4. IMPLEMENTASI SISTEM

Pada bab ini akan membahas mengenai implementasi dari desain sistem yang telah dibuat pada bab sebelumnya. Implementasi sistem berupa penggunaan perangkat lunak yang digunakan, library, syntax, cleaning & preprocessing data, training dan testing model algoritma yang digunakan. Serta penerapan pembuatan sistem berupa website yang menerima input sinopsis berupa teks dan dapat memprediksi *vote average* dan genre dari film/series tersebut.

4.1 Implementasi Perangkat Lunak yang Digunakan

Penerapan sistem AI yang digunakan dalam aplikasi ini adalah berupa website. Website dikembangkan dengan HTML, Bootstrap 5, Javascript, dan Python 3.11.4 menggunakan library *Flask*. Python juga digunakan dalam proses scraping data, cleaning, preprocessing, dan melakukan train model.

4.2 Model Vote Average

PyCharm berperan sebagai Integrated Development Environment (IDE) untuk Python yang digunakan untuk mengedit teks pemrograman Python, melakukan training dan testing algoritma, serta instalasi library. Segmen program ini bertujuan untuk mengimpor library dan persiapan awal sebelum melakukan pemrosesan dan pelatihan model dengan menggunakan IndoBERT dan Indonesian RoBERTa.

4.2.1 Library yang Digunakan

```
import torch
import pandas as pd
import numpy as np
from transformers import set_seed, AutoTokenizer, AutoModelForSequenceClassification,
Trainer, TrainingArguments
from sklearn.model_selection import KFold
from torch.utils.data import Dataset
from datasets import Dataset as HFDataset
from sklearn.metrics import mean_absolute_error
```

Segmen Program 4.2.1 Import Library

- **torch**: Digunakan untuk operasi tensor dan manipulasi GPU.
- **pandas**: Digunakan untuk manipulasi data dan analisis dalam format DataFrame.
- **numpy**: Digunakan untuk operasi numerik.
- **transformers**: Digunakan untuk bekerja dengan model transformator, tokenizer, dan pelatihan model.
- **sklearn.model_selection**: Digunakan untuk melakukan cross-validation (KFold).
- **torch.utils.data**: Digunakan untuk bekerja dengan dataset PyTorch.
- **datasets**: Digunakan untuk bekerja dengan dataset dalam format Hugging Face.
- **sklearn.metrics**: Digunakan untuk menghitung metrik evaluasi (Mean Absolute Error).

4.2.2 Custom Trainer Class dan Cek GPU

```
class CustomTrainer(Trainer):
   def compute_loss(self, model, inputs, return_outputs=False):
      labels = inputs.pop("labels")  # Remove labels from inputs
      outputs = model(**inputs)
      logits = outputs.logits
      loss = torch.nn.functional.mse_loss(logits.view(-1), labels.float().view(-1))
      return (loss, outputs) if return_outputs else loss

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

Segmen Program 4.2.2 Class Custom Trainer dan pengecekan GPU

- Membuat kelas CustomTrainer yang mewarisi dari kelas Trainer dari library transformers.
- compute_loss: Menghitung loss dengan menggunakan Mean Squared Error (MSE) loss yang sesuai untuk tugas regresi.
- Mengecek ketersediaan GPU dan menetapkan perangkat ke cuda jika GPU tersedia, atau cpu jika tidak tersedia.

4.2.3 Load Data dan Preprocessing

```
data = pd.read_csv("../dataset/dataset2.csv", sep=';')
data = data[['overview', 'vote_average']].dropna()
data = data[data['vote_average'] != 0.0]
data = data.drop_duplicates()
```

Segmen Program 4.2.3 Load Data dan Preprocessing

- Membaca dataset dari file CSV.
- Memilih kolom 'overview' dan 'vote_average', kemudian menghapus data yang null.
- Menghapus baris dengan 'vote_average' yang sama dengan 0.0.

4.2.4 Load Tokenizer dan Model

```
tokenizer = AutoTokenizer.from_pretrained("indolem/indobert-base-uncased")
model = AutoModelForSequenceClassification.from_pretrained("indolem/indobert-base-
uncased", num_labels=1)
model.to(device)
```

Segmen Program 4.2.4 Load Tokenizer dan Model untuk IndoBERT

- Memuat tokenizer dan model IndoBERT dari model pramuat (pre-trained).
- Model dikonfigurasi untuk tugas regresi dengan satu label.
- Memindahkan model ke perangkat yang telah ditentukan sebelumnya (GPU atau CPU).

```
tokenizer = AutoTokenizer.from_pretrained("cahya/roberta-base-indonesian-522M")
model = AutoModelForSequenceClassification.from_pretrained("cahya/roberta-base-indonesian-522M", num_labels=1)  # Regression with 1 label
```

Segmen Program 4.2.5 Load Tokenizer dan Model untuk Indonesian RoBERTa

- Memuat tokenizer dan model Indonesian RoBERTa dari model pramuat (pre-trained).
- Model dikonfigurasi untuk tugas regresi dengan satu label.
- Memindahkan model ke perangkat yang telah ditentukan sebelumnya (GPU atau CPU).

4.2.5 Konversi dan Tokenisasi Dataset

```
dataset = HFDataset.from_pandas(data.rename(columns={'vote_average': 'labels'}))
max_seq_length = 250
dataset = dataset.map(lambda x: tokenizer(x['overview'], padding='max_length', truncation=True, max_length=max_seq_length), batched=True)
```

Segmen Program 4.2.6 Konversi dan Tokenisasi

- Mengonversi data Pandas DataFrame menjadi dataset Hugging Face dengan mengganti nama kolom 'vote_average' menjadi 'labels'.
- Men-tokenisasi dataset dengan padding dan truncation untuk memastikan panjang urutan teks seragam (250 token).

4.2.6 Definisi Argumen Pelatihan & Inisialisasi Variabel Evaluasi

```
training_args = TrainingArguments(
   output_dir="./kfold_bert",
   num_train_epochs=20,
   per_device_train_batch_size=16,
   per_device_eval_batch_size=16,
   warmup_steps=500,
   evaluation_strategy="epoch",
   logging_dir="./logs",
   save_strategy="no",
)

eval_results = []
best_mae = float('inf')
best_model = None
```

Segmen Program 4.2.7 Inisialisasi variabel pelatihan

- Menetapkan argumen untuk pelatihan seperti direktori keluaran, jumlah epoch, ukuran batch untuk pelatihan dan evaluasi, langkah pemanasan, strategi evaluasi, dan direktori log.
- Menyiapkan variabel untuk melacak hasil evaluasi setiap fold, nilai Mean Absolute Error (MAE) terbaik, dan model terbaik.

4.2.7 Pelaksanaan K-fold Cross-Validation

```
kf = KFold(n_splits=5, shuffle=True)

for fold, (train_idx, test_idx) in enumerate(kf.split(dataset)):
   print(f"Fold {fold + 1}/5")
   train_data = dataset.select(train_idx)
   test_data = dataset.select(test_idx)

   trainer = CustomTrainer(
      model=model,
      args=training_args,
      train_dataset=train_data,
      eval_dataset=test_data,
      compute_metrics=lambda x: {"mae": mean_absolute_error(x.label_ids, x.predictions)},
   )

   trainer.train()
   evaluation_result = trainer.evaluate(test_data)
   print("Evaluation result:", evaluation_result)
   print("MAE:", evaluation_result['eval_mae'])

   if evaluation_result['eval_mae'] < best_mae:
      best_mae = evaluation_result['eval_mae']
      best_model = trainer.model

   eval_results.append(evaluation_result)
```

Segmen Program 4.2.8 Training Model

- Menggunakan K-fold cross-validation dengan 5 fold dan pengacakan data.
- Melakukan loop untuk setiap fold dalam K-fold cross-validation.
- Membagi data menjadi train dan test set untuk setiap fold.
- Membuat instance dari CustomTrainer dengan data train dan test set.
- Melatih model pada train set.
- Mengevaluasi model pada test set.
- Memeriksa apakah MAE dari fold ini adalah yang terbaik, jika ya, simpan model sebagai model terbaik.
- Menyimpan hasil evaluasi untuk setiap fold.

4.2.8 Hitung Rata-rata Hasil Evaluasi

```
avg_mae = np.mean([result['eval_mae'] for result in eval_results])
print("Average MAE:", avg_mae)
print("Model with the lowest MAE:", best_model)
best_model.save_pretrained("./kfold_bert")
```

Segmen Program 4.2.9 Hitung MAE

- Menghitung rata-rata MAE dari semua fold.
- Mencetak rata-rata MAE.
- Mencetak model dengan MAE terendah.
- Menyimpan model terbaik ke direktori yang ditentukan.

4.3 Model Genre

Berikut ini adalah bagian program untuk training genre film berdasarkan sinopsis.

4.3.1 Library yang Digunakan

```
import pandas as pd
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification, AdamW
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import KFold
import numpy as np
```

Segmen Program 4.3.1 Import Library

- **torch**: Digunakan untuk operasi tensor dan manipulasi GPU.
- **pandas**: Digunakan untuk manipulasi data dan analisis dalam format DataFrame.
- **transformers**: Digunakan untuk bekerja dengan model transformator, tokenizer, dan pelatihan model.
- **sklearn.model_selection**: Digunakan untuk melakukan cross-validation (KFold).
- **torch.utils.data**: Digunakan untuk bekerja dengan dataset PyTorch.

4.3.2 Definisi Fungsi

```
# Function to calculate Jaccard similarity
def jaccard_similarity(pred_labels, true_labels):
    intersection = torch.logical_and(pred_labels, true_labels).sum(dim=1)
    union = torch.logical_or(pred_labels, true_labels).sum(dim=1)
    jaccard = intersection / union
    return jaccard.mean()
```

Segmen Program 4.3.2 Class Custom Trainer dan pengecekan GPU

- Mendefinisikan fungsi “jaccard_similarity” untuk menghitung kesamaan Jaccard antara label prediksi dan label sebenarnya.

4.3.3 Load Data dan Preprocessing

```
# Load dataset
df = pd.read_csv("../dataset/dataset2.csv", sep=";")
```

```
# Cleaning dataset: drop null values and check for duplicates
df.dropna(inplace=True)
df.drop_duplicates(inplace=True)

# Drop data with vote_average equal to 0.0
df = df[df['vote_average'] != 0.0]

# Select 'overview' and 'genre_ids' columns
df = df[['overview', 'genre_ids']]

# Clean and check the 'genre_ids' column
df['genre_ids'] = df['genre_ids'].str.strip('[]').replace('', None)
df.dropna(subset=['genre_ids'], inplace=True)

# Convert genre_ids to list of integers
df['genre_ids'] = df['genre_ids'].apply(
   lambda x: [int(genre_id) for genre_id in x.split(',')] if ',' in x else [int(x)])

# Prepare genre labels in the form of one-hot encoding
unique_genres = df['genre_ids'].explode().unique()
num_labels = len(unique_genres)
```

Segmen Program 4.3.3 Load Data dan Preprocessing

- Memuat dataset dari file CSV.
- Membersihkan dataset dengan menghapus nilai kosong dan duplikat.
- Menghapus data dengan vote_average sama dengan 0.0.
- Memilih kolom overview dan genre_ids.
- Membersihkan dan memeriksa kolom genre_ids.
- Mengonversi genre_ids ke daftar integer.
- Menyiapkan label genre dalam bentuk one-hot encoding.

4.3.4 Definisi KFold Cross-Validator

```
# Define KFold cross-validator
kf = KFold(n_splits=5, shuffle=True, random_state=42)
```

Segmen Program 4.3.4 Load Data dan Preprocessing

- Mendefinisikan validasi silang KFold dengan 5 lipatan.

4.3.5 Load Tokenizer dan Model

```
# Initialize model and tokenizer outside the loop
model = AutoModelForSequenceClassification.from_pretrained("indolem/indobert-base-
uncased", num_labels=num_labels)
tokenizer = AutoTokenizer.from_pretrained("indolem/indobert-base-uncased")
```

```
# Define optimizer
optimizer = AdamW(model.parameters(), lr=5e-5)

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

Segmen Program 4.3.5 Load Tokenizer dan Model untuk IndoBERT

- Memuat tokenizer dan model IndoBERT dari model pramuat (pre-trained).
- num_labels disesuaikan dengan jumlah genre yang ada
- Mendefinisikan optimizer AdamW dengan learning rate 5e-5.
- Memindahkan model ke perangkat yang telah ditentukan sebelumnya (GPU atau CPU).

```
# Initialize model and tokenizer outside the loop
model = AutoModelForSequenceClassification.from_pretrained("cahya/roberta-base-
indonesian-522M", num_labels=num_labels)
tokenizer = AutoTokenizer.from_pretrained("cahya/roberta-base-indonesian-522M")

# Define optimizer
optimizer = AdamW(model.parameters(), lr=5e-5)

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

Segmen Program 4.3.6 Load Tokenizer dan Model untuk Indonesian RoBERTa

- Memuat tokenizer dan model Indonesian RoBERTa dari model pramuat (pre-trained).
- num_labels disesuaikan dengan jumlah genre yang ada
- Mendefinisikan optimizer AdamW dengan learning rate 5e-5.
- Memindahkan model ke perangkat yang telah ditentukan sebelumnya (GPU atau CPU).

4.3.6 Training Loop

```
# Training loop
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

num_epochs = 20

best_overall_val_jaccard = 0.0
best_overall_model_state = None
best_fold = None

for fold, (train_index, val_index) in enumerate(kf.split(df)):
    print(f"Fold {fold + 1}")
    train_df, val_df = df.iloc[train_index], df.iloc[val_index]

    # Tokenize training and validation data
```

```
  train_encodings = tokenizer(train_df['overview'].tolist(), truncation=True, padding=True)
  val_encodings = tokenizer(val_df['overview'].tolist(), truncation=True, padding=True)

  # Prepare genre labels in the form of one-hot encoding
  train_labels = np.zeros((len(train_df), num_labels))
  val_labels = np.zeros((len(val_df), num_labels))

  for i, genres in enumerate(train_df['genre_ids']):
    for genre in genres:
      if genre in unique_genres:
        genre_index = np.where(unique_genres == genre)[0][0]
        train_labels[i, genre_index] = 1

  for i, genres in enumerate(val_df['genre_ids']):
    for genre in genres:
      if genre in unique_genres:
        genre_index = np.where(unique_genres == genre)[0][0]
        val_labels[i, genre_index] = 1

  # Create DataLoader for training and validation data
  train_dataset = TensorDataset(torch.tensor(train_encodings['input_ids']),
                    torch.tensor(train_encodings['attention_mask']),
                    torch.tensor(train_labels, dtype=torch.float))

  val_dataset = TensorDataset(torch.tensor(val_encodings['input_ids']),
                  torch.tensor(val_encodings['attention_mask']),
                  torch.tensor(val_labels, dtype=torch.float))

  train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
  val_loader = DataLoader(val_dataset, batch_size=16, shuffle=False)

  best_fold_val_jaccard = 0.0
  best_fold_model_state = None

  for epoch in range(num_epochs):
    print(f"Epoch {epoch + 1}/{num_epochs}")

    # Training loop
    model.train()
    total_train_loss = 0
    total_train_jaccard = 0
    for batch in train_loader:
      input_ids, attention_mask, labels = batch
      input_ids, attention_mask, labels = input_ids.to(device), attention_mask.to(device),
labels.to(device)

      optimizer.zero_grad()

      outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
```

```
            logits = outputs.logits
            loss = outputs.loss

            # Convert logits to binary predictions using threshold 0.5
            pred_labels = torch.sigmoid(logits) > 0.5

            # Calculate Jaccard similarity
            jaccard = jaccard_similarity(pred_labels, labels)

            # Backpropagation and optimization
            loss.backward()
            optimizer.step()

            total_train_loss += loss.item()
            total_train_jaccard += jaccard.item()

        avg_train_loss = total_train_loss / len(train_loader)
        avg_train_jaccard = total_train_jaccard / len(train_loader)

        # Validation loop
        model.eval()
        total_val_loss = 0
        total_val_jaccard = 0
        with torch.no_grad():
            for batch in val_loader:
                input_ids, attention_mask, labels = batch
                input_ids, attention_mask, labels = input_ids.to(device), attention_mask.to(device),
labels.to(device)

                outputs = model(input_ids=input_ids, attention_mask=attention_mask,
labels=labels)
                logits = outputs.logits
                loss = outputs.loss

                # Convert logits to binary predictions using threshold 0.5
                pred_labels = torch.sigmoid(logits) > 0.5

                # Calculate Jaccard similarity
                jaccard = jaccard_similarity(pred_labels, labels)

                total_val_loss += loss.item()
                total_val_jaccard += jaccard.item()

        avg_val_loss = total_val_loss / len(val_loader)
        avg_val_jaccard = total_val_jaccard / len(val_loader)

        print(f"Epoch {epoch + 1} - Average Training Loss: {avg_train_loss:.4f}, Average Jaccard
Similarity: {avg_train_jaccard:.4f}")
        print(f"Epoch {epoch + 1} - Average Validation Loss: {avg_val_loss:.4f}, Average Jaccard
```

```
Similarity: {avg_val_jaccard:.4f}")

      # Save the best model based on validation Jaccard similarity
      if avg_val_jaccard > best_fold_val_jaccard:
         best_fold_val_jaccard = avg_val_jaccard
         best_fold_model_state = model.state_dict()
         print(f"New best model found for fold {fold + 1} and epoch {epoch + 1} with Jaccard
similarity: {best_fold_val_jaccard:.4f}")

   # Compare the best model of this fold with the overall best model
   if best_fold_val_jaccard > best_overall_val_jaccard:
      best_overall_val_jaccard = best_fold_val_jaccard
      best_overall_model_state = best_fold_model_state
      best_fold = fold + 1
      print(f"New overall best model found from fold {fold + 1} with Jaccard similarity:
{best_overall_val_jaccard:.4f}")
```

Segmen Program 4.3.7 Konversi dan Tokenisasi

- Menyiapkan perangkat (device) untuk menjalankan model.
- Menentukan jumlah epoch.
- Memulai loop untuk setiap lipatan KFold.
- Tokenisasi data pelatihan dan validasi.
- Menyiapkan label genre dalam bentuk one-hot encoding untuk data pelatihan dan validasi.
- Membuat DataLoader untuk data pelatihan dan validasi.
- Melakukan loop untuk setiap epoch:
    - Pelatihan: Melakukan forward pass, menghitung loss, melakukan backpropagation, dan mengupdate model.
    - Validasi: Mengevaluasi model pada data validasi tanpa melakukan update.
    - Menyimpan model terbaik berdasarkan kesamaan Jaccard pada data validasi.

4.3.7 Menyimpan Model Terbaik

```
# Save the overall best model
if best_overall_model_state:
   model.load_state_dict(best_overall_model_state)
   model_path = f"./kfold_berta_genre_best_fold_{best_fold}"
   model.save_pretrained(model_path)
   tokenizer.save_pretrained(model_path)
   print(f"Overall best model saved from fold {best_fold} in folder: {model_path}")
```

Segmen Program 4.3.8 Inisialisasi variabel pelatihan

- Memuat state model terbaik dan menyimpannya bersama tokenizer.

4.4 Website

4.4.1 Library yang Digunakan

```
from flask import Flask, request, jsonify, render_template
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from flask_cors import CORS
```

Segmen Program 4.4.1 Import Library

- **flask**: Digunakan untuk membuat aplikasi web.
- **torch**: Digunakan untuk operasi tensor dengan PyTorch.
- **transformers**: Digunakan untuk bekerja dengan model transformator dan tokenizer.
- **flask_cors**: Digunakan untuk menambahkan dukungan Cross-Origin Resource Sharing (CORS) pada aplikasi Flask.

4.4.2 Inisialisasi Flask dan CORS

```
app = Flask(__name__)
CORS(app, resources={r"/*": {"origins": "*"}})
```

Segmen Program 4.4.2 Inisialisasi Flask dan set CORS

- Inisialisasi aplikasi Flask.
- Menambahkan dukungan CORS agar aplikasi dapat diakses dari berbagai domain.

4.4.3 Menambahkan header CORS

```
@app.after_request
def add_cors_headers(response):
    response.headers['Access-Control-Allow-Origin'] = '*'
    response.headers['Access-Control-Allow-Headers'] = 'Content-Type'
    response.headers['Access-Control-Allow-Methods'] = 'POST'
    return response
```

Segmen Program 4.4.3 Header CORS

- Fungsi ini menambahkan header CORS pada setiap response dari server.

4.4.4 Load Tokenizer dan Model

```
indobert_tokenizer = AutoTokenizer.from_pretrained("indolem/indobert-base-uncased")
roberta_tokenizer = AutoTokenizer.from_pretrained("cahya/roberta-base-indonesian-
522M")

indobert_model =
AutoModelForSequenceClassification.from_pretrained("./model/kfold_bert")
roberta_model =
```

```
AutoModelForSequenceClassification.from_pretrained("./model/kfold_roberta")

indobert_genre_model =
AutoModelForSequenceClassification.from_pretrained("./model/kfold_bert_genre")
roberta_genre_model =
AutoModelForSequenceClassification.from_pretrained("./model/kfold_roberta_genre")

indobert_model.eval()
roberta_model.eval()
indobert_genre_model.eval()
roberta_genre_model.eval()
```

Segmen Program 4.4.4 Memuat Model

- Memuat tokenizer dan model dari IndoBERT dan RoBERTa.
- Model untuk prediksi rating dan genre dimuat dan disetel ke mode evaluasi.

4.4.5 Definisi *Threshold* untuk Prediksi

```
threshold = 0.5
```

Segmen Program 4.4.5 Set ambang batas

- Ambang batas untuk prediksi genre (probabilitas di atas 0.5 akan diklasifikasikan sebagai genre yang diprediksi).

4.4.6 Preprocessing Input

```
def preprocess_text(text):
  return text.strip()
```

Segmen Program 4.4.6 Trim Whitespace

- Fungsi sederhana untuk menghapus spasi di awal dan akhir teks.

4.4.7 Route utama aplikasi Flask

```
@app.route('/')
def index():
  return render_template('index.html')
```

Segmen Program 4.4.7 Route untuk halaman utama

- Menyajikan halaman utama index.html.

4.4.8 Route untuk prediksi rating

```
@app.route('/predict_rating_indobert', methods=['POST'])
def predict_rating_indobert():
  input_text = request.form.get('synopsis')
```

```
    if not input_text:
        return jsonify({'error': 'No synopsis provided'}), 400

    input_text = preprocess_text(input_text)
    inputs = indobert_tokenizer(input_text, padding='max_length', truncation=True,
max_length=250, return_tensors='pt')
    with torch.no_grad():
        outputs = indobert_model(**inputs)
    predicted_rating = outputs.logits.item()

    return jsonify({'predicted_rating': predicted_rating})


@app.route('/predict_rating_roberta', methods=['POST'])
def predict_rating_roberta():
    input_text = request.form.get('synopsis')
    if not input_text:
        return jsonify({'error': 'No synopsis provided'}), 400

    input_text = preprocess_text(input_text)
    inputs = roberta_tokenizer(input_text, padding='max_length', truncation=True,
max_length=250, return_tensors='pt')
    with torch.no_grad():
        outputs = roberta_model(**inputs)
    predicted_rating = outputs.logits.item()

    return jsonify({'predicted_rating': predicted_rating})
```

Segmen Program 4.4.8 Route untuk prediksi vote_average

- Menerima sinopsis film melalui form POST request.
- Mengembalikan prediksi rating dari model IndoBERT dalam bentuk JSON.
- Mengembalikan prediksi rating dari model Indonesian Roberta dalam bentuk JSON.

4.4.9 Route untuk prediksi genre

```
@app.route('/predict_genre_indobert', methods=['POST'])
def predict_genre_indobert():
    input_text = request.form.get('synopsis')
    if not input_text:
        return jsonify({'error': 'No synopsis provided'}), 400

    input_text = preprocess_text(input_text)
    inputs = indobert_tokenizer(input_text, return_tensors="pt", truncation=True,
padding='max_length', max_length=250)
    with torch.no_grad():
        outputs = indobert_genre_model(**inputs)

    logits = outputs.logits
```

```
    probabilities = torch.sigmoid(logits)

    predicted_genres = []
    for i, prob in enumerate(probabilities[0]):
        if prob > threshold:
            predicted_genres.append(i)

    genre_labels = [
        "Drama", "Family", "Adventure", "Animation", "Comedy", "Fantasy", "Action", "Science
Fiction",
        "Crime", "Thriller", "War", "History", "Romance", "Mystery", "Horror", "TV Movie",
"Music",
        "Documentary", "Western"
    ]

    predicted_genre_labels = [genre_labels[idx] for idx in predicted_genres]

    return jsonify({'predicted_genres': predicted_genre_labels})



@app.route('/predict_genre_roberta', methods=['POST'])
def predict_genre_roberta():
    input_text = request.form.get('synopsis')
    if not input_text:
        return jsonify({'error': 'No synopsis provided'}), 400

    input_text = preprocess_text(input_text)
    inputs = roberta_tokenizer(input_text, return_tensors="pt", truncation=True,
padding='max_length', max_length=250)
    with torch.no_grad():
        outputs = roberta_genre_model(**inputs)

    logits = outputs.logits
    probabilities = torch.sigmoid(logits)

    predicted_genres = []
    for i, prob in enumerate(probabilities[0]):
        if prob > threshold:
            predicted_genres.append(i)

    genre_labels = [
        "Drama", "Family", "Adventure", "Animation", "Comedy", "Fantasy", "Action", "Science
Fiction",
        "Crime", "Thriller", "War", "History", "Romance", "Mystery", "Horror", "TV Movie",
"Music",
        "Documentary", "Western"
    ]
```

```
    predicted_genre_labels = [genre_labels[idx] for idx in predicted_genres]

    return jsonify({'predicted_genres': predicted_genre_labels})
```

Segmen Program 4.4.9 Route untuk prediksi genre

- Menerima sinopsis film melalui form POST request.
- Mengembalikan prediksi genre dari model IndoBERT dalam bentuk JSON.
- Mengembalikan prediksi genre dari model Indonesian Roberta dalam bentuk JSON.

4.4.10 Menjalankan Aplikasi

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=443, ssl_context=('cert.pem', 'key.pem'))
```

Segmen Program 4.4.10 Run aplikasi dengan SSL

- Menjalankan aplikasi Flask pada port 443 dengan SSL menggunakan sertifikat yang ditentukan (cert.pem dan key.pem).

4.4.11 HTML Header dan Metadata

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Movie Synopsis Analyzer</title>
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
rel="stylesheet">
    <style>
        body {
            background-color: #333;
            color: white;
        }
        .container {
            max-width: 800px;
            margin-top: 50px;
        }
        .word-count {
            text-align: left;
            font-size: 0.9em;
            color: #aaa;
        }
        .loading-overlay {
            display: none;
            position: fixed;
            top: 0;
            left: 0;
```

```
        width: 100%;
        height: 100%;
        background: rgba(0, 0, 0, 0.5);
        z-index: 9999;
      }
      .loading-spinner {
        position: absolute;
        top: 50%;
        left: 50%;
        transform: translate(-50%, -50%);
        border: 5px solid #f3f3f3;
        border-top: 5px solid #3498db;
        border-radius: 50%;
        width: 50px;
        height: 50px;
        animation: spin 1s linear infinite;
      }
      @keyframes spin {
        0% { transform: rotate(0deg); }
        100% { transform: rotate(360deg); }
      }
    </style>
</head>
```

Segmen Program 4.4.11 Header HTML

- Bagian header dari halaman HTML berisi informasi metadata seperti pengaturan karakter dan skala tampilan, serta referensi ke stylesheet eksternal untuk menggunakan Bootstrap.

4.4.12 Form Input Sinopsis

```
<body>
  <div class="container bg-dark p-4 rounded">
    <h1 class="text-center">Movie Synopsis Analyzer</h1>
    <div class="form-group">
      <label for="synopsis">Your Synopsis</label>
      <textarea class="form-control bg-dark text-white" id="synopsis"
rows="8"></textarea>
      <div class="word-count">Word count: <span id="wordCount">0/250</span></div>
    </div>
    <div class="text-right mb-3">
      <button id="sampleTextBtn" class="btn btn-primary">Sample Text</button>
      <button id="clearTextBtn" class="btn btn-danger ml-2">Clear Text</button>
      <button id="predictBtn" class="btn btn-success ml-2">Predict</button>
    </div>
```

Segmen Program 4.4.12 Form HTML

- Elemen div dengan kelas container digunakan sebagai kontainer utama yang membungkus seluruh isi halaman.
- textarea digunakan untuk input sinopsis film. Pengguna dapat mengetik sinopsis film di sini.
- div dengan kelas word-count menampilkan jumlah kata yang telah diinput oleh pengguna.
- Tiga tombol disediakan untuk memanipulasi teks sinopsis:
    - Sample Text untuk mengisi textarea dengan contoh teks.
    - Clear Text untuk menghapus teks yang ada di textarea.
    - Predict untuk memulai proses prediksi rating dan genre.

4.4.13 Tabel Hasil Prediksi

```
    <div id="results">
      <table class="table table-dark table-striped">
        <thead>
          <tr>
            <th>Model</th>
            <th>Predicted Vote Rating</th>
            <th>Predicted Genres</th>
          </tr>
        </thead>
        <tbody>
          <tr>
            <td>IndoBERT</td>
            <td id="rating_indobert">-</td>
            <td id="genres_indobert">-</td>
          </tr>
          <tr>
            <td>RoBERTa</td>
            <td id="rating_roberta">-</td>
            <td id="genres_roberta">-</td>
          </tr>
        </tbody>
      </table>
    </div>
  </div>

  <div class="loading-overlay" id="loading">
    <div class="loading-spinner"></div>
  </div>
```

Segmen Program 4.4.13 Tabel hasil prediksi

- table digunakan untuk menampilkan hasil prediksi. Tabel ini memiliki dua baris, satu untuk masing-masing model (IndoBERT dan RoBERTa).
- td dengan id rating_indobert dan genres_indobert digunakan untuk menampilkan hasil prediksi dari model IndoBERT.

- td dengan id rating_roberta dan genres_roberta digunakan untuk menampilkan hasil prediksi dari model RoBERTa.
- div dengan kelas loading-overlay menampilkan layar gelap transparan untuk menutupi halaman saat proses prediksi berlangsung.
- div dengan kelas loading-spinner menampilkan animasi spinner sebagai indikator loading.

4.4.14 Program Javascript

```
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
  const MAX_WORD_COUNT = 250;

  function updateWordCount() {
    var text = $('#synopsis').val();
    var wordCount = text.trim().split(/\s+/).length;
    if (text.trim().length === 0) {
      wordCount = 0;
    }

    if (wordCount > MAX_WORD_COUNT) {
      var words = text.trim().split(/\s+/).slice(0, MAX_WORD_COUNT);
      text = words.join(' ');
      $('#synopsis').val(text);
      wordCount = MAX_WORD_COUNT;
    }

    $('#wordCount').text(wordCount + '/' + MAX_WORD_COUNT);

    $('#wordCount').css('color', wordCount >= MAX_WORD_COUNT ? 'red' :
      wordCount >= 200 ? 'yellow' : 'white');
  }

  $(document).ready(function () {
    $('#synopsis').val('');
    updateWordCount();

    $('#synopsis').on('input', updateWordCount);

    $('#sampleTextBtn').click(function () {
      var defaultText = "Film ini berpusat pada cerita Caesar (Andy Serkis), seekor
simpanse yang mendapatkan kecerdasan dan emosi seperti manusia dari obat
eksperimental. Caesar dibesarkan seperti anak manusia oleh pencipta obat tersebut, Will
Rodman (James Franco), dan seorang primatolog Caroline Aranha (Freida Pinto). Namun,
Caesar akhirnya terpisah dari manusia yang dicintainya dan dipenjara di sebuah tempat
perlindungan kera di San Bruno. Mencari keadilan untuk teman-teman satu penjaranya,
Caesar memberikan obat yang sama kepada kera-kera lain yang dia warisi. Dia kemudian
mengumpulkan sebuah pasukan simpanse dan melarikan diri dari tempat perlindungan.
```

```
Menempatkan manusia dan kera dalam jalur tabrakan yang bisa mengubah planet ini
selamanya.";
            $('#synopsis').val(defaultText);
            updateWordCount();
        });

        $('#clearTextBtn').click(function () {
            $('#synopsis').val('');
            updateWordCount();
        });

        $('#predictBtn').click(async function () {
            var synopsis = $('#synopsis').val();
            $('#loading').show();

            try {
                const results = await Promise.all([
                    fetch('/predict_rating_indobert', {
                        method: 'POST',
                        headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
                        body: new URLSearchParams({ synopsis: synopsis })
                    }).then(response => response.json()),
                    fetch('/predict_rating_roberta', {
                        method: 'POST',
                        headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
                        body: new URLSearchParams({ synopsis: synopsis })
                    }).then(response => response.json()),
                    fetch('/predict_genre_indobert', {
                        method: 'POST',
                        headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
                        body: new URLSearchParams({ synopsis: synopsis })
                    }).then(response => response.json()),
                    fetch('/predict_genre_roberta', {
                        method: 'POST',
                        headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
                        body: new URLSearchParams({ synopsis: synopsis })
                    }).then(response => response.json())
                ]);

                const [ratingIndoBertData, ratingRoBertaData, genresIndoBertData,
genresRoBertaData] = results;

                $('#rating_indobert').text(ratingIndoBertData.error ? `Error:
${ratingIndoBertData.error}` :
                    `${ratingIndoBertData.predicted_rating.toFixed(5)} / 10`);
                $('#rating_roberta').text(ratingRoBertaData.error ? `Error:
${ratingRoBertaData.error}` :
                    `${ratingRoBertaData.predicted_rating.toFixed(5)} / 10`);
```

```
            $('#genres_indobert').text(genresIndoBertData.error ? `Error:
${genresIndoBertData.error}` :
                genresIndoBertData.predicted_genres.join(', '));
            $('#genres_roberta').text(genresRoBertaData.error ? `Error:
${genresRoBertaData.error}` :
                genresRoBertaData.predicted_genres.join(', '));
          } catch (error) {
            console.error(`API error: ${error.message}`);
          } finally {
            $('#loading').hide();
          }
        });
    });
  </script>
</body>
</html>
```

Segmen Program 4.4.14 Code Javascript

- updateWordCount() adalah fungsi untuk menghitung jumlah kata di textarea dan memperbarui tampilan jumlah kata.
- $(document).ready() adalah event handler yang memastikan skrip dijalankan setelah dokumen selesai dimuat.
- fetch() digunakan untuk memanggil API prediksi dan mengambil hasilnya.
- Promise.all() digunakan untuk menjalankan beberapa permintaan API secara paralel.
- Hasil prediksi kemudian ditampilkan di tabel hasil.