LAMPIRAN

Lampiran 1 Surat Keterangan Hasil Pengecekan Turnitin

UNIVERSITAS DARMA PERSADA
UPT PERPUSTAKAAN
Gedung Rektorat Lantai 3,
Jl.Taman Malaka Selatan, Pondok Kelapa – Jakarta Timur 13450

SURAT KETERANGAN
HASIL PENGECEKAN TURNITIN

UPT Perpustakaan Universitas Darma Persada menerangkan telah selesai melakukan pemeriksaan duplikasi/*similarity* menggunakan perangkat lunak Turnitin terhadap hasil karya sebagai berikut:

Judul : Analisis sentimen berbasis aspek terhadap wisata kota dan kabupaten tegal dengan metode Transformers Indobert
Penulis : Ageng Prayogi
NIM : **2020230085**
Tgl pemeriksaan : 12 Februari 2025

Dengan hasil Tingkat Kesamaan (*similarity index*) **25%**

Demikian Surat Keterangan kami buat, untuk dipergunakan sebagaimana mestinya.

Jakarta, 12 Februari 2025

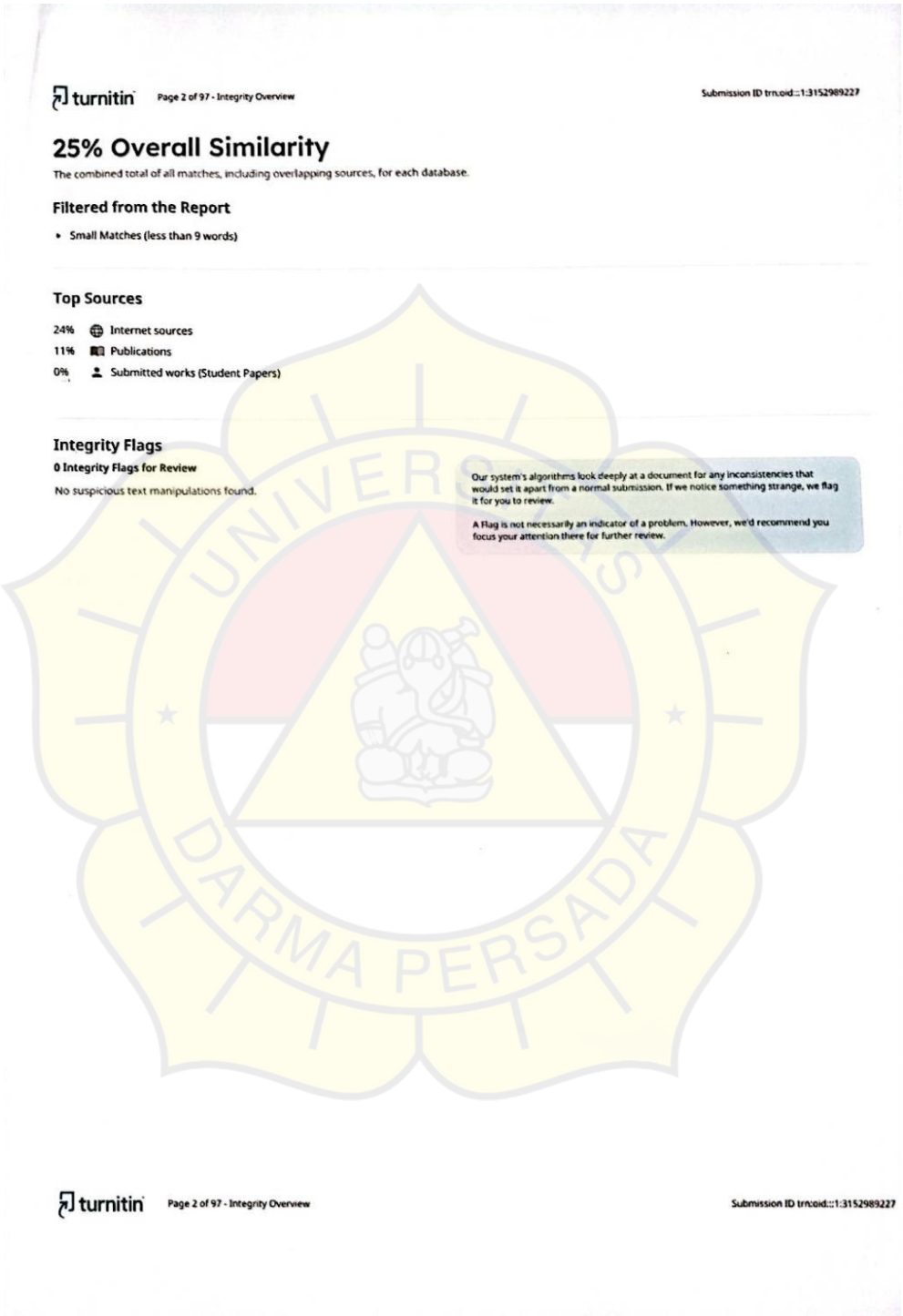
Ka.UPT Perpustakaan Unsada

Yus Rusmiyati, SS., MM

Batas maksimal similarity 30% untuk Fakultas Sastra dan Ekonomi

Batas maksimal similarity 25% untuk Fakultas Teknik, Kelautan dan Pasca Sarjana

Lampiran 2 Hasil Turnitin



turnitin Page 2 of 97 - Integrity Overview Submission ID trn:oid:::1:3152989227

25% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Small Matches (less than 9 words)

Top Sources

24% Internet sources
11% Publications
0% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

turnitin Page 2 of 97 - Integrity Overview Submission ID trn:oid:::1:3152989227

Top Sources

24% Internet sources
11% Publications
0% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Internet	repository.binadarma.ac.id 2%
2	Internet	perpustakaan.poltekkes-malang.ac.id 1%
3	Internet	ejournal.indobarunasional.ac.id 1%
4	Internet	repository.ulnjkt.ac.id 1%
5	Internet	jurnal.polibatam.ac.id 1%
6	Internet	journal.universitassuryadarma.ac.id 1%
7	Internet	ejournal.itn.ac.id <1%
8	Internet	docplayer.info <1%
9	Internet	digilib.unila.ac.id <1%
10	Internet	123dok.com <1%
11	Internet	dirdosen.budiluhur.ac.id <1%

12	Internet jim.teknokrat.ac.id	<1%
13	Internet etd.umy.ac.id	<1%
14	Internet journal.admi.or.id	<1%
15	Internet repository.wicida.ac.id	<1%
16	Internet repository.uin-suska.ac.id	<1%
17	Internet www.researchgate.net	<1%
18	Internet journal.undiknas.ac.id	<1%
19	Internet ecampus.pelitabangsa.ac.id	<1%
20	Internet jurnal.laii.or.id	<1%
21	Internet repositori.buddhidharma.ac.id	<1%
22	Internet repository.unuja.ac.id	<1%
23	Internet jursistekni.nusaputra.ac.id	<1%
24	Internet medium.com	<1%
25	Internet files.osf.io	<1%

26	Internet	
www.slideshare.net		<1%
27	Publication	
Zalzabila Rani, Bain Khusnul Khotimah, "ANALISIS SENTIMEN TERHADAP KARAPA...		<1%
28	Internet	
conference.binadarma.ac.id		<1%
29	Internet	
repository.radenfatah.ac.id		<1%
30	Internet	
repository.its.ac.id		<1%
31	Internet	
adsii.or.id		<1%
32	Internet	
repository.uinsu.ac.id		<1%
33	Internet	
isakzhang.github.io		<1%
34	Internet	
jurnal.likmi.ac.id		<1%
35	Internet	
text-id.123dok.com		<1%
36	Internet	
export.arxiv.org		<1%
37	Internet	
vdocuments.site		<1%
38	Internet	
github.com		<1%
39	Internet	
es.scribd.com		<1%

Lampiran 3 Kode Program Analisis Sentimen

```
# Import pustaka yang diperlukan
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer
from nltk import pos_tag
import pandas as pd
from transformers import AutoTokenizer,
AutoModelForSequenceClassification
from tqdm import tqdm
from nltk import word_tokenize
import nltk
import re

# Download NLTK resources (hanya punkt untuk word_tokenize)
nltk.download('punkt', quiet=True)

# Mapping aspek dan sentimen yang diperluas (seperti sebelumnya)
aspect_keywords = {
    "keindahan_alam": {
        "aspect": ["pemandangan", "alam", "gunung", "pantai",
"hutan", "udara", "cuaca", "matahari", "senja", "sunrise", "ombak",
"pasir", "air", "pohon", "langit", "kabut", "curug", "danau",
"telaga", "pohon pinus", "pepohonan", "kebun", "taman", "rumput"],
        "positive": ["memukau", "menakjubkan", "spektakuler",
"dahsyat", "memesona", "istimewa", "megah", "agung", "permai",
"elok", "rupawan", "cemerlang", "cantik", "molek", "menawan", "ayu",
"jelita", "hijau", "segar", "alami", "lestari", "teduh", "sejuk",
"nyaman", "unik", "khas", "langka", "menarik", "mengagumkan",
"memikat", "luar biasa", "eksotis"],
        "negative": ["kotor", "rusak", "tercemar", "tidak terawat",
"gersang", "tandus", "mati", "berantakan", "kacau", "semrawut",
"tidak teratur", "amburadul", "panas", "gerah", "sumuk", "terik",
"membakar", "menyengat", "beruap", "bising", "gaduh", "ribut",
"hingar-bingar", "memekakkan telinga", "berisik"]
    },
    "pengalaman_wisata": {
        "aspect": ["pengalaman", "liburan", "wisata", "perjalanan",
"waktu", "kenangan", "kesan", "aktivitas", "rekreasi", "healing",
"santai", "petualangan"],
        "positive": ["menyenangkan", "seru", "asyik", "gembira",
"riang", "bahagia", "suka cita", "menghibur", "memuaskan",
"mengagumkan", "luar biasa", "hebat", "fantastis", "dahsyat",
"memikat", "mempesona", "mengesankan", "berkesan", "istimewa",
"abadi", "unik", "mengagumkan", "bermakna", "sempurna", "cukup",
"ideal", "tepat", "sesuai", "pas", "mantap", "prima", "unggul",
"bermutu", "terpercaya", "terjamin"],
```

```
    "negative": ["mengecewakan", "buruk", "tidak memuaskan",
"menyedihkan", "menyebalkan", "menjengkelkan", "membosankan",
"monoton", "jenuh", "tidak menarik", "hambar", "datar", "sepi",
"menakutkan", "mengerikan", "menyeramkan", "mencekam", "genting",
"berbahaya", "tidak nyaman", "gelisah", "risih", "tidak tenang",
"tidak enak", "cemas", "was-was", "melelahkan", "letih", "capai",
"capek", "payah", "lesu", "lemah", "lunglai"]
    },
    "fasilitas_pelayanan": {
        "aspect": ["fasilitas", "pelayanan", "staff", "hotel",
"penginapan", "toilet", "kamar mandi", "mushola", "parkir", "tempat
duduk", "warung", "restoran", "kafe", "jajanan", "akses jalan",
"transportasi", "petunjuk arah", "penerangan", "jembatan",
"dermaga", "wahana", "permainan", "tiket", "keamanan", "kebersihan",
"peraturan", "tempat sampah"],
        "positive": ["ramah", "sopan", "baik hati", "akrab",
"hangat", "bersahabat", "santun", "penuh perhatian", "nyaman",
"tentram", "enak", "aman", "rileks", "betah", "santai", "damai",
"bersih", "higienis", "rapi", "terawat", "bebas kotoran",
"kinclong", "cemerlang", "lengkap", "memadai", "mencukupi",
"tersedia", "terfasilitasi", "terorganisir", "profesional", "ahli",
"kompeten", "cakap", "terampil", "mumpuni", "andal"],
        "negative": ["tidak ramah", "kasar", "cuek", "judes",
"ketus", "tidak sopan", "buruk hati", "tidak nyaman", "sempit",
"pengap", "panas", "tidak bersih", "berantakan", "tidak teratur",
"kotor", "jorok", "kumuh", "tidak higienis", "berbau", "berdebu",
"berlumpur", "tidak lengkap", "kurang", "minim", "terbatas", "tidak
memadai", "tidak mencukupi", "tidak profesional", "ceroboh",
"lambat", "tidak becus", "tidak ahli", "tidak kompeten"]
    },
        "kuliner": {
        "aspect": ["makanan", "minuman", "kuliner", "restoran",
"warung", "jajanan", "sate", "mendoan", "es kelapa", "kopi", "teh",
"nasi", "bakso", "mie", "seafood"],
        "positive": ["lezat", "enak", "nikmat", "sedap", "gurih",
"maknyus", "istimewa", "menggugah selera", "khas", "unik",
"otentik", "tradisional", "asli", "spesial", "segar", "baru",
"fresh", "alami", "sehat", "beragam", "bervariasi", "banyak
pilihan", "beraneka ragam", "kaya cita rasa"],
        "negative": ["tidak enak", "hambar", "tidak sedap", "tidak
nikmat", "menjijikkan", "pahit", "basi", "tidak segar", "lama",
"layu", "tidak fresh", "basi", "tengik", "berjamur", "mahal", "tidak
terjangkau", "boros", "menguras dompet", "berlebihan", "fantastis",
"tidak higienis", "kotor", "kumuh", "tidak bersih", "berbau",
"berdebu"]
    },
```

```
    "budaya_tradisi":{
         "aspect": ["budaya", "tradisi", "adat", "pertunjukan",
"kesenian", "lokal", "sejarah"],
        "positive": ["kaya", "beragam", "berlimpah", "beraneka
ragam", "makmur", "subur", "unik", "khas", "istimewa", "langka",
"otentik", "berbeda", "menarik", "memikat", "mempesona",
"mengagumkan", "mengesankan", "seru", "luhur", "mulia", "agung",
"terhormat", "tinggi", "bermartabat", "beradab"],
        "negative":["tertinggal", "kuno", "usang", "tidak relevan",
"ketinggalan zaman", "luntur", "pudar", "hilang", "berkurang",
"memudar", "surut", "kasar", "tidak sopan", "buruk", "tidak
beradab", "tidak beretika", "menyeramkan", "mengerikan",
"menakutkan", "mengancam", "mencekam"]
    }
}

sentiment_mapping = {"LABEL_0": "negatif", "LABEL_1": "netral",
"LABEL_2": "positif"}

# Load model dan tokenizer IndoBERT
model_name = "indobenchmark/indobert-large-p2"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model =
AutoModelForSequenceClassification.from_pretrained(model_name,
num_labels=3)

# Definisikan pipeline tanpa menyertakan token_type_ids
def sentiment_pipeline_no_token_type_ids(comment):
    tokens = tokenizer(comment, return_tensors="pt",
truncation=True, padding=True, max_length=512)
    if "token_type_ids" in tokens:
        tokens.pop("token_type_ids")
    return model(**tokens)

# Fungsi-fungsi yang digunakan tetap sama kecuali bagian pipeline:
def determine_sentiment_with_model(comment):
    inputs = tokenizer(comment, return_tensors="pt",
truncation=True, padding=True, max_length=512)
    if "token_type_ids" in inputs:
        del inputs["token_type_ids"]
    outputs = model(**inputs)
    logits = outputs.logits
    sentiment_id = logits.argmax().item()
    return sentiment_mapping[f"LABEL_{sentiment_id}"]

def determine_aspect(comment, keywords):
    comment = comment.lower()
    found_aspects = []
    for aspect, words in keywords.items():
        for word in words["aspect"]:
```

```
            if word in comment:
                found_aspects.append(aspect)

    return ", ".join(found_aspects) if found_aspects else "tidak
ada"

def determine_sentiment_with_keywords(comment, aspect_keywords):
    comment = comment.lower()

    positive_count = 0
    negative_count = 0

    for aspect, sentiments in aspect_keywords.items():
        for sentiment_type, words in sentiments.items():
            if sentiment_type != "aspect":
                for word in words:
                    if word in comment:
                        if sentiment_type == "positive":
                            positive_count += 1
                        elif sentiment_type == "negative":
                            negative_count += 1

    if positive_count > negative_count:
         return "positif"
    elif negative_count > positive_count:
          return "negatif"
    else:
         return "netral"

def extract_nouns(comment):
    tokens = word_tokenize(comment)
    nouns = [word for word in tokens if
re.search(r'\b(tempat|lokasi|pemandangan|alam|gunung|pantai|hutan|ud
ara|cuaca|matahari|senja|sunrise|ombak|pasir|air|pohon|langit|kabut|
curug|danau|telaga|pohon
pinus|pepohonan|kebun|taman|rumput|pengalaman|liburan|wisata|perjala
nan|waktu|kenangan|kesan|aktivitas|rekreasi|healing|santai|petualang
an|fasilitas|pelayanan|staff|hotel|penginapan|toilet|kamar
mandi|mushola|parkir|tempat duduk|warung|restoran|kafe|jajanan|akses
jalan|transportasi|petunjuk
arah|penerangan|jembatan|dermaga|wahana|permainan|tiket|keamanan|keb
ersihan|peraturan|tempat
sampah|makanan|minuman|kuliner|sate|mendoan|es
kelapa|kopi|teh|nasi|bakso|mie|seafood|budaya|tradisi|adat|pertunjuk
an|kesenian|lokal|sejarah)\b', word, re.IGNORECASE)]
    return nouns

# Fungsi untuk memproses file tetap sama kecuali memanggil pipeline
yang diperbaiki
def process_file(file_path, keywords):
```

```
      df = pd.read_csv(file_path, delimiter=",", encoding="utf-8",
on_bad_lines="skip")
    df = df.dropna(subset=["Comment"])
    df = df.reset_index(drop=True)

    processed_data = []

    for idx, row in tqdm(df.iterrows(), total=len(df),
desc="Processing comments"):
        comment = row["Comment"]

        # Task 1
        aspect_task1 = determine_aspect(comment, keywords)

        # Sentimen based on keywords (adjusted to use nested
structure)
        sentiment_task1 = determine_sentiment_with_keywords(comment,
aspect_keywords)

        # Task 2
        nouns_task2 = extract_nouns(comment)
        sentiment_task_nouns = determine_sentiment_with_keywords(",
".join(nouns_task2), aspect_keywords)

        # Task 3
        sentiment_task3 = determine_sentiment_with_model(comment)

        # Task 4
        adjectives_task4 = extract_adjectives(comment)
        sentiment_task_adjectives =
determine_sentiment_with_keywords(",
".join(adjectives_task4),aspect_keywords)

        processed_data.append({
            "Komentar": comment,
            "Aspek": aspect_task1,
             "Sentimen": sentiment_task1,
            "Sentimen_Kata_Sifat": sentiment_task_adjectives,
             "Sentimen_Kata_Benda": sentiment_task_nouns,
            "Sentimen_Model": sentiment_task3,
        })

    output_file = file_path.replace(".csv", "_annotated.csv")
    pd.DataFrame(processed_data).to_csv(output_file, index=False)
    print(f"Processed file saved at: {output_file}")
```

```
def extract_adjectives(comment):
    tokens = word_tokenize(comment)
    adjectives = [word for word in tokens if
re.search(r'\b(bagus|indah|nyaman|sejuk|terawat|mantap|keren|recomme
nded|puas|enak|murah|cocok|adem|asri|memukau|menakjubkan|elok|cantik
|unik|khas|cukup|lumayan|kotor|mahal|jorok|kecewa|payah|buruk|kumuh|
hambar|basi|tidak terjangkau|tidak sopan|beradab|pudar|usang)\b',
word, re.IGNORECASE)]
    return adjectives

# File paths
file_paths = {
    "PAI": "comments_PAI.csv",
    "Praban_Lintang": "comments_praban_lintang_cleaned.csv",
    "Guci": "comments_Guci.csv"
}

for place, file_path in file_paths.items():
    print(f"\nProcessing comments for {place}...")
    process_file(file_path, aspect_keywords)

#Mengulangi Proses anotasi untuk memastikan labeling oleh model
IndoBERT benar benar sesuai dengan isi komentar.
def predict_sentiment(text):
    """Memprediksi sentimen teks menggunakan IndoBERT."""
    inputs = tokenizer(text, return_tensors="pt", padding=True,
truncation=True, max_length=128)
    with torch.no_grad():
        outputs = model(**inputs)
    predicted_class = torch.argmax(outputs.logits, -1).item()
    return sentiment_mapping[predicted_class]

def detect_aspect(text, aspect_keywords):
    """Mendeteksi aspek dari teks."""
    detected_aspects = set()
    for aspect, keywords in aspect_keywords.items():
    # jika tidak ditemukan aspek dan ada kata sifat yang positif
atau negatif maka masukan pada aspek keindahan alam
    if not found_aspects:
        for aspect, sentiments in keywords.items():
            if aspect == "keindahan_alam":
                if any(keyword in text for keyword in
sentiments['positive'] + sentiments['negative']):
                    found_aspects.append(aspect)
                break

    return ", ".join(found_aspects) if found_aspects else "tidak
ada"

def determine_sentiment_with_keywords(comment, aspect_keywords):
    comment = comment.lower()
```

```
    positive_count = 0
    negative_count = 0

    for aspect, sentiments in aspect_keywords.items():
        for sentiment_type, words in sentiments.items():
            if sentiment_type != "aspect":
                for word in words:
                    if word in comment:
                        if sentiment_type == "positive":
                            positive_count += 1
                        elif sentiment_type == "negative":
                            negative_count += 1

    if positive_count > negative_count:
         return "positif"
    elif negative_count > positive_count:
          return "negatif"
    else:
         return "netral"

def extract_nouns(comment):
    tokens = word_tokenize(comment)
    nouns = [word for word in tokens if
re.search(r'\b(tempat|lokasi|pemandangan|alam|gunung|pantai|hutan|ud
ara|cuaca|matahari|senja|sunrise|ombak|pasir|air|pohon|langit|kabut|
curug|danau|telaga|pohon
pinus|pepohonan|kebun|taman|rumput|pengalaman|liburan|wisata|perjala
nan|waktu|kenangan|kesan|aktivitas|rekreasi|healing|santai|petualang
an|fasilitas|pelayanan|staff|hotel|penginapan|toilet|kamar
mandi|mushola|parkir|tempat duduk|warung|restoran|kafe|jajanan|akses
jalan|transportasi|petunjuk
arah|penerangan|jembatan|dermaga|wahana|permainan|tiket|keamanan|keb
ersihan|peraturan|tempat
sampah|makanan|minuman|kuliner|sate|mendoan|es
kelapa|kopi|teh|nasi|bakso|mie|seafood|budaya|tradisi|adat|pertunjuk
an|kesenian|lokal|sejarah)\b', word, re.IGNORECASE)]
    return nouns

def extract_adjectives(comment):
    tokens = word_tokenize(comment)
    adjectives = [word for word in tokens if
re.search(r'\b(bagus|indah|nyaman|sejuk|terawat|mantap|keren|recomme
nded|puas|enak|murah|cocok|adem|asri|memukau|menakjubkan|elok|cantik
|unik|khas|cukup|lumayan|kotor|mahal|jorok|kecewa|payah|buruk|kumuh|
hambar|basi|tidak terjangkau|tidak sopan|beradab|pudar|usang)\b',
word, re.IGNORECASE)]
    return adjectives

def fix_anotations(df, aspect_keywords):
    """Memperbaiki anotasi aspek, sentimen dan menggabungkan jadi 1
kolom."""
```

```
    fixed_rows = []

    for index, row in df.iterrows():
        comment = row['Komentar']
        predicted_sentiment = predict_sentiment(comment)
        detected_aspect = detect_aspect(comment, aspect_keywords)

        new_row = row.to_dict()
        new_row['(Aspek, Sentimen)'] = f"({detected_aspect},
{predicted_sentiment})"
        fixed_rows.append(new_row)

    return pd.DataFrame(fixed_rows)

def find_and_remove_duplicates(df, threshold=80):
    """Mencari dan menghapus duplikasi berdasarkan kemiripan teks
dan akhiran '...Baca'."""

    duplicates_indices = set()

    for i in range(len(df)):
        for j in range(i + 1, len(df)):
            comment1 = df['Komentar'][i]
            comment2 = df['Komentar'][j]

            if isinstance(comment1, str) and isinstance(comment2,
str):
                # Check for '...Baca' ending
                if comment1.endswith("…Baca") and comment1[:-5] ==
comment2:
                    duplicates_indices.add(j)
                    continue
                elif comment2.endswith("…Baca") and comment2[:-5] ==
comment1:
                    duplicates_indices.add(i)
                    continue

                # Fuzzy matching if not ending with '...Baca'
                similarity = fuzz.ratio(comment1, comment2)
                if similarity >= threshold:
                    duplicates_indices.add(j)

    df_cleaned = df.drop(duplicates_indices).reset_index(drop=True)
    return df_cleaned

# --- Main Program ---
file_paths = ["corrected_comments_PrabanLintang.csv",
"corrected_comments_PAI.csv", "corrected_comments_Guci.csv"]
all_dfs = []

for file_path in file_paths:
```

```
    print(f"Processing file: {file_path}")
    df = pd.read_csv(file_path)

    df_cleaned = find_and_remove_duplicates(df)
    print(f"  Duplicates removed. Remaining rows:
{len(df_cleaned)}")

    df_corrected = fix_anotations(df_cleaned, aspect_keywords)
    print(f"  Annotations corrected.")

    output_path = f"corrected_and_cleaned_{file_path.split('_')[-
1]}"
    df_corrected.to_csv(output_path, index=False)
    print(f"  Output saved to: {output_path}")
    all_dfs.append(df_corrected)

print("\nAll files processed.")



import re
import pandas as pd
import numpy as np
from transformers import BertTokenizer
import os
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import RandomOverSampler
import matplotlib.pyplot as plt
import seaborn as sns

# === Hyperparameters dan Konfigurasi Lainnya ===
MAX_LEN = 128
aspect_labels = sorted(list(aspect_keywords.keys())[:-1])
sentiment_labels = sorted(list(sentiment_mapping.values()))
joint_labels = []
for aspect in aspect_labels:
    for sentiment in sentiment_labels:
        joint_labels.append(f"({aspect}, {sentiment})")
joint_label_to_id = {label: i for i, label in
enumerate(joint_labels)}
file_paths = {
    "PAI": "Dataset_PAI_fix.csv",
    "Praban_Lintang": "Dataset_Praban_Lintang_fix.csv",
    "Guci": "Dataset_Guci_fix.csv"
}
tokenizer_path = "/content/"  # Ganti sesuai kebutuhan
vocab_file = os.path.join(tokenizer_path, "vocab.txt")
tokenizer_config_file = os.path.join(tokenizer_path,
"tokenizer_config.json")
special_tokens_file = os.path.join(tokenizer_path,
"special_tokens_map.json")
```

```
tokenizer = BertTokenizer.from_pretrained(
    "indobenchmark/indobert-base-p2",
    vocab_file=vocab_file,
    tokenizer_config_file=tokenizer_config_file,
    special_tokens_map_file=special_tokens_file,
    use_fast=True,
    trust_remote_code=True,
    force_download=True,
    from_slow=True,
)

# === Fungsi Pembantu (sama seperti sebelumnya) ===
def split_text_into_clauses(text): # ... (fungsi
split_text_into_clauses Anda, sama seperti sebelumnya) ...
    clauses =
re.split(r"[,.();]|(?:\s+(?:dan|tapi|namun|tetapi|cuma|hanya)\s+)",
text.lower())
    return [clause.strip() for clause in clauses if clause.strip()]
def extract_aspects_from_text(text, aspect_keywords): # ... (fungsi
extract_aspects_from_text Anda, sama seperti sebelumnya) ...
    clauses = split_text_into_clauses(text)
    detected = set()
    for clause in clauses:
        for asp_key, asp_data in aspect_keywords.items():
            if asp_key == "multi_aspect":
                continue
            for keyword in asp_data["aspect"]:
                if keyword in clause:
                    detected.add(asp_key)
    return list(detected)
def count_aspects(text, aspect_keywords):
    return len(extract_aspects_from_text(text, aspect_keywords))

# === Proses Utama untuk Setiap Dataset ===
for place, path in file_paths.items():
    print(f"\n=== Memproses dataset untuk {place} ===\n")
    df = pd.read_csv(path)
    df = df.dropna(subset=["Komentar", "Aspek",
"Sentimen"]).reset_index(drop=True)
    df["Aspek"] = df["Aspek"].map(lambda x: x.lower() if
isinstance(x, str) else x)
    df["Sentimen"] = df["Sentimen"].map(lambda x: x.lower() if
isinstance(x, str) else x)
    df["num_aspects"] = df["Komentar"].apply(lambda x:
count_aspects(x, aspect_keywords))

    # === 1. Split Data Test (15%) ===
    train_val_df, test_df = train_test_split(df, test_size=0.15,
stratify=df["num_aspects"], random_state=42)
    test_csv_path = path.replace("_fix.csv", "_test.csv")
    test_df.to_csv(test_csv_path, index=False)
```

```
    print(f"Test set untuk {place} disimpan ke {test_csv_path}")

    # === 2. Split Data Train dan Validasi (dari sisa 85%) ===
    train_df, val_df = train_test_split(train_val_df,
test_size=0.15/0.85, stratify=train_val_df["num_aspects"],
random_state=42) # Proporsi validasi 15% dari total awal

    # === 3. Oversampling Hanya Data Train ===
    print("\nDistribusi Kelas Aspek Data Train Sebelum
Oversampling:")
    print(train_df["Aspek"].value_counts()) # Tampilkan distribusi
sebelum oversampling

    train_texts_original = train_df["Komentar"].tolist()
    train_sentiments_original = train_df["Sentimen"].tolist()
    train_aspects_original = train_df["Aspek"].tolist() # Simpan
aspek asli untuk analisis distribusi

    train_data_oversample = train_df[["Komentar", "Aspek",
"Sentimen"]]
    ros = RandomOverSampler(random_state=42)
    train_data_resampled, _ =
ros.fit_resample(train_data_oversample, train_df["Aspek"]) #
Oversample berdasarkan kolom 'Aspek' (multi-label)
    train_df_resampled = pd.DataFrame(train_data_resampled,
columns=["Komentar", "Aspek", "Sentimen"])

    print("\nDistribusi Kelas Aspek Data Train Setelah
Oversampling:")
    print(train_df_resampled["Aspek"].value_counts()) # Tampilkan
distribusi setelah oversampling

    # === Visualisasi Distribusi Kelas Aspek (Sebelum dan Sesudah
Oversampling) ===
    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    sns.countplot(y=train_aspects_original,
order=pd.Series(train_aspects_original).value_counts().index) #
Hitung frekuensi langsung dari list
    plt.title(f"Distribusi Aspek - Train (Sebelum Oversampling) -
{place}")
    plt.xlabel("Jumlah Komentar")
    plt.ylabel("Aspek")

    aspects_resampled_list = train_df_resampled["Aspek"].tolist() #
Ambil list aspek dari dataframe resampled
    plt.subplot(1, 2, 2)
    sns.countplot(y=aspects_resampled_list,
order=pd.Series(aspects_resampled_list).value_counts().index) #
Hitung frekuensi langsung dari list
```

```
   plt.title(f"Distribusi Aspek - Train (Setelah Oversampling) -
{place}")
    plt.xlabel("Jumlah Komentar")
    plt.ylabel("Aspek")

    plt.tight_layout()
    plt.show()

    # Simpan data train, validasi, dan test (opsional, untuk
penggunaan selanjutnya)
    train_csv_path_resampled = path.replace("_fix.csv",
"_train_resampled.csv")
    train_df_resampled.to_csv(train_csv_path_resampled, index=False)
    val_csv_path = path.replace("_fix.csv", "_val.csv")
    val_df.to_csv(val_csv_path, index=False)
    print(f"\nData train resampled, validasi, dan test untuk {place}
disimpan.")

    # Data siap untuk training di Cell Kode 2
    train_texts = train_df_resampled["Komentar"].tolist() # Gunakan
data train *resampled* untuk training
    train_sentiments = train_df_resampled["Sentimen"].tolist() #
Gunakan data train *resampled* untuk training
    val_texts = val_df["Komentar"].tolist() # Gunakan data validasi
*asli* (tidak di-oversample)
    val_sentiments = val_df["Sentimen"].tolist() # Gunakan data
validasi *asli* (tidak di-oversample)
    test_texts = test_df["Komentar"].tolist() # Gunakan data test
*asli* (tidak di-oversample)
    test_sentiments = test_df["Sentimen"].tolist() # Gunakan data
test *asli* (tidak di-oversample)

    print("\nData train, validasi, dan test siap untuk training di
Cell Kode 2.")


import re
import pandas as pd
import numpy as np
from transformers import BertTokenizer, AutoModel, AutoConfig
import torch
import os
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import StratifiedKFold,
train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from tqdm import tqdm
from imblearn.over_sampling import RandomOverSampler
```

```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

# === Hyperparameters ===
BEST_DROPOUT_RATE = 0.5
BEST_LEARNING_RATE = 2e-5
BEST_WEIGHT_DECAY = 0.0001
BEST_BATCH_SIZE = 64
MAX_LEN = 128
EPOCHS = 5

# === Aspect Keywords dan Sentiment Mapping ===
aspect_keywords = {
    "keindahan_alam": {
        "aspect": ["pemandangan", "alam", "gunung", "pantai",
"hutan", "udara", "cuaca", "matahari", "senja", "sunrise", "ombak",
"pasir", "air", "pohon", "langit", "kabut", "curug", "danau",
"telaga", "pohon pinus", "pepohonan", "kebun", "taman", "rumput",
"sungai", "laut", "bukit", "lembah", "rawa", "goa", "batu",
"karang", "tebing", "kawah", "flora", "fauna", "biodiversitas",
"ekosistem", "panorama", "lanskap", "horizon", "cahaya", "sinar",
"awan", "pelangi", "embun", "salju"],
        "positive": ["memukau", "menakjubkan", "spektakuler",
"dahsyat", "memesona", "istimewa", "megah", "agung", "permai",
"elok", "rupawan", "cemerlang", "cantik", "molek", "menawan", "ayu",
"jelita", "hijau", "segar", "alami", "lestari", "teduh", "sejuk",
"nyaman", "unik", "khas", "langka", "menarik", "mengagumkan",
"memikat", "luar biasa", "eksotis", "indah", "bersih", "terang",
"cerah", "damai", "tenang", "syahdu", "mempesona", "memanjakan
mata"],
        "negative": ["kotor", "rusak", "tercemar", "tidak terawat",
"gersang", "tandus", "mati", "berantakan", "kacau", "semrawut",
"tidak teratur", "amburadul", "panas", "gerah", "sumuk", "terik",
"membakar", "menyengat", "beruap", "bising", "gaduh", "ribut",
"hingar-bingar", "memekakkan telinga", "berisik", "gelap", "suram",
"mendung", "kumuh", "berlumpur", "becek", "jorok", "berbau", "tidak
sedap"]
    },
    "pengalaman_wisata": {
        "aspect": ["pengalaman", "liburan", "wisata", "perjalanan",
"waktu", "kenangan", "kesan", "aktivitas", "rekreasi", "healing",
"santai", "petualangan", "atraksi", "kunjungan", "eksplorasi",
"destinasi", "tempat wisata", "lokasi", "spot", "tujuan", "agenda",
"rencana", "itinerary", "trip", "getaway", "plesiran", "tour"],
        "positive": ["bagus", "cocok", "mantap", "menyenangkan",
"seru", "asyik", "gembira", "riang", "bahagia", "suka cita",
"menghibur", "memuaskan", "mengagumkan", "luar biasa", "hebat",
"fantastis", "dahsyat", "memikat", "mempesona", "mengesankan",
"berkesan", "istimewa", "abadi", "unik", "mengagumkan", "bermakna",
```

```
"sempurna", "cukup", "ideal", "tepat", "sesuai", "pas", "mantap",
"prima", "unggul", "bermutu", "terpercaya", "terjamin", "keren",
"populer", "terkenal", "ramai", "rame", "hidup", "meriah",
"bergairah", "semangat", "antusias", "positif", "inspiratif",
"edukatif", "menambah wawasan", "kaya informasi"],
        "negative": ["mengecewakan", "buruk", "tidak memuaskan",
"menyedihkan", "menyebalkan", "menjengkelkan", "membosankan",
"monoton", "jenuh", "tidak menarik", "hambar", "datar", "sepi",
"menakutkan", "mengerikan", "menyeramkan", "mencekam", "genting",
"berbahaya", "tidak nyaman", "gelisah", "risih", "tidak tenang",
"tidak enak", "cemas", "was-was", "melelahkan", "letih", "capai",
"capek", "payah", "lesu", "lemah", "lunglai", "terpaksa",
"dipaksakan", "sia-sia", "percuma", "buang-buang waktu", "tidak
recommended", "kurang greget"]
    },
    "fasilitas_pelayanan": {
        "aspect": ["fasilitas", "pelayanan", "staff", "hotel",
"penginapan", "toilet", "kamar mandi", "mushola", "parkir", "tempat
duduk", "warung", "restoran", "kafe", "jajanan", "akses jalan",
"transportasi", "petunjuk arah", "penerangan", "jembatan",
"dermaga", "wahana", "permainan", "tiket", "keamanan", "kebersihan",
"peraturan", "tempat sampah", "pusat informasi", "pemandu wisata",
"guide", "porter", "resepsionis", "lounge", "wifi", "sinyal",
"listrik", "air bersih", "ruang tunggu", "area bermain", "spot
foto", "instalasi seni", "toko oleh-oleh", "ATM", "money changer",
"pos kesehatan"],
        "positive": ["ramah", "sopan", "baik hati", "akrab",
"hangat", "bersahabat", "santun", "penuh perhatian", "nyaman",
"tentram", "enak", "aman", "rileks", "betah", "santai", "damai",
"bersih", "higienis", "rapi", "terawat", "bebas kotoran",
"kinclong", "cemerlang", "lengkap", "memadai", "mencukupi",
"tersedia", "terfasilitasi", "terorganisir", "profesional", "ahli",
"kompeten", "cakap", "terampil", "mumpuni", "andal", "cepat",
"efisien", "responsif", "tanggap", "mudah diakses", "terjangkau",
"praktis", "modern", "canggih", "update", "informatif", "helpful"],
        "negative": ["tidak ramah", "kasar", "cuek", "judes",
"ketus", "tidak sopan", "buruk hati", "tidak nyaman", "sempit",
"pengap", "panas", "tidak bersih", "berantakan", "tidak teratur",
"kotor", "jorok", "kumuh", "tidak higienis", "berbau", "berdebu",
"berlumpur", "tidak lengkap", "kurang", "minim", "terbatas", "tidak
memadai", "tidak mencukupi", "tidak profesional", "ceroboh",
"lambat", "tidak becus", "tidak ahli", "tidak kompeten", "sulit
diakses", "mahal", "membebani", "rumit", "ribet", "manual",
"ketinggalan zaman", "tidak informatif", "tidak membantu", "ugal-
ugalan", "semrawut"]
    },
    "kuliner": {
        "aspect": ["makanan", "minuman", "kuliner", "restoran",
"warung", "jajanan", "sate", "mendoan", "es kelapa", "kopi", "teh",
"nasi", "bakso", "mie", "seafood", "gudeg", "rendang", "gado-gado",
"pecel", "rujak", "soto", "sop", "bubur", "roti", "kue", "camilan",
```

```
"oleh-oleh", "bumbu", "rasa", "cita rasa", "aroma", "tekstur",
"penyajian", "porsi", "bahan baku", "menu", "hidangan", "santapan",
"jamuan"],
        "positive": ["lezat", "enak", "nikmat", "sedap", "gurih",
"maknyus", "istimewa", "menggugah selera", "khas", "unik",
"otentik", "tradisional", "asli", "spesial", "segar", "baru",
"fresh", "alami", "sehat", "beragam", "bervariasi", "banyak
pilihan", "beraneka ragam", "kaya cita rasa", "mantap", "juara",
"nampol", "nagih", "lezatos", "yummy", "delicious", "tasteful",
"flavorful", "appetizing", "mouthwatering", "mengenyangkan", "pas di
lidah"],
        "negative": ["tidak enak", "hambar", "tidak sedap", "tidak
nikmat", "menjijikkan", "pahit", "basi", "tidak segar", "lama",
"layu", "tidak fresh", "basi", "tengik", "berjamur", "mahal", "tidak
terjangkau", "boros", "menguras dompet", "berlebihan", "fantastis",
"tidak higienis", "kotor", "kumuh", "tidak bersih", "berbau",
"berdebu", "asing", "aneh", "overpriced", "kemahalan", "kurang
bumbu", "keasinan", "kemanisan", "pedas berlebihan", "tidak matang",
"mentah", "gosong"]
    },
    "multi_aspect": {
        "aspect": []
    }
}

sentiment_mapping = {"LABEL_0": "negatif", "LABEL_1": "netral",
"LABEL_2": "positif"}

# === Buat Joint Label Mapping (4 aspek x 3 sentimen = 12) ===
aspect_labels = sorted(list(aspect_keywords.keys())[:-1])
sentiment_labels = sorted(list(sentiment_mapping.values()))
joint_labels = []
for aspect in aspect_labels:
    for sentiment in sentiment_labels:
        joint_labels.append(f"({aspect}, {sentiment})")
joint_label_to_id = {label: i for i, label in
enumerate(joint_labels)}

# === File Paths untuk tiap dataset ===
file_paths = {
    "PAI": "Dataset_PAI_fix.csv",
    "Praban_Lintang": "Dataset_Praban_Lintang_fix.csv",
    "Guci": "Dataset_Guci_fix.csv"
}

# === Inisialisasi Tokenizer ===
tokenizer_path = "/content/"  # Ganti sesuai kebutuhan
vocab_file = os.path.join(tokenizer_path, "vocab.txt")
tokenizer_config_file = os.path.join(tokenizer_path,
"tokenizer_config.json")
```

```
special_tokens_file = os.path.join(tokenizer_path,
"special_tokens_map.json")

tokenizer = BertTokenizer.from_pretrained(
    "indobenchmark/indobert-base-p2",
    vocab_file=vocab_file,
    tokenizer_config_file=tokenizer_config_file,
    special_tokens_map_file=special_tokens_file,
    use_fast=True,
    trust_remote_code=True,
    force_download=True,
    from_slow=True,
)

# === Fungsi Pembantu: Split Komentar Menjadi Klausa ===
def split_text_into_clauses(text):
    """
    Memecah teks menjadi klausa berdasarkan tanda baca (koma, titik,
titik koma)
    dan konjungsi (dan, tapi, namun, tetapi, cuma, hanya).
    Mengembalikan list klausa yang tidak kosong.
    """
    clauses =
re.split(r"[,.();]|(?:\s+(?:dan|tapi|namun|tetapi|cuma|hanya)\s+)",
text.lower())
    return [clause.strip() for clause in clauses if clause.strip()]

# === Fungsi Pembantu: Ekstrak Aspek dari Teks (untuk encoding
target) ===
def extract_aspects_from_text(text, aspect_keywords):
    """
    Mengembalikan list aspek yang terdeteksi dari teks dengan
menggunakan fungsi split_text_into_clauses.
    """
    clauses = split_text_into_clauses(text)
    detected = set()
    for clause in clauses:
        for asp_key, asp_data in aspect_keywords.items():
            if asp_key == "multi_aspect":
                continue
            for keyword in asp_data["aspect"]:
                if keyword in clause:
                    detected.add(asp_key)
    return list(detected)

# === Custom Collate Function ===
def custom_collate_fn(batch):
    batch = [item for item in batch if item["input_ids"] is not
None]
    if not batch:
        return {
```

```
            "input_ids": torch.zeros((1, MAX_LEN),
dtype=torch.long),
            "attention_mask": torch.zeros((1, MAX_LEN),
dtype=torch.long),
            "joint_labels": torch.zeros((1, len(joint_label_to_id)),
dtype=torch.float)
        }
    input_ids = torch.stack([item["input_ids"] for item in batch])
    attention_mask = torch.stack([item["attention_mask"] for item in
batch])
    joint_labels = torch.stack([item["joint_labels"] for item in
batch])
    return {
        "input_ids": input_ids,
        "attention_mask": attention_mask,
        "joint_labels": joint_labels
    }

# === Dataset yang Memecah Komentar Menjadi Klausa ===
class MultiTaskClauseDataset(Dataset):
    def __init__(self, texts, sentiments, tokenizer, max_len,
aspect_keywords, joint_label_to_id):
        """
        Menerima list komentar dan sentimen, kemudian memecah setiap
komentar menjadi klausa.
        Setiap klausa menjadi satu sampel dengan target yang
dihitung secara mandiri.
        """
        self.samples = []  # List tuple: (clause, sentiment)
        for text, sentiment in zip(texts, sentiments):
            clauses = split_text_into_clauses(text)
            if clauses:
                for clause in clauses:
                    self.samples.append((clause, sentiment))
            else:
                # Jika tidak ada klausa terpisah, gunakan teks utuh
                self.samples.append((text, sentiment))
        self.tokenizer = tokenizer
        self.max_len = max_len
        self.aspect_keywords = aspect_keywords
        self.joint_label_to_id = joint_label_to_id

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

    def __getitem__(self, idx):
        clause, sentiment = self.samples[idx]
        # Hitung target multi-label untuk klausa ini
        target = self.encode_joint_label(clause, sentiment)
        encoding = self.tokenizer.encode_plus(
            clause,
```

```
            add_special_tokens=True,
            max_length=self.max_len,
            return_token_type_ids=False,
            padding="max_length",
            truncation=True,
            return_attention_mask=True,
            return_tensors="pt"
        )
        return {
            "input_ids": encoding["input_ids"].flatten(),
            "attention_mask": encoding["attention_mask"].flatten(),
            "joint_labels": target
        }

    def encode_joint_label(self, text, sentiment):
        detected_aspects = extract_aspects_from_text(text,
self.aspect_keywords)
        target = np.zeros(len(self.joint_label_to_id),
dtype=np.float32)
        if detected_aspects:
            for asp in detected_aspects:
                joint_label_str = f"({asp}, {sentiment})"
                if joint_label_str in self.joint_label_to_id:
                    idx = self.joint_label_to_id[joint_label_str]
                    target[idx] = 1.0
        else:
            # Fallback: jika tidak ada aspek terdeteksi, gunakan
default
            fallback_label = "(pengalaman_wisata, netral)"
            if fallback_label in self.joint_label_to_id:
                idx = self.joint_label_to_id[fallback_label]
                target[idx] = 1.0
        return torch.tensor(target)

# === Model Definition ===
class MultiTaskModel(nn.Module):
    def __init__(self, base_model, dropout_rate):
        super(MultiTaskModel, self).__init__()
        self.base_model = base_model
        self.embedding_dim = base_model.config.hidden_size  #
Biasanya 768
        self.dropout = nn.Dropout(dropout_rate)
        # Layer output: langsung memetakan 768 ke 12 kelas
        self.joint_classifier = nn.Linear(self.embedding_dim, 12)

    def forward(self, input_ids, attention_mask):
        outputs = self.base_model(input_ids=input_ids,
attention_mask=attention_mask)
        cls_output = outputs.last_hidden_state[:, 0, :]  # Ambil
token CLS
        cls_output = self.dropout(cls_output)
```

```
        joint_logits = self.joint_classifier(cls_output)
        return joint_logits

# === Training and Evaluation Function (Multi-label) ===
def train_and_evaluate(model, train_loader, val_loader, optimizer,
scheduler, device):
    all_val_preds = []
    all_val_targets = []
    for epoch in range(EPOCHS):
        # --- Training Phase ---
        model.train()
        train_losses = []
        train_accs = []
        for batch in tqdm(train_loader, desc=f"Epoch {epoch + 1} -
Training"):
            input_ids = batch["input_ids"].to(device)
            attention_mask = batch["attention_mask"].to(device)
            targets = batch["joint_labels"].to(device)  # [batch,
num_classes]

            optimizer.zero_grad()
            logits = model(input_ids=input_ids,
attention_mask=attention_mask)
            loss_fn = nn.BCEWithLogitsLoss()
            loss = loss_fn(logits, targets)
            loss.backward()
            optimizer.step()
            train_losses.append(loss.item())

            preds = (torch.sigmoid(logits) > 0.5).float()
            acc = (preds == targets).float().mean().item()
            train_accs.append(acc)

        # --- Validation Phase ---
        model.eval()
        val_losses = []
        val_accs = []
        val_preds = []
        val_targets = []
        with torch.no_grad():
            for batch in val_loader:
                input_ids = batch["input_ids"].to(device)
                attention_mask = batch["attention_mask"].to(device)
                targets = batch["joint_labels"].to(device)
                logits = model(input_ids=input_ids,
attention_mask=attention_mask)
                loss = nn.BCEWithLogitsLoss()(logits, targets)
                val_losses.append(loss.item())

                preds = (torch.sigmoid(logits) > 0.5).float()
                acc = (preds == targets).float().mean().item()
```

```
                val_accs.append(acc)
                val_preds.append(preds.cpu().numpy())
                val_targets.append(targets.cpu().numpy())

        avg_train_loss = np.mean(train_losses)
        avg_val_loss = np.mean(val_losses)
        avg_train_acc = np.mean(train_accs)
        avg_val_acc = np.mean(val_accs)
        scheduler.step(avg_val_loss)
        print(f"Epoch {epoch + 1}: Train Loss: {avg_train_loss:.4f},
Train Acc: {avg_train_acc:.4f}, Val Loss: {avg_val_loss:.4f}, Val
Acc: {avg_val_acc:.4f}")

        all_val_preds.append(np.concatenate(val_preds, axis=0))
        all_val_targets.append(np.concatenate(val_targets, axis=0))

    final_val_preds = np.concatenate(all_val_preds, axis=0)
    final_val_targets = np.concatenate(all_val_targets, axis=0)
    return final_val_preds, final_val_targets

# === Pembagian Data: Stratifikasi Berdasarkan Jumlah Aspek yang
Terdeteksi ===
# Karena setiap komentar bisa membahas lebih dari satu aspek, kita
hitung jumlah aspek terdeteksi dari teks asli.
def count_aspects(text, aspect_keywords):
    return len(extract_aspects_from_text(text, aspect_keywords))

# === Setup Device dan Base Model ===
device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
config = AutoConfig.from_pretrained("indobenchmark/indobert-base-
p2", output_hidden_states=True)
base_model = AutoModel.from_pretrained("indobenchmark/indobert-base-
p2", config=config)

# === Proses Utama untuk Setiap Dataset ===
for place, path in file_paths.items():
    print(f"\n=== Memproses dataset untuk {place} ===\n")
    df = pd.read_csv(path)
    df = df.dropna(subset=["Komentar", "Aspek",
"Sentimen"]).reset_index(drop=True)
    df["Aspek"] = df["Aspek"].map(lambda x: x.lower() if
isinstance(x, str) else x)
    df["Sentimen"] = df["Sentimen"].map(lambda x: x.lower() if
isinstance(x, str) else x)
    # Hitung jumlah aspek terdeteksi untuk stratifikasi
    df["num_aspects"] = df["Komentar"].apply(lambda x:
count_aspects(x, aspect_keywords))

    # Pisahkan test set sebanyak 15% (stratified berdasarkan
num_aspects)
```

```
    train_val_df, test_df = train_test_split(df, test_size=0.15,
stratify=df["num_aspects"], random_state=42)
    test_csv_path = path.replace("_fix.csv", "_test.csv")
    test_df.to_csv(test_csv_path, index=False)
    print(f"Test set untuk {place} disimpan ke {test_csv_path}")

    # Untuk splitting training/validation, gunakan stratifikasi
berdasarkan num_aspects
    texts = train_val_df["Komentar"].tolist()
    sentiments = train_val_df["Sentimen"].tolist()
    strat_labels = train_val_df["num_aspects"].tolist()

    # Oversampling pada data training (pada tingkat komentar)
    train_val_data = train_val_df[["Komentar", "Aspek", "Sentimen"]]
    ros = RandomOverSampler(random_state=42)
    train_val_resampled, _ = ros.fit_resample(train_val_data,
train_val_df["num_aspects"])
    train_val_resampled = pd.DataFrame(train_val_resampled,
columns=["Komentar", "Aspek", "Sentimen"])

    # Split resampled data menjadi training dan validation
menggunakan StratifiedKFold (berdasarkan num_aspects)
    texts = train_val_resampled["Komentar"].tolist()
    sentiments = train_val_resampled["Sentimen"].tolist()
    strat_labels = train_val_resampled["Komentar"].apply(lambda x:
count_aspects(x, aspect_keywords)).tolist()

    skf = StratifiedKFold(n_splits=2, shuffle=True, random_state=42)
    fold_idx = 1
    for train_idx, val_idx in skf.split(texts, strat_labels):
        print(f"\n--- Fold {fold_idx} untuk {place} ---\n")
        train_texts = [texts[i] for i in train_idx]
        train_sentiments = [sentiments[i] for i in train_idx]
        val_texts = [texts[i] for i in val_idx]
        val_sentiments = [sentiments[i] for i in val_idx]

        # Buat dataset menggunakan MultiTaskClauseDataset (yang
memecah komentar menjadi klausa)
        train_dataset = MultiTaskClauseDataset(
            train_texts,
            train_sentiments,
            tokenizer,
            MAX_LEN,
            aspect_keywords,
            joint_label_to_id
        )
        val_dataset = MultiTaskClauseDataset(
            val_texts,
            val_sentiments,
            tokenizer,
            MAX_LEN,
```

```
            aspect_keywords,
            joint_label_to_id
        )

        train_loader = DataLoader(train_dataset,
batch_size=BEST_BATCH_SIZE, shuffle=True,
collate_fn=custom_collate_fn)
        val_loader = DataLoader(val_dataset,
batch_size=BEST_BATCH_SIZE, shuffle=False,
collate_fn=custom_collate_fn)

        # Inisialisasi model, optimizer, dan scheduler
        model = MultiTaskModel(base_model,
BEST_DROPOUT_RATE).to(device)
        optimizer = torch.optim.AdamW(model.parameters(),
lr=BEST_LEARNING_RATE, weight_decay=BEST_WEIGHT_DECAY)
        scheduler =
torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min',
factor=0.5, patience=2)

        # Training dan evaluasi per fold
        val_preds, val_targets = train_and_evaluate(model,
train_loader, val_loader, optimizer, scheduler, device)

        # Untuk evaluasi akhir, kita konversi prediksi multi-label
ke label tunggal dengan argmax
        single_val_preds = np.argmax(val_preds, axis=1)
        single_val_targets = np.argmax(val_targets, axis=1)

        print("\nClassification Report untuk Joint Task (Aspect,
Sentiment):")
        print(classification_report(single_val_targets,
single_val_preds, labels=list(range(len(joint_labels))),
target_names=joint_labels, zero_division=0))

        cm = confusion_matrix(single_val_targets, single_val_preds)
        plt.figure(figsize=(10, 10))
        sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
xticklabels=joint_labels, yticklabels=joint_labels)
        plt.title(f"Confusion Matrix untuk Joint Task - {place} -
Fold {fold_idx}")
        plt.xlabel("Predicted")
        plt.ylabel("True")
        plt.show()

        fold_idx += 1

import torch

# Misalkan `model` adalah model yang telah dilatih
# Misalkan `optimizer` adalah optimizer yang digunakan (jika perlu)
```

```
torch.save({
    'model_state_dict': model.state_dict(),          # Simpan
parameter model
}, 'model_last.pth')

print("Model berhasil disimpan!")

checkpoint = torch.load('model_last.pth')
# Periksa apakah semua parameter cocok
missing_keys, unexpected_keys =
model.load_state_dict(checkpoint['model_state_dict'], strict=False)
print(f"Missing keys: {missing_keys}")
print(f"Unexpected keys: {unexpected_keys}")

print("Model berhasil disimpan!")
```