed =
    len(trainer.train_dataset) *
    (trainer.state.global_step -
    compute_metrics_on.last_global_step)

        # Menghitung Samples Per Second
        samples_per_sec = samples_processed /
    elapsed_time

        # Memperbarui waktu terakhir logging dan
    langkah global terakhir
        compute_metrics_on.last_logging_time =
    current_time
        compute_metrics_on.last_global_step =
    trainer.state.global_step

    return {"wer": wer, "cer": cer, "bleu": bleu,
"wer_per_word": wer_per_word, "cer_per_char":
cer_per_char, "Runtime": Runtime, "samples_per_sec":
samples_per_sec}
```

```
    model = Wav2Vec2ForCTC.from_pretrained(
        "facebook/wav2vec2-large-xlsr-53",
        attention_dropout=0.2,
        hidden_dropout=0.2,
        feat_proj_dropout=0.0,
        mask_time_prob=0.3,
        layerdrop=0.1
        gradient_checkpointing=True,
        ctc_loss_reduction="mean",

    pad_token_id=processor.tokenizer.pad_token_id,
        vocab_size=len(processor.tokenizer),
        ctc_zero_infinity=True
)

    # Menginisialisasi objek TrainingArguments untuk
    mengatur proses pelatihan model
    training_args = TrainingArguments(
        output_dir="./wav2vec2-sunda-V1.1",
    # Direktori output untuk menyimpan model dan log
        group_by_length=True,
    # Mengelompokkan sampel berdasarkan panjang
    untuk efisiensi
        per_device_train_batch_size=2,
    # Ukuran batch pelatihan per perangkat
        gradient_accumulation_steps=2,
    # Jumlah langkah akumulasi gradien
        evaluation_strategy="steps",
    # Strategi evaluasi pada langkah tertentu
        num_train_epochs=50,
    # Jumlah epoch pelatihan
        fp16=True,
    # Menggunakan bfloat16 (brain floating point)
        save_steps=4000,
    # Langkah interval penyimpanan model
        eval_steps=2000,
    # Langkah interval evaluasi model
        logging_steps=2000,
    # Langkah interval logging
        learning_rate=3e-4,
    # Tingkat pembelajaran awal
        warmup_steps=100,
    # Jumlah langkah pemanasan (warmup)
        save_total_limit=2,
    # Batas total model yang disimpan
        eval_accumulation_steps=2,
    # Jumlah langkah akumulasi evaluasi
```

```
        use_cpu=False,
    # Gunakan CPU, defaultnya adalah False (gunakan
    GPU)
        load_best_model_at_end=True,
    # Muat model terbaik pada akhir pelatihan
        optim="adamw_torch"
    # Pengoptimal yang digunakan selama pelatihan
)

    # Menginisialisasi objek Trainer untuk melatih
    model
    trainer = Trainer(
        model=model,                        # Model
    yang akan dilatih
        data_collator=data_collator,            #
    Data collator untuk pemrosesan data
        args=training_args,                     #
    Argumen pelatihan
        compute_metrics=compute_metrics_on,     #
    Fungsi untuk menghitung metrik
        train_dataset=common_voice_train,       #
    Dataset pelatihan
        eval_dataset=common_voice_test,         #
    Dataset evaluasi
        tokenizer=processor.feature_extractor,  #
    Tokenizer untuk pengolahan fitur
    )

    # Inisialisasi variabel statis untuk menyimpan
    waktu terakhir logging dan langkah global
    terakhir
    compute_metrics_on.last_logging_time =
    time.time()
compute_metrics_on.last_global_step =
trainer.state.global_step
    # Melatih model menggunakan objek trainer
trainer.train()
    trainer.save_model()
trainer.save_state()
```

5. Source Code OCR

```
import pandas as pd

# Membaca dataset
df                                                    =
pd.read_csv('./Dataset_v1/output_images1/data.csv')
df.head()

# Pastikan kolom 'teks' berisi nilai string
df['teks'] = df['teks'].astype(str)

from sklearn.model_selection import train_test_split

# Membagi dataset menjadi train, validation, dan test
train_df,     temp_df     =     train_test_split(df,
test_size=0.1)
val_df,    test_df    =    train_test_split(temp_df,
test_size=0.1)
#  Reset  indeks  untuk  memastikan  tidak  ada  masalah
indexing
train_df.reset_index(drop=True, inplace=True)
val_df.reset_index(drop=True, inplace=True)
test_df.reset_index(drop=True, inplace=True)

import torch
from torch.utils.data import Dataset
from PIL import Image

# Definisi kelas IAMDataset
class IAMDataset(Dataset):
    def   __init__(self,   root_dir,   df,   processor,
max_target_length=128):
        self.root_dir = root_dir
        self.df = df
        self.processor = processor
        self.max_target_length = max_target_length

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        if idx >= len(self.df) or idx < 0:
            raise IndexError("Index out of bounds")
```

```
        file_name = self.df.loc[idx, 'nama_gambar']
        text = str(self.df.loc[idx, 'teks']) #
Pastikan teks adalah string

        # Siapkan gambar (resize + normalisasi)
        image = Image.open(self.root_dir + "/" +
file_name).convert("RGB")
        pixel_values = self.processor(image,
return_tensors="pt").pixel_values

        # Tambahkan label (input_ids) dengan meng-
encode teks
        labels = self.processor.tokenizer(text,

padding="max_length",

max_length=self.max_target_length).input_ids
        # Pastikan token PAD diabaikan oleh fungsi loss
        labels = [label if label !=
self.processor.tokenizer.pad_token_id else -100 for
label in labels]

        encoding = {"pixel_values":
pixel_values.squeeze(), "labels":
torch.tensor(labels)}
        return encoding

from transformers import TrOCRProcessor

# Inisialisasi processor
processor =
TrOCRProcessor.from_pretrained("microsoft/trocr-base-
handwritten")

# Membuat dataset untuk training dan evaluasi
train_dataset =
IAMDataset(root_dir='./Dataset_v1/output_images1',
df=train_df, processor=processor)
eval_dataset =
IAMDataset(root_dir='./Dataset_v1/output_images1',
df=val_df, processor=processor)
test_dataset =
IAMDataset(root_dir='./Dataset_v1/output_images1',
df=test_df, processor=processor)

# Debugging jumlah data
print("Number of training examples:",
len(train_dataset))
```

```
print("Number of validation examples:", len(eval_dataset))
print("Number of test examples:", len(test_dataset))

# Debugging satu sampel dari dataset
encoding = train_dataset[0]
for k, v in encoding.items():
    print(k, v.shape if isinstance(v, torch.Tensor) else v)

# Menampilkan gambar pertama dan labelnya
image = Image.open(train_dataset.root_dir + "/" + train_df.loc[0, 'nama_gambar']).convert("RGB")
image

labels = encoding['labels']
labels[labels == -100] = processor.tokenizer.pad_token_id
label_str = processor.decode(labels, skip_special_tokens=True)
print(label_str)

from tqdm import tqdm
import torch

# Fungsi untuk memeriksa tokenisasi label dengan bar progres
def check_tokenization(processor, dataset, batch_size=32):
    invalid_labels = []
    invalid_chars_set = set()
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    for start_idx in tqdm(range(0, len(dataset), batch_size), desc="Checking tokenization"):
        end_idx = min(start_idx + batch_size, len(dataset))
        batch_encodings = [dataset[idx] for idx in range(start_idx, end_idx)]

        batch_labels = torch.stack([encoding['labels'] for encoding in batch_encodings]).to(device)
        batch_labels[batch_labels == -100] = processor.tokenizer.pad_token_id

        with torch.no_grad():
```

```
            decoded_labels                                  =
processor.batch_decode(batch_labels.cpu(),
skip_special_tokens=True)

        for idx, label in enumerate(decoded_labels):
            invalid_chars = [(pos, char) for pos, char
in enumerate(label) if ord(char) > 255]
            if invalid_chars:  # Periksa karakter non-
ASCII
                invalid_labels.append((start_idx    +
idx, label, invalid_chars))
                for _, char in invalid_chars:
                    invalid_chars_set.add(char)

    return invalid_labels, invalid_chars_set

# Periksa tokenisasi label
invalid_labels,          invalid_chars_set          =
check_tokenization(processor, train_dataset)
if invalid_labels:
    print("Invalid labels found:")
    for idx, label, invalid_chars in invalid_labels:
        print(f"Index {idx}: {label}")
        print("Invalid characters:")
        for pos, char in invalid_chars:
            print(f"    Position   {pos}:   '{char}'
(Unicode: {ord(char)})")

    print("\nAll unique invalid characters:")
    for char in sorted(invalid_chars_set):
        print(f"'{char}' (Unicode: {ord(char)})")
else:
    print("All labels are correctly tokenized.")

from transformers import VisionEncoderDecoderModel

# Inisialisasi model
model                                                   =
VisionEncoderDecoderModel.from_pretrained("Model_v2")

model.config.decoder_start_token_id                     =
processor.tokenizer.cls_token_id
model.config.pad_token_id                               =
processor.tokenizer.pad_token_id
model.config.vocab_size                                 =
model.config.decoder.vocab_size

# Set parameter beam search
```

```
model.config.eos_token_id                                    =
processor.tokenizer.sep_token_id
model.config.max_length = 64
model.config.early_stopping = True
model.config.no_repeat_ngram_size = 3
model.config.length_penalty = 2.0
model.config.num_beams = 4

from     transformers     import     Seq2SeqTrainer,
Seq2SeqTrainingArguments

# Argument training
training_args = Seq2SeqTrainingArguments(
    predict_with_generate=True,
    group_by_length=True,
    evaluation_strategy="steps",
    per_device_train_batch_size=4,    #  Meningkatkan
batch size jika memori GPU memungkinkan
    per_device_eval_batch_size=4,
    fp16=True,
    output_dir="./Model_v2",
    save_steps=5411,
    eval_steps=5411,
    logging_steps=5411,
    warmup_steps=500,  # Menambah warmup steps
    save_total_limit=1,
    optim="adamw_torch",
    learning_rate=3e-5,  # Menurunkan learning rate
    weight_decay=0.01,
    lr_scheduler_type="cosine_with_restarts",
    load_best_model_at_end=True,
    metric_for_best_model="cer",
    greater_is_better=False
)

from datasets import load_metric

# Menghitung metrics
cer_metric = load_metric("cer")

def compute_metrics(pred):
    labels_ids = pred.label_ids
    pred_ids = pred.predictions

    pred_str    =    processor.batch_decode(pred_ids,
skip_special_tokens=True)
    labels_ids[labels_ids       ==       -100]       =
processor.tokenizer.pad_token_id
```

```
    label_str = processor.batch_decode(labels_ids,
skip_special_tokens=True)

    cer = cer_metric.compute(predictions=pred_str,
references=label_str)

    return {"cer": cer}

from transformers import default_data_collator,
EarlyStoppingCallback

# Inisialisasi trainer dengan EarlyStoppingCallback
trainer = Seq2SeqTrainer(
    model=model,
    tokenizer=processor.feature_extractor,
    args=training_args,
    compute_metrics=compute_metrics,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    data_collator=default_data_collator,

callbacks=[EarlyStoppingCallback(early_stopping_patie
nce=3)]  # Menambahkan early stopping dengan patience
3
)

# Melatih model
trainer.train()

import shutil

# Menyimpan model dan processor ke direktori
model.save_pretrained("Model_v3")
processor.save_pretrained("Model_v3")

# Menyimpan state trainer ke direktori output yang
ditentukan
trainer.save_state()

# Pindahkan state trainer ke direktori yang berbeda
shutil.move(trainer.args.output_dir,
"Model_v3/state")
```