LAMPIRAN

Lampiran 1. Riwayat Hidup

RIWAYAT HIDUP

Aprilia Monica Sari lahir di Singaraja pada 18 April 2003. Penulis lahir dari pasangan suami istri, Bapak Nyoman Sutawan dan Ibu Putu Suasmini. Penulis berkebangsaan Indonesia dan beragama Budha. Kini penulis beralamat di Jalan Merpati No 2, Keluruhan Kaliuntu, Kecamatan Buleleng, Kota Singaraja, Kabupaten Buleleng, Provinsi Bali. Penulis penyelesaikan pendidikan dasar di SD Negeri 3 Banjar Jawa dan lulus pada tahun 2015. Kemudian penulis melanjutkan pendidikan di SMP Negeri 2 Singaraja dan lulus pada tahun 2018. Pada tahun 2021, penulis lulus dari SMA Negeri 1 Singaraja jurusan Ilmu Pengetahuan Alam dan melanjutkan studi (S1) di Universitas Pendidikan Ganesha dengan Program Studi Sistem Informasi, Jurusan Teknik Informatika.

Lampiran 2. Surat Permohonan Pelabelan Data Validator 1

KEMENTERIAN PENDIDIKAN TINGGI, SAINS,
DAN TEKNOLOGI
UNIVERSITAS PENDIDIKAN GANESHA
FAKULTAS TEKNIK DAN KEJURUAN
Jalan Udayana Nomor 11 Singaraja - Bali Kode Pos 81116
Telepon (0362) 22570 Email: ftk@undiksha.ac.id Laman: http://ftk.undiksha.ac.id

Nomor : **2312**/UN48.11.1/DI.03.00/2025 Singaraja, 19 Agustus 2025
Perihal : Surat Permohonan Pengambilan Data

Yth. Kepala SD Negeri 1 Astina
di tempat

Dengan hormat, sehubungan dengan proses penyelesaian Tugas Akhir/Skripsi, maka melalui surat ini kami mohon Bapak/Ibu berkenan memberikan data yang terkait dengan data yang dibutuhkan. Adapun mahasiswa yang akan melakukan pengambilan data seperti tersebut di bawah ini:

Nama	:	Aprilia Monica Sari
NIM	:	2115091063
Program Studi	:	Sistem Informasi
Jurusan	:	Teknik Informatika
Data yang dibutuhkan	:	Pelabelan data penelitian skripsi
Judul Penelitian	:	Analisis Sentimen Program Petani Milenial Pada Komentar Tiktok Menggunakan Metode IndoBERT

Demikian kami sampaikan, atas perhatian dan kerjasamanya, diucapkan terima kasih.

a.n Dekan
Wakil Dekan Bidang Akademik,

Made Windu Antara Kesiman
NIP 198211112008121001

Catatan :
- UU ITE No. 11 Tahun 2008 Pasal 5 ayat 1"Informasi Elektronik dan/atau Dokumen Elektornik dan/atau hasil cetaknya merupakan alat bukti hukum yang sah"
- Dokumen ini tertanda ditandatangani secara elektronik menggunakan sertifikat elektronik yang diterbitkan BsrE
- Surat ini dapat dibuktikan keasliannya dengan menggunakan *qr code* yang telah tersedia

Lampiran 3. Surat Permohonan Pelabelan Data Validator 2

KEMENTERIAN PENDIDIKAN TINGGI, SAINS, DAN TEKNOLOGI
UNIVERSITAS PENDIDIKAN GANESHA
FAKULTAS TEKNIK DAN KEJURUAN
Jalan Udayana Nomor 11 Singaraja - Bali Kode Pos 81116
Telepon (0362) 22570 Email: ftk@undiksha.ac.id Laman: http://ftk.undiksha.ac.id

Nomor : **2313**/UN48.11.1/DI.03.00/2025 Singaraja, 19 Agustus 2025
Perihal : Surat Permohonan Pengambilan Data

Yth. Kepala SMP Negeri 1 Singaraja
di tempat

Dengan hormat, sehubungan dengan proses penyelesaian Tugas Akhir/Skripsi, maka melalui surat ini kami mohon Bapak/Ibu berkenan memberikan data yang terkait dengan data yang dibutuhkan. Adapun mahasiswa yang akan melakukan pengambilan data seperti tersebut di bawah ini:

Nama	:	Aprilia Monica Sari
NIM	:	2115091063
Program Studi	:	Sistem Informasi
Jurusan	:	Teknik Informatika
Data yang dibutuhkan	:	Pelabelan data penelitian skripsi
Judul Penelitian	:	Analisis Sentimen Program Petani Milenial Pada Komentar Tiktok Menggunakan Metode IndoBERT

Demikian kami sampaikan, atas perhatian dan kerjasamanya, diucapkan terima kasih.

a.n Dekan
Wakil Dekan Bidang Akademik,

Made Windu Antara Kesiman
NIP 198211112008121001

Balai Sertifikasi Elektronik

Catatan :
- UU ITE No. 11 Tahun 2008 Pasal 5 ayat 1"Informasi Elektronik dan/atau Dokumen Elektornik dan/atau hasil cetaknya merupakan alat bukti hukum yang sah"
- Dokumen ini tertanda ditandatangani secara elektronik menggunakan sertifikat elektronik yang diterbitkan BsrE
- Surat ini dapat dibuktikan keasliannya dengan menggunakan *qr code* yang telah tersedia

Lampiran 4. Surat Pernyataan Hasil Validasi oleh Validator 1

SURAT PERNYATAAN HASIL VALIDASI DATA

Yang bertanda tangan di bawah ini:

Nama : I GUSTI AYU PUTU SRI DARMAWATI, M.PD

NIP/NIK : 197202251992032009

Jabatan : Guru Bahasa Indonesia Kelas 6

Instansi : SD Negeri 1 Astina

Dengan ini menyatakan bahwa saya telah melakukan validasi terhadap data komentar TikTok mengenai Program Petani Milenial yang telah diberikan oleh:

Nama Mahasiswa : Aprilia Monica Sari

NIM : 2115091069

Program Studi : Sistem Informasi

Universitas : Universitas Pendidikan Ganesha

Data tersebut terdiri dari sejumlah kalimat komentar yang telah diberi label sentimen positif dan negatif oleh mahasiswa. Berdasarkan hasil penelaahan terhadap konteks kalimat, diksi, serta makna yang terkandung di dalam komentar, saya menyatakan bahwa pelabelan yang dilakukan sudah sesuai dengan kaidah Bahasa Indonesia dan dapat diterima secara makna dan konteks dalam analisis sentimen.

Demikian surat ini dibuat untuk digunakan sebagaimana mestinya.

Singaraja, 25 Agustus 2025
Guru Bahasa Indonesia,

I GUSTI AYU PUTU SRI DARMAWATI, M.PD
NIP. 197202251992032009

Lampiran 5. Surat Pernyataan Hasil Validasi oleh Validator 2

SURAT PERNYATAAN HASIL VALIDASI DATA

Yang bertanda tangan di bawah ini:

Nama : Dewa Ayu Wijayanti Kusuma Dewi, S.Pd

NIP/NIK : 199504162022212002

Jabatan : Guru Bahasa Indonesia

Instansi : SMP Negeri 1 Singaraja

Dengan ini menyatakan bahwa saya telah melakukan validasi terhadap data komentar TikTok mengenai Program Petani Milenial yang telah diberikan oleh:

Nama Mahasiswa : Aprilia Monica Sari

NIM : 2115091069

Program Studi : Sistem Informasi

Universitas : Universitas Pendidikan Ganesha

Data tersebut terdiri dari sejumlah kalimat komentar yang telah diberi label sentimen positif dan negatif oleh mahasiswa. Berdasarkan hasil penelaahan terhadap konteks kalimat, diksi, serta makna yang terkandung di dalam komentar, saya menyatakan bahwa pelabelan yang dilakukan sudah sesuai dengan kaidah Bahasa Indonesia dan dapat diterima secara makna dan konteks dalam analisis sentimen.

Demikian surat ini dibuat untuk digunakan sebagaimana mestinya.

Singaraja, 25 Agustus 2025
Guru Bahasa Indonesia,

Dewa Ayu Wijayanti Kusuma Dewi, S.Pd
NIP. 197202251992032009

Lampiran 6. Bukti Foto Bersama Validator Ahli Bahasa

Lampiran 7. *Code* Cleaning

```
# === Tahap Cleaning ===
def cleaning(text):
    text = str(text)
    text = re.sub(r"http\S+|www.\S+", "", text)  # hapus URL
    text = re.sub(r"@\w+", "", text)             # hapus mention
    text = re.sub(r"#\w+", "", text)             # hapus hashtag
    text = re.sub(r"[^0-9a-zA-Z\s]", " ", text)  # hapus simbol & tanda
baca, simpan huruf+angka
    text = re.sub(r"\s+", " ", text).strip()     # hapus spasi berlebih
    return text

df["cleaned"] = df["text"].apply(cleaning)

print("=== Setelah Cleaning ===")
print(df[["uniqueId", "text", "cleaned", "label"]].head())

# === Simpan hasil cleaning dengan kolom uniqueId, text, cleaned, label
===
df_cleaning = df[["uniqueId", "text", "cleaned", "label"]]
df_cleaning.to_excel("/content/1-Data Cleaning.xlsx", index=False)
print("File berhasil disimpan sebagai cleaning.xlsx")
```

Lampiran 8. *Code* Case Folding

```
import pandas as pd

# === Baca file cleaning ===
df_cleaning = pd.read_excel("/content/1-Data Cleaning.xlsx")

# === Casefolding ===
df_cleaning["casefolding"] = df_cleaning["cleaned"].str.lower()

# === Buat DataFrame final ===
df_casefolding = df_cleaning[["uniqueId", "cleaned", "casefolding",
"label"]]

# Rename kolom 'cleaned' jadi 'text'
df_casefolding = df_casefolding.rename(columns={"cleaned": "text"})

print("=== Setelah Casefolding ===")
print(df_casefolding.head())

# === Simpan ===
df_casefolding.to_excel("/content/2-Data Casefolding.xlsx",
index=False)
print("File berhasil disimpan sebagai casefolding.xlsx")
```

Lampiran 9. *Code* Normalize

```
import pandas as pd

# === Baca file casefolding ===
df_casefolding = pd.read_excel("/content/2-Data Casefolding.xlsx")

# === Kamus normalisasi ===
normalization_dict = {
    "trs": "terus",
    "pak": "bapak",
    "udah": "sudah",
    "gak": "tidak",
    "kayak": "seperti",
    "ngga": "tidak",
    "nggak": "tidak",
    "ga": "tidak",
    "g": "tidak",
    "bs": "bisa",
    "aamiin": "amin",
    "aja": "saja",
    "nyampe": "sampai",
    "gini": "ini",
    "yg": "yang",
    "dgn": "dengan",
    "sampe": "sampai",
    "emang": "memang",
    "udh": "sudah",
    "jgn": "jangan",
    "info": "informasi",
    "sdm": "sumber daya manusia",
    "ak": "aku",
    "php": "memberi harapan palsu",
    "beresin": "selesaikan",
    "kementan": "kementrian pertanian",
    "liat": "lihat",
    "joss": "mantap",
    "gass": "mantap",
    "wow": "mantap",
    "pengen": "ingin",
    "infonya": "informasinya",
    "rame": "ramai",
    "ngerasa": "merasa",
    "mantul": "mantap",
    "sdh": "sudah",
    "tidk": "tidak",
    "ngerasain": "merasakan",
    "ngerti" : "mengerti",
    "tetep": "tetap"
}

# === Fungsi normalisasi ===
def normalize_text(text):
    words = str(text).split()
    normalized_words = [normalization_dict.get(w, w) for w in words]
    return " ".join(normalized_words)

# 🚩 Ambil hasil casefolding sebagai sumber teks
df_casefolding["text_casefolding"] = df_casefolding["casefolding"]

# === Terapkan Normalisasi ===
df_casefolding["normalize"] =
df_casefolding["text_casefolding"].apply(normalize_text)

# === Buat DataFrame final ===
```

```
df_normalized = df_casefolding[["uniqueId", "text_casefolding",
"normalize", "label"]]

print("=== Setelah Normalisasi ===")
print(df_normalized.head())

# === Simpan ===
df_normalized.to_excel("/content/3-Data Normalize.xlsx", index=False)
print("File berhasil disimpan sebagai normalize.xlsx")
```

Lampiran 10. *Code* Skenario 1

```
# ============================================================
# 1️⃣ Import Library
# ============================================================
import torch
import numpy as np
import pandas as pd
from torch.utils.data import DataLoader, TensorDataset, RandomSampler,
SequentialSampler
from sklearn.model_selection import KFold
from sklearn.metrics import classification_report, confusion_matrix,
precision_score, recall_score, f1_score
from transformers import BertTokenizer, BertForSequenceClassification
from torch.optim import AdamW
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm

# ============================================================
# 2️⃣ Load Dataset
# ============================================================
df = pd.read_excel("/content/3-Data Normalize.xlsx")

TEXT_COLUMN = "normalize"
LABEL_COLUMN = "label"

# mapping label → angka
label_map = {label: idx for idx, label in
enumerate(df[LABEL_COLUMN].unique())}
df["label_id"] = df[LABEL_COLUMN].map(label_map)
print("Label mapping:", label_map)

texts = df[TEXT_COLUMN].astype(str).tolist()
labels = df["label_id"].astype(int).tolist()

tokenizer = BertTokenizer.from_pretrained("indobenchmark/indobert-
base-p1")
encodings = tokenizer(
    texts, truncation=True, padding=True, max_length=128,
return_tensors="pt"
)

input_ids = encodings["input_ids"]
attention_mask = encodings["attention_mask"]
labels = torch.tensor(labels)
dataset = TensorDataset(input_ids, attention_mask, labels)

# ============================================================
# 3️⃣ Parameter Training
# ============================================================
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
num_epochs = 100
```

```
num_folds = 5
batch_size = 16
lr = 1e-5
patience = 10

# ============================================================
# 4️⃣ Early Stopping Class
# ============================================================
class EarlyStopping:
    def __init__(self, patience=10, min_delta=0, verbose=True):
        """
        Args:
            patience (int): Berapa epoch menunggu setelah tidak ada
peningkatan
            min_delta (float): Minimal perubahan untuk dianggap
sebagai peningkatan
            verbose (bool): Print message saat early stop
        """
        self.patience = patience
        self.min_delta = min_delta
        self.verbose = verbose
        self.counter = 0
        self.best_loss = None
        self.early_stop = False
        self.best_epoch = 0

    def __call__(self, val_loss, epoch):
        if self.best_loss is None:
            self.best_loss = val_loss
            self.best_epoch = epoch
        elif val_loss > self.best_loss - self.min_delta:
            self.counter += 1
            if self.verbose:
                print(f"EarlyStopping counter:
{self.counter}/{self.patience}")
            if self.counter >= self.patience:
                self.early_stop = True
        else:
            self.best_loss = val_loss
            self.best_epoch = epoch
            self.counter = 0

# ============================================================
# 5️⃣ K-Fold Cross Validation dengan Early Stopping
# ============================================================
kfold = KFold(n_splits=num_folds, shuffle=True, random_state=42)

fold_train_acc, fold_val_acc = [], []
fold_prec, fold_rec, fold_f1 = [], [], []
all_preds, all_labels = [], []

train_acc_per_epoch, val_acc_per_epoch = [], []
train_loss_per_epoch, val_loss_per_epoch = [], []
epochs_completed_per_fold = []
best_epoch_per_fold = []

for fold, (train_idx, val_idx) in enumerate(kfold.split(dataset)):
    print(f"\n========== Fold {fold+1} ==========")

    train_subset = torch.utils.data.Subset(dataset, train_idx)
    val_subset = torch.utils.data.Subset(dataset, val_idx)

    train_loader = DataLoader(train_subset,
sampler=RandomSampler(train_subset), batch_size=batch_size)
    val_loader = DataLoader(val_subset,
```

```
sampler=SequentialSampler(val_subset), batch_size=batch_size)

    model = BertForSequenceClassification.from_pretrained(
        "indobenchmark/indobert-base-p1",
        num_labels=len(label_map)
    ).to(device)

    optimizer = AdamW(model.parameters(), lr=lr)

    early_stopping = EarlyStopping(patience=patience, verbose=True)

    fold_train_acc_epoch, fold_val_acc_epoch = [], []
    fold_train_loss_epoch, fold_val_loss_epoch = [], []

    for epoch in range(num_epochs):
        # ---- Training ----
        model.train()
        total_loss, correct, total = 0, 0, 0
        for batch in tqdm(train_loader, desc=f"Training Epoch
{epoch+1}"):
            optimizer.zero_grad()
            input_ids, attention_mask, labels_batch = [b.to(device)
for b in batch]
            outputs = model(input_ids, attention_mask=attention_mask,
labels=labels_batch)
            loss = outputs.loss
            logits = outputs.logits
            loss.backward()
            optimizer.step()

            total_loss += loss.item()
            preds = torch.argmax(logits, dim=1)
            correct += (preds == labels_batch).sum().item()
            total += labels_batch.size(0)

        train_acc = correct / total
        avg_train_loss = total_loss / len(train_loader)

        # ---- Validation ----
        model.eval()
        val_loss, val_correct, val_total = 0, 0, 0
        val_preds, val_true = [], []
        with torch.no_grad():
            for batch in val_loader:
                input_ids, attention_mask, labels_batch =
[b.to(device) for b in batch]
                outputs = model(input_ids,
attention_mask=attention_mask, labels=labels_batch)
                val_loss += outputs.loss.item()
                preds = torch.argmax(outputs.logits, dim=1)
                val_correct += (preds == labels_batch).sum().item()
                val_total += labels_batch.size(0)
                val_preds.extend(preds.cpu().numpy())
                val_true.extend(labels_batch.cpu().numpy())

        val_acc = val_correct / val_total
        avg_val_loss = val_loss / len(val_loader)

        fold_train_acc_epoch.append(train_acc)
        fold_val_acc_epoch.append(val_acc)
        fold_train_loss_epoch.append(avg_train_loss)
        fold_val_loss_epoch.append(avg_val_loss)

        print(f"Epoch {epoch+1}/{num_epochs} - Train Loss:
{avg_train_loss:.4f}, Val Loss: {avg_val_loss:.4f}, "
```

```
                f"Train Acc: {train_acc:.4f}, Val Acc: {val_acc:.4f}")

        early_stopping(avg_val_loss, epoch+1)

        if early_stopping.early_stop:
            print(f"⚠️ Early stopping triggered at epoch {epoch+1}")
            print(f"✅ Best validation loss was
{early_stopping.best_loss:.4f} at epoch {early_stopping.best_epoch}")
            break

    epochs_completed = len(fold_train_acc_epoch)
    epochs_completed_per_fold.append(epochs_completed)
    best_epoch_per_fold.append(early_stopping.best_epoch)
    print(f"📊 Fold {fold+1} completed {epochs_completed} epochs
(best at epoch {early_stopping.best_epoch})")

    # Metrics per fold
    precision = precision_score(val_true, val_preds,
average="weighted", zero_division=0)
    recall = recall_score(val_true, val_preds, average="weighted",
zero_division=0)
    f1 = f1_score(val_true, val_preds, average="weighted",
zero_division=0)

    fold_train_acc.append(train_acc)
    fold_val_acc.append(val_acc)
    fold_prec.append(precision)
    fold_rec.append(recall)
    fold_f1.append(f1)

    all_preds.extend(val_preds)
    all_labels.extend(val_true)

    train_acc_per_epoch.append(fold_train_acc_epoch)
    val_acc_per_epoch.append(fold_val_acc_epoch)
    train_loss_per_epoch.append(fold_train_loss_epoch)
    val_loss_per_epoch.append(fold_val_loss_epoch)

    print(f"Fold {fold+1} - Train Acc: {train_acc:.4f} | Val Acc:
{val_acc:.4f} | "
          f"Prec: {precision:.4f} | Rec: {recall:.4f} | F1: {f1:.4f}")

# ============================================================
# 6️⃣ Informasi Early Stopping
# ============================================================
print("\n===== Informasi Early Stopping =====")
print(f"Epoch minimum: {min(epochs_completed_per_fold)}")
print(f"Epoch maksimum: {max(epochs_completed_per_fold)}")
print(f"Rata-rata epoch: {np.mean(epochs_completed_per_fold):.2f}")
print(f"Epoch per fold: {epochs_completed_per_fold}")

print(f"\nBest epoch per fold: {best_epoch_per_fold}")
print(f"Rata-rata best epoch: {np.mean(best_epoch_per_fold):.2f}")
print(f"Best epoch minimum: {min(best_epoch_per_fold)}")
print(f"Best epoch maksimum: {max(best_epoch_per_fold)}")

# ============================================================
# 7️⃣ Rata-rata Akhir 5 Fold
# ============================================================
print("\n===== Rata-Rata Hasil 5 Fold =====")
print(f"Train Acc: {np.mean(fold_train_acc):.4f}")
print(f"Val Acc:   {np.mean(fold_val_acc):.4f}")
print(f"Precision: {np.mean(fold_prec):.4f}")
print(f"Recall:    {np.mean(fold_rec):.4f}")
```

```
print(f"F1 Score:  {np.mean(fold_f1):.4f}")

# ============================================================
# 8️⃣ Grafik Akurasi & Loss Rata-rata per Epoch
# ============================================================
min_epochs = min(epochs_completed_per_fold)

train_acc_trimmed = [fold[:min_epochs] for fold in
train_acc_per_epoch]
val_acc_trimmed = [fold[:min_epochs] for fold in val_acc_per_epoch]
train_loss_trimmed = [fold[:min_epochs] for fold in
train_loss_per_epoch]
val_loss_trimmed = [fold[:min_epochs] for fold in val_loss_per_epoch]

avg_train_acc = np.mean(train_acc_trimmed, axis=0)
avg_val_acc = np.mean(val_acc_trimmed, axis=0)
avg_train_loss = np.mean(train_loss_trimmed, axis=0)
avg_val_loss = np.mean(val_loss_trimmed, axis=0)

epochs_range = range(1, min_epochs + 1)

# ============================================================
# 9️⃣ Confusion Matrix & Classification Report
# ============================================================
cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
xticklabels=label_map.keys(), yticklabels=label_map.keys())
plt.xlabel("Predicted", fontsize=12)
plt.ylabel("True", fontsize=12)
plt.title("Confusion Matrix (Keseluruhan)", fontsize=14)
plt.tight_layout()
plt.savefig("confusion_matrix_overall.png", dpi=300)
plt.show()

target_names = list(label_map.keys())

# ✅ Custom classification report dengan nilai bulat (persentase)
report_dict = classification_report(all_labels, all_preds,
target_names=target_names, output_dict=True, zero_division=0)

print("\n===== 📋 Classification Report Final =====")
print(f"{'Class':<15} {'Precision':<12} {'Recall':<12} {'F1-
Score':<12} {'Support':<10}")
print("=" * 65)

# Print per kelas
for label_name in target_names:
    prec = report_dict[label_name]['precision'] * 100
    rec = report_dict[label_name]['recall'] * 100
    f1 = report_dict[label_name]['f1-score'] * 100
    sup = int(report_dict[label_name]['support'])
    print(f"{label_name:<15}
{prec:>6.0f}%       {rec:>6.0f}%       {f1:>6.0f}%       {sup:>6}")

print("-" * 65)

# Print accuracy
acc = report_dict['accuracy'] * 100
total_support = int(report_dict['weighted avg']['support'])
print(f"{'Accuracy':<15} {'':<12} {'':<12}
{acc:>6.0f}%       {total_support:>6}")

print("-" * 65)
```

```
# Print macro avg
macro_prec = report_dict['macro avg']['precision'] * 100
macro_rec = report_dict['macro avg']['recall'] * 100
macro_f1 = report_dict['macro avg']['f1-score'] * 100
print(f"{'Macro Avg':<15}
{macro_prec:>6.0f}%       {macro_rec:>6.0f}%       {macro_f1:>6.0f}%
    {total_support:>6}")

# Print weighted avg
weighted_prec = report_dict['weighted avg']['precision'] * 100
weighted_rec = report_dict['weighted avg']['recall'] * 100
weighted_f1 = report_dict['weighted avg']['f1-score'] * 100
print(f"{'Weighted Avg':<15}
{weighted_prec:>6.0f}%       {weighted_rec:>6.0f}%       {weighted_f1:
>6.0f}%       {total_support:>6}")

# Simpan ke CSV
report_df = pd.DataFrame(report_dict).transpose()
report_df_pct = report_df.copy()
for col in ['precision', 'recall', 'f1-score']:
    if col in report_df_pct.columns:
        report_df_pct[col] = (report_df_pct[col] *
100).round(0).astype(int).astype(str) + '%'
report_df_pct.to_csv("classification_report_overall.csv", index=True)

# ✅ Grafik Akurasi (SMOOTH - tanpa marker)
plt.figure(figsize=(10,5))
plt.plot(epochs_range, avg_train_acc, label="Train Accuracy",
linewidth=2.5, color='#1f77b4')
plt.plot(epochs_range, avg_val_acc, label="Validation Accuracy",
linewidth=2.5, color='#ff7f0e')
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Accuracy", fontsize=12)
plt.title(f"Rata-rata Akurasi per Epoch (5 Fold, Early Stop)",
fontsize=14)
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig("avg_accuracy_epoch.png", dpi=300)
plt.show()

# ✅ Grafik Loss (SMOOTH - tanpa marker)
plt.figure(figsize=(10,5))
plt.plot(epochs_range, avg_train_loss, label="Train Loss",
linewidth=2.5, color='#1f77b4')
plt.plot(epochs_range, avg_val_loss, label="Validation Loss",
linewidth=2.5, color='#ff7f0e')
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Loss", fontsize=12)
plt.title(f"Rata-rata Loss per Epoch (5 Fold, Early Stop)",
fontsize=14)
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig("avg_loss_epoch.png", dpi=300)
plt.show()

print(f"\n✅ Panjang sumbu X pada grafik: {min_epochs} epoch")
print(f"✅ Semua hasil (grafik & laporan) sudah tersimpan dan tampil
di output Colab.")
```

Lampiran 11. *Code* Skenario 2

```
# ============================================================
# 1 Import Library
# ============================================================
import torch
import numpy as np
import pandas as pd
from torch.utils.data import DataLoader, TensorDataset, RandomSampler,
SequentialSampler
from sklearn.model_selection import KFold
from sklearn.metrics import classification_report, confusion_matrix,
precision_score, recall_score, f1_score
from transformers import BertTokenizer, BertForSequenceClassification
from torch.optim import AdamW
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm

# ============================================================
# 2 Load Dataset
# ============================================================
df = pd.read_excel("/content/3-Data Normalize.xlsx")

TEXT_COLUMN = "normalize"
LABEL_COLUMN = "label"

# mapping label → angka
label_map = {label: idx for idx, label in
enumerate(df[LABEL_COLUMN].unique())}
df["label_id"] = df[LABEL_COLUMN].map(label_map)
print("Label mapping:", label_map)

texts = df[TEXT_COLUMN].astype(str).tolist()
labels = df["label_id"].astype(int).tolist()

tokenizer = BertTokenizer.from_pretrained("indobenchmark/indobert-
base-p1")
encodings = tokenizer(
    texts, truncation=True, padding=True, max_length=128,
return_tensors="pt"
)

input_ids = encodings["input_ids"]
attention_mask = encodings["attention_mask"]
labels = torch.tensor(labels)
dataset = TensorDataset(input_ids, attention_mask, labels)

# ============================================================
# 3 Parameter Training
# ============================================================
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
num_epochs = 100
num_folds = 5
batch_size = 32
lr = 1e-5
patience = 10

# ============================================================
# 4 Early Stopping Class
# ============================================================
class EarlyStopping:
    def __init__(self, patience=10, min_delta=0, verbose=True):
        """
        Args:
            patience (int): Berapa epoch menunggu setelah tidak ada
```

```
peningkatan
            min_delta (float): Minimal perubahan untuk dianggap
sebagai peningkatan
            verbose (bool): Print message saat early stop
        """
        self.patience = patience
        self.min_delta = min_delta
        self.verbose = verbose
        self.counter = 0
        self.best_loss = None
        self.early_stop = False
        self.best_epoch = 0

    def __call__(self, val_loss, epoch):
        if self.best_loss is None:
            self.best_loss = val_loss
            self.best_epoch = epoch
        elif val_loss > self.best_loss - self.min_delta:
            self.counter += 1
            if self.verbose:
                print(f"EarlyStopping counter:
{self.counter}/{self.patience}")
            if self.counter >= self.patience:
                self.early_stop = True
        else:
            self.best_loss = val_loss
            self.best_epoch = epoch
            self.counter = 0

# ========================================================
# 5️⃣ K-Fold Cross Validation dengan Early Stopping
# ========================================================
kfold = KFold(n_splits=num_folds, shuffle=True, random_state=42)

fold_train_acc, fold_val_acc = [], []
fold_prec, fold_rec, fold_f1 = [], [], []
all_preds, all_labels = [], []

train_acc_per_epoch, val_acc_per_epoch = [], []
train_loss_per_epoch, val_loss_per_epoch = [], []
epochs_completed_per_fold = []
best_epoch_per_fold = []

for fold, (train_idx, val_idx) in enumerate(kfold.split(dataset)):
    print(f"\n========== Fold {fold+1} ==========")

    train_subset = torch.utils.data.Subset(dataset, train_idx)
    val_subset = torch.utils.data.Subset(dataset, val_idx)

    train_loader = DataLoader(train_subset,
sampler=RandomSampler(train_subset), batch_size=batch_size)
    val_loader = DataLoader(val_subset,
sampler=SequentialSampler(val_subset), batch_size=batch_size)

    model = BertForSequenceClassification.from_pretrained(
        "indobenchmark/indobert-base-p1",
        num_labels=len(label_map)
    ).to(device)

    optimizer = AdamW(model.parameters(), lr=lr)

    early_stopping = EarlyStopping(patience=patience, verbose=True)

    fold_train_acc_epoch, fold_val_acc_epoch = [], []
    fold_train_loss_epoch, fold_val_loss_epoch = [], []
```

```
    for epoch in range(num_epochs):
        # ---- Training ----
        model.train()
        total_loss, correct, total = 0, 0, 0
        for batch in tqdm(train_loader, desc=f"Training Epoch
{epoch+1}"):
            optimizer.zero_grad()
            input_ids, attention_mask, labels_batch = [b.to(device)
for b in batch]
            outputs = model(input_ids, attention_mask=attention_mask,
labels=labels_batch)
            loss = outputs.loss
            logits = outputs.logits
            loss.backward()
            optimizer.step()

            total_loss += loss.item()
            preds = torch.argmax(logits, dim=1)
            correct += (preds == labels_batch).sum().item()
            total += labels_batch.size(0)

        train_acc = correct / total
        avg_train_loss = total_loss / len(train_loader)

        # ---- Validation ----
        model.eval()
        val_loss, val_correct, val_total = 0, 0, 0
        val_preds, val_true = [], []
        with torch.no_grad():
            for batch in val_loader:
                input_ids, attention_mask, labels_batch =
[b.to(device) for b in batch]
                outputs = model(input_ids,
attention_mask=attention_mask, labels=labels_batch)
                val_loss += outputs.loss.item()
                preds = torch.argmax(outputs.logits, dim=1)
                val_correct += (preds == labels_batch).sum().item()
                val_total += labels_batch.size(0)
                val_preds.extend(preds.cpu().numpy())
                val_true.extend(labels_batch.cpu().numpy())

        val_acc = val_correct / val_total
        avg_val_loss = val_loss / len(val_loader)

        fold_train_acc_epoch.append(train_acc)
        fold_val_acc_epoch.append(val_acc)
        fold_train_loss_epoch.append(avg_train_loss)
        fold_val_loss_epoch.append(avg_val_loss)

        print(f"Epoch {epoch+1}/{num_epochs} - Train Loss:
{avg_train_loss:.4f}, Val Loss: {avg_val_loss:.4f}, "
              f"Train Acc: {train_acc:.4f}, Val Acc: {val_acc:.4f}")

        early_stopping(avg_val_loss, epoch+1)

        if early_stopping.early_stop:
            print(f"⚠️ Early stopping triggered at epoch {epoch+1}")
            print(f"✅ Best validation loss was
{early_stopping.best_loss:.4f} at epoch {early_stopping.best_epoch}")
            break

    epochs_completed = len(fold_train_acc_epoch)
    epochs_completed_per_fold.append(epochs_completed)
    best_epoch_per_fold.append(early_stopping.best_epoch)
```

```
    print(f"📊 Fold {fold+1} completed {epochs_completed} epochs
(best at epoch {early_stopping.best_epoch})")

    # Metrics per fold
    precision = precision_score(val_true, val_preds,
average="weighted", zero_division=0)
    recall = recall_score(val_true, val_preds, average="weighted",
zero_division=0)
    f1 = f1_score(val_true, val_preds, average="weighted",
zero_division=0)

    fold_train_acc.append(train_acc)
    fold_val_acc.append(val_acc)
    fold_prec.append(precision)
    fold_rec.append(recall)
    fold_f1.append(f1)

    all_preds.extend(val_preds)
    all_labels.extend(val_true)

    train_acc_per_epoch.append(fold_train_acc_epoch)
    val_acc_per_epoch.append(fold_val_acc_epoch)
    train_loss_per_epoch.append(fold_train_loss_epoch)
    val_loss_per_epoch.append(fold_val_loss_epoch)

    print(f"Fold {fold+1} - Train Acc: {train_acc:.4f} | Val Acc:
{val_acc:.4f} | "
          f"Prec: {precision:.4f} | Rec: {recall:.4f} | F1: {f1:.4f}")

# ============================================================
# 6️⃣ Informasi Early Stopping
# ============================================================
print("\n===== Informasi Early Stopping =====")
print(f"Epoch minimum: {min(epochs_completed_per_fold)}")
print(f"Epoch maksimum: {max(epochs_completed_per_fold)}")
print(f"Rata-rata epoch: {np.mean(epochs_completed_per_fold):.2f}")
print(f"Epoch per fold: {epochs_completed_per_fold}")

print(f"\nBest epoch per fold: {best_epoch_per_fold}")
print(f"Rata-rata best epoch: {np.mean(best_epoch_per_fold):.2f}")
print(f"Best epoch minimum: {min(best_epoch_per_fold)}")
print(f"Best epoch maksimum: {max(best_epoch_per_fold)}")

# ============================================================
# 7️⃣ Rata-rata Akhir 5 Fold
# ============================================================
print("\n===== Rata-Rata Hasil 5 Fold =====")
print(f"Train Acc: {np.mean(fold_train_acc):.4f}")
print(f"Val Acc:   {np.mean(fold_val_acc):.4f}")
print(f"Precision: {np.mean(fold_prec):.4f}")
print(f"Recall:    {np.mean(fold_rec):.4f}")
print(f"F1 Score:  {np.mean(fold_f1):.4f}")

# ============================================================
# 8️⃣ Grafik Akurasi & Loss Rata-rata per Epoch
# ============================================================
min_epochs = min(epochs_completed_per_fold)

train_acc_trimmed = [fold[:min_epochs] for fold in
train_acc_per_epoch]
val_acc_trimmed = [fold[:min_epochs] for fold in val_acc_per_epoch]
train_loss_trimmed = [fold[:min_epochs] for fold in
train_loss_per_epoch]
val_loss_trimmed = [fold[:min_epochs] for fold in val_loss_per_epoch]
```

```
avg_train_acc = np.mean(train_acc_trimmed, axis=0)
avg_val_acc = np.mean(val_acc_trimmed, axis=0)
avg_train_loss = np.mean(train_loss_trimmed, axis=0)
avg_val_loss = np.mean(val_loss_trimmed, axis=0)

epochs_range = range(1, min_epochs + 1)

# ============================================================
# 📍 Confusion Matrix & Classification Report
# ============================================================
cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
xticklabels=label_map.keys(), yticklabels=label_map.keys())
plt.xlabel("Predicted", fontsize=12)
plt.ylabel("True", fontsize=12)
plt.title("Confusion Matrix (Keseluruhan)", fontsize=14)
plt.tight_layout()
plt.savefig("confusion_matrix_overall.png", dpi=300)
plt.show()

target_names = list(label_map.keys())

# ✅ Custom classification report dengan nilai bulat (persentase)
report_dict = classification_report(all_labels, all_preds,
target_names=target_names, output_dict=True, zero_division=0)

print("\n===== 📋 Classification Report Final =====")
print(f"{'Class':<15} {'Precision':<12} {'Recall':<12} {'F1-
Score':<12} {'Support':<10}")
print("=" * 65)

# Print per kelas
for label_name in target_names:
    prec = report_dict[label_name]['precision'] * 100
    rec = report_dict[label_name]['recall'] * 100
    f1 = report_dict[label_name]['f1-score'] * 100
    sup = int(report_dict[label_name]['support'])
    print(f"{label_name:<15}
{prec:>6.0f}%       {rec:>6.0f}%       {f1:>6.0f}%       {sup:>6}")

print("-" * 65)

# Print accuracy
acc = report_dict['accuracy'] * 100
total_support = int(report_dict['weighted avg']['support'])
print(f"{'Accuracy':<15} {'':<12} {'':<12}
{acc:>6.0f}%       {total_support:>6}")

print("-" * 65)

# Print macro avg
macro_prec = report_dict['macro avg']['precision'] * 100
macro_rec = report_dict['macro avg']['recall'] * 100
macro_f1 = report_dict['macro avg']['f1-score'] * 100
print(f"{'Macro Avg':<15}
{macro_prec:>6.0f}%       {macro_rec:>6.0f}%       {macro_f1:>6.0f}%
    {total_support:>6}")

# Print weighted avg
weighted_prec = report_dict['weighted avg']['precision'] * 100
weighted_rec = report_dict['weighted avg']['recall'] * 100
weighted_f1 = report_dict['weighted avg']['f1-score'] * 100
print(f"{'Weighted Avg':<15}
```

```
{weighted_prec:>6.0f}%       {weighted_rec:>6.0f}%       {weighted_f1:
>6.0f}%       {total_support:>6}")

# Simpan ke CSV
report_df = pd.DataFrame(report_dict).transpose()
report_df_pct = report_df.copy()
for col in ['precision', 'recall', 'f1-score']:
    if col in report_df_pct.columns:
        report_df_pct[col] = (report_df_pct[col] *
100).round(0).astype(int).astype(str) + '%'
report_df_pct.to_csv("classification_report_overall.csv", index=True)

# ✅ Grafik Akurasi (SMOOTH - tanpa marker)
plt.figure(figsize=(10,5))
plt.plot(epochs_range, avg_train_acc, label="Train Accuracy",
linewidth=2.5, color='#1f77b4')
plt.plot(epochs_range, avg_val_acc, label="Validation Accuracy",
linewidth=2.5, color='#ff7f0e')
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Accuracy", fontsize=12)
plt.title(f"Rata-rata Akurasi per Epoch (5 Fold, Early Stop)",
fontsize=14)
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig("avg_accuracy_epoch.png", dpi=300)
plt.show()

# ✅ Grafik Loss (SMOOTH - tanpa marker)
plt.figure(figsize=(10,5))
plt.plot(epochs_range, avg_train_loss, label="Train Loss",
linewidth=2.5, color='#1f77b4')
plt.plot(epochs_range, avg_val_loss, label="Validation Loss",
linewidth=2.5, color='#ff7f0e')
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Loss", fontsize=12)
plt.title(f"Rata-rata Loss per Epoch (5 Fold, Early Stop)",
fontsize=14)
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig("avg_loss_epoch.png", dpi=300)
plt.show()

print(f"\n✅ Panjang sumbu X pada grafik: {min_epochs} epoch")
print(f"✅ Semua hasil (grafik & laporan) sudah tersimpan dan tampil
di output Colab.")
```

Lampiran 12. *Code* Skenario 3

```
# ============================================================
# 1️⃣ Import Library
# ============================================================
import torch
import numpy as np
import pandas as pd
from torch.utils.data import DataLoader, TensorDataset, RandomSampler,
SequentialSampler
from sklearn.model_selection import KFold
from sklearn.metrics import classification_report, confusion_matrix,
precision_score, recall_score, f1_score
from transformers import BertTokenizer, BertForSequenceClassification
from torch.optim import AdamW
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
from imblearn.over_sampling import SMOTE  # ✅ GANTI: Import SMOTE
from collections import Counter

# ============================================================
# 2️⃣ Load Dataset
# ============================================================
df = pd.read_excel("/content/3-Data Normalize.xlsx")

TEXT_COLUMN = "normalize"
LABEL_COLUMN = "label"

# mapping label → angka
label_map = {label: idx for idx, label in
enumerate(df[LABEL_COLUMN].unique())}
df["label_id"] = df[LABEL_COLUMN].map(label_map)
print("Label mapping:", label_map)

texts = df[TEXT_COLUMN].astype(str).tolist()
labels = df["label_id"].astype(int).tolist()

# ✅ Tampilkan distribusi kelas asli
print("\n===== Distribusi Kelas Original =====")
label_counts = Counter(labels)
for label, count in label_counts.items():
    label_name = [k for k, v in label_map.items() if v == label][0]
    print(f"{label_name}: {count} ({count/len(labels)*100:.2f}%)")

tokenizer = BertTokenizer.from_pretrained("indobenchmark/indobert-
base-p1")
encodings = tokenizer(
    texts, truncation=True, padding=True, max_length=128,
return_tensors="pt"
)

input_ids = encodings["input_ids"]
attention_mask = encodings["attention_mask"]
labels_tensor = torch.tensor(labels)

# ============================================================
# 3️⃣ Parameter Training
# ============================================================
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
num_epochs = 100
num_folds = 5
batch_size = 16
lr = 1e-5
```

```
patience = 10
use_smote = True

# ============================================================
# 4️⃣ Early Stopping Class
# ============================================================
class EarlyStopping:
    def __init__(self, patience=10, min_delta=0, verbose=True):
        self.patience = patience
        self.min_delta = min_delta
        self.verbose = verbose
        self.counter = 0
        self.best_loss = None
        self.early_stop = False
        self.best_epoch = 0

    def __call__(self, val_loss, epoch):
        if self.best_loss is None:
            self.best_loss = val_loss
            self.best_epoch = epoch
        elif val_loss > self.best_loss - self.min_delta:
            self.counter += 1
            if self.verbose:
                print(f"EarlyStopping counter:
{self.counter}/{self.patience}")
            if self.counter >= self.patience:
                self.early_stop = True
        else:
            self.best_loss = val_loss
            self.best_epoch = epoch
            self.counter = 0

# ============================================================
# 5️⃣ K-Fold Cross Validation dengan Early Stopping + SMOTE
# ============================================================
kfold = KFold(n_splits=num_folds, shuffle=True, random_state=42)

fold_train_acc, fold_val_acc = [], []
fold_prec, fold_rec, fold_f1 = [], [], []
all_preds, all_labels = [], []

train_acc_per_epoch, val_acc_per_epoch = [], []
train_loss_per_epoch, val_loss_per_epoch = [], []
epochs_completed_per_fold = []
best_epoch_per_fold = []

# ✅ TAMBAHAN: Tracking info SMOTE per fold
smote_info_per_fold = []

for fold, (train_idx, val_idx) in
enumerate(kfold.split(range(len(labels)))):
    print(f"\n{'='*60}")
    print(f"========== Fold {fold+1} ==========")
    print(f"{'='*60}")

    # ✅ LANGKAH 1: Ambil data train dan validation
    train_input_ids = input_ids[train_idx]
    train_attention_mask = attention_mask[train_idx]
    train_labels = labels_tensor[train_idx]

    val_input_ids = input_ids[val_idx]
    val_attention_mask = attention_mask[val_idx]
    val_labels = labels_tensor[val_idx]
```

```
    print(f"\n--- Distribusi Sebelum SMOTE ---")
    train_label_counts = Counter(train_labels.numpy())
    for label, count in sorted(train_label_counts.items()):
        label_name = [k for k, v in label_map.items() if v ==
label][0]
        print(f"  Train {label_name}: {count}")

    # ✅ LANGKAH 2: Terapkan SMOTE pada Training Set
    if use_smote:
        print(f"\n⚙️ Menerapkan SMOTE pada Training Set...")
        print(f"   SMOTE akan membuat data sintetis untuk kelas
minoritas")

        # Gabungkan input_ids dan attention_mask untuk SMOTE
        original_shape = train_input_ids.shape

        # Reshape untuk SMOTE: (num_samples, features)
        train_data = torch.cat([
            train_input_ids,
            train_attention_mask
        ], dim=1).numpy()

        # Apply SMOTE
        # k_neighbors=5 artinya menggunakan 5 tetangga terdekat untuk
interpolasi
        # Pastikan k_neighbors <= jumlah sampel kelas minoritas - 1
        min_class_count = min(train_label_counts.values())
        k_neighbors = min(5, min_class_count - 1) if min_class_count >
1 else 1

        smote = SMOTE(random_state=42, k_neighbors=k_neighbors)
        train_data_resampled, train_labels_resampled =
smote.fit_resample(
            train_data,
            train_labels.numpy()
        )

        # Split kembali menjadi input_ids dan attention_mask
        seq_length = original_shape[1]
        train_input_ids = torch.tensor(train_data_resampled[:,
:seq_length])
        train_attention_mask = torch.tensor(train_data_resampled[:,
seq_length:])
        train_labels = torch.tensor(train_labels_resampled)

        print(f"\n--- Distribusi Setelah SMOTE ---")
        train_label_counts_after = Counter(train_labels.numpy())
        for label, count in sorted(train_label_counts_after.items()):
            label_name = [k for k, v in label_map.items() if v ==
label][0]
            print(f"  Train {label_name}: {count}")

        print(f"\n📊 Ukuran dataset:")
        print(f"  Training: {len(train_labels)} samples (setelah
SMOTE)")
        print(f"  Validation: {len(val_labels)} samples (TIDAK di-
SMOTE)")

        # Simpan info SMOTE
        before_count = len(train_idx)
        after_count = len(train_labels)
        smote_info = {
            'fold': fold + 1,
            'before': before_count,
            'after': after_count,
```

```
            'addition': after_count - before_count,
            'addition_pct': ((after_count - before_count) /
before_count) * 100
        }
        smote_info_per_fold.append(smote_info)
        print(f"  ✅ Data sintetis bertambah:
{smote_info['addition']} sampel ({smote_info['addition_pct']:.1f}%)")

    # ✅ LANGKAH 3: Buat DataLoader
    train_dataset = TensorDataset(train_input_ids,
train_attention_mask, train_labels)
    val_dataset = TensorDataset(val_input_ids, val_attention_mask,
val_labels)

    train_loader = DataLoader(train_dataset,
sampler=RandomSampler(train_dataset), batch_size=batch_size)
    val_loader = DataLoader(val_dataset,
sampler=SequentialSampler(val_dataset), batch_size=batch_size)

    # ✅ LANGKAH 4: Inisialisasi Model
    model = BertForSequenceClassification.from_pretrained(
        "indobenchmark/indobert-base-p1",
        num_labels=len(label_map)
    ).to(device)

    optimizer = AdamW(model.parameters(), lr=lr)
    early_stopping = EarlyStopping(patience=patience, verbose=True)

    fold_train_acc_epoch, fold_val_acc_epoch = [], []
    fold_train_loss_epoch, fold_val_loss_epoch = [], []

    # ✅ LANGKAH 5: Training Loop dengan Early Stopping
    for epoch in range(num_epochs):
        # ---- Training ----
        model.train()
        total_loss, correct, total = 0, 0, 0
        for batch in tqdm(train_loader, desc=f"Training Epoch
{epoch+1}"):
            optimizer.zero_grad()
            input_ids_batch, attention_mask_batch, labels_batch =
[b.to(device) for b in batch]
            outputs = model(input_ids_batch,
attention_mask=attention_mask_batch, labels=labels_batch)
            loss = outputs.loss
            logits = outputs.logits
            loss.backward()
            optimizer.step()

            total_loss += loss.item()
            preds = torch.argmax(logits, dim=1)
            correct += (preds == labels_batch).sum().item()
            total += labels_batch.size(0)

        train_acc = correct / total
        avg_train_loss = total_loss / len(train_loader)

        # ---- Validation (pada data ASLI, tanpa SMOTE) ----
        model.eval()
        val_loss, val_correct, val_total = 0, 0, 0
        val_preds, val_true = [], []
        with torch.no_grad():
            for batch in val_loader:
                input_ids_batch, attention_mask_batch, labels_batch =
[b.to(device) for b in batch]
```

```
                outputs = model(input_ids_batch,
attention_mask=attention_mask_batch, labels=labels_batch)
                val_loss += outputs.loss.item()
                preds = torch.argmax(outputs.logits, dim=1)
                val_correct += (preds == labels_batch).sum().item()
                val_total += labels_batch.size(0)
                val_preds.extend(preds.cpu().numpy())
                val_true.extend(labels_batch.cpu().numpy())

        val_acc = val_correct / val_total
        avg_val_loss = val_loss / len(val_loader)

        fold_train_acc_epoch.append(train_acc)
        fold_val_acc_epoch.append(val_acc)
        fold_train_loss_epoch.append(avg_train_loss)
        fold_val_loss_epoch.append(avg_val_loss)

        print(f"Epoch {epoch+1}/{num_epochs} - Train Loss:
{avg_train_loss:.4f}, Val Loss: {avg_val_loss:.4f}, "
              f"Train Acc: {train_acc:.4f}, Val Acc: {val_acc:.4f}")

        # ✅ LANGKAH 6: Early Stopping memantau validation loss
        early_stopping(avg_val_loss, epoch+1)

        if early_stopping.early_stop:
            print(f"⚠️ Early stopping triggered at epoch {epoch+1}")
            print(f"✅ Best validation loss was
{early_stopping.best_loss:.4f} at epoch {early_stopping.best_epoch}")
            break

    epochs_completed = len(fold_train_acc_epoch)
    epochs_completed_per_fold.append(epochs_completed)
    best_epoch_per_fold.append(early_stopping.best_epoch)
    print(f"📊 Fold {fold+1} completed {epochs_completed} epochs
(best at epoch {early_stopping.best_epoch})")

    # Metrics per fold
    precision = precision_score(val_true, val_preds,
average="weighted", zero_division=0)
    recall = recall_score(val_true, val_preds, average="weighted",
zero_division=0)
    f1 = f1_score(val_true, val_preds, average="weighted",
zero_division=0)

    fold_train_acc.append(train_acc)
    fold_val_acc.append(val_acc)
    fold_prec.append(precision)
    fold_rec.append(recall)
    fold_f1.append(f1)

    all_preds.extend(val_preds)
    all_labels.extend(val_true)

    train_acc_per_epoch.append(fold_train_acc_epoch)
    val_acc_per_epoch.append(fold_val_acc_epoch)
    train_loss_per_epoch.append(fold_train_loss_epoch)
    val_loss_per_epoch.append(fold_val_loss_epoch)

    print(f"Fold {fold+1} - Train Acc: {train_acc:.4f} | Val Acc:
{val_acc:.4f} | "
          f"Prec: {precision:.4f} | Rec: {recall:.4f} | F1: {f1:.4f}")

# ============================================================
# 6️⃣ Informasi SMOTE (Synthetic Minority Over-sampling Technique)
```

```
# ============================================================
if use_smote:
    print("\n===== 📊 Informasi SMOTE =====")
    smote_df = pd.DataFrame(smote_info_per_fold)
    print(smote_df.to_string(index=False))
    print(f"\nRata-rata penambahan data sintetis:
{smote_df['addition'].mean():.1f} sampel
({smote_df['addition_pct'].mean():.1f}%)")
    print(f"Total data training sebelum SMOTE:
{smote_df['before'].mean():.0f}")
    print(f"Total data training setelah SMOTE:
{smote_df['after'].mean():.0f}")

# ============================================================
# 7️⃣ Informasi Early Stopping
# ============================================================
print("\n===== ⏱️ Informasi Early Stopping =====")
print(f"Epoch minimum: {min(epochs_completed_per_fold)}")
print(f"Epoch maksimum: {max(epochs_completed_per_fold)}")
print(f"Rata-rata epoch: {np.mean(epochs_completed_per_fold):.2f}")
print(f"Epoch per fold: {epochs_completed_per_fold}")
print(f"\nBest epoch per fold: {best_epoch_per_fold}")
print(f"Rata-rata best epoch: {np.mean(best_epoch_per_fold):.2f}")
print(f"Best epoch minimum: {min(best_epoch_per_fold)}")
print(f"Best epoch maksimum: {max(best_epoch_per_fold)}")

# ============================================================
# 8️⃣ Rata-rata Akhir 5 Fold
# ============================================================
print("\n===== 📈 Rata-Rata Hasil 5 Fold =====")
print(f"SMOTE: {'AKTIF' if use_smote else 'TIDAK AKTIF'}")
print(f"Train Acc: {np.mean(fold_train_acc):.4f}")
print(f"Val Acc:   {np.mean(fold_val_acc):.4f}")
print(f"Precision: {np.mean(fold_prec):.4f}")
print(f"Recall:    {np.mean(fold_rec):.4f}")
print(f"F1 Score:  {np.mean(fold_f1):.4f}")

# ============================================================
# 9️⃣ Grafik Akurasi & Loss Rata-rata per Epoch (SMOOTH - TANPA
MARKER)
# ============================================================
min_epochs = min(epochs_completed_per_fold)

train_acc_trimmed = [fold[:min_epochs] for fold in
train_acc_per_epoch]
val_acc_trimmed = [fold[:min_epochs] for fold in val_acc_per_epoch]
train_loss_trimmed = [fold[:min_epochs] for fold in
train_loss_per_epoch]
val_loss_trimmed = [fold[:min_epochs] for fold in val_loss_per_epoch]

avg_train_acc = np.mean(train_acc_trimmed, axis=0)
avg_val_acc = np.mean(val_acc_trimmed, axis=0)
avg_train_loss = np.mean(train_loss_trimmed, axis=0)
avg_val_loss = np.mean(val_loss_trimmed, axis=0)

epochs_range = range(1, min_epochs + 1)

# ✅ Grafik Akurasi (SMOOTH - tanpa marker)
plt.figure(figsize=(10,5))
plt.plot(epochs_range, avg_train_acc, label="Train Accuracy",
linewidth=2.5, color='#1f77b4')
plt.plot(epochs_range, avg_val_acc, label="Validation Accuracy",
linewidth=2.5, color='#ff7f0e')
plt.xlabel("Epoch", fontsize=12)
```

```
plt.ylabel("Accuracy", fontsize=12)
smote_status = "dengan SMOTE" if use_smote else "tanpa SMOTE"
plt.title(f"Rata-rata Akurasi per Epoch (5 Fold, Early Stop
{smote_status})", fontsize=14)
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig("avg_accuracy_epoch_smote.png", dpi=300)
plt.show()

# ✅ Grafik Loss (SMOOTH - tanpa marker)
plt.figure(figsize=(10,5))
plt.plot(epochs_range, avg_train_loss, label="Train Loss",
linewidth=2.5, color='#1f77b4')
plt.plot(epochs_range, avg_val_loss, label="Validation Loss",
linewidth=2.5, color='#ff7f0e')
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Loss", fontsize=12)
plt.title(f"Rata-rata Loss per Epoch (5 Fold, Early Stop
{smote_status})", fontsize=14)
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig("avg_loss_epoch_smote.png", dpi=300)
plt.show()

# ============================================================
# 🔟 Confusion Matrix & Classification Report
# ============================================================
cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
xticklabels=label_map.keys(), yticklabels=label_map.keys())
plt.xlabel("Predicted", fontsize=12)
plt.ylabel("True", fontsize=12)
plt.title("Confusion Matrix (Keseluruhan - dengan SMOTE)",
fontsize=14)
plt.tight_layout()
plt.savefig("confusion_matrix_overall_smote.png", dpi=300)
plt.show()

target_names = list(label_map.keys())

# ✅ Custom classification report dengan nilai bulat (persentase)
report_dict = classification_report(all_labels, all_preds,
target_names=target_names, output_dict=True, zero_division=0)

print("\n===== 📋 Classification Report Final (dengan SMOTE) =====")
print(f"{'Class':<15} {'Precision':<12} {'Recall':<12} {'F1-
Score':<12} {'Support':<10}")
print("=" * 65)

# Print per kelas
for label_name in target_names:
    prec = report_dict[label_name]['precision'] * 100
    rec = report_dict[label_name]['recall'] * 100
    f1 = report_dict[label_name]['f1-score'] * 100
    sup = int(report_dict[label_name]['support'])
    print(f"{label_name:<15}
{prec:>6.0f}%       {rec:>6.0f}%       {f1:>6.0f}%       {sup:>6}")

print("-" * 65)

# Print accuracy
```

```
acc = report_dict['accuracy'] * 100
total_support = int(report_dict['weighted avg']['support'])
print(f"{'Accuracy':<15} {'':<12} {'':<12}
{acc:>6.0f}%       {total_support:>6}")

print("-" * 65)

# Print macro avg
macro_prec = report_dict['macro avg']['precision'] * 100
macro_rec = report_dict['macro avg']['recall'] * 100
macro_f1 = report_dict['macro avg']['f1-score'] * 100
print(f"{'Macro Avg':<15}
{macro_prec:>6.0f}%       {macro_rec:>6.0f}%       {macro_f1:>6.0f}%
    {total_support:>6}")

# Print weighted avg
weighted_prec = report_dict['weighted avg']['precision'] * 100
weighted_rec = report_dict['weighted avg']['recall'] * 100
weighted_f1 = report_dict['weighted avg']['f1-score'] * 100
print(f"{'Weighted Avg':<15}
{weighted_prec:>6.0f}%       {weighted_rec:>6.0f}%       {weighted_f1:
>6.0f}%       {total_support:>6}")

# Simpan ke CSV
report_df = pd.DataFrame(report_dict).transpose()
report_df_pct = report_df.copy()
for col in ['precision', 'recall', 'f1-score']:
    if col in report_df_pct.columns:
        report_df_pct[col] = (report_df_pct[col] *
100).round(0).astype(int).astype(str) + '%'
report_df_pct.to_csv("classification_report_overall_smote.csv",
index=True)

print(f"\n✅ Panjang sumbu X pada grafik: {min_epochs} epoch")
print(f"✅ Semua hasil (grafik & laporan) sudah tersimpan dan tampil
di output Colab.")
print(f"✅ SMOTE berhasil diterapkan untuk membuat data sintetis bagi
kelas minoritas.")
```

Lampiran 13. *Code* Skenario 4

```
# ============================================================
# 1️⃣ Import Library
# ============================================================
import torch
import numpy as np
import pandas as pd
from torch.utils.data import DataLoader, TensorDataset, RandomSampler,
SequentialSampler
from sklearn.model_selection import KFold
from sklearn.metrics import classification_report, confusion_matrix,
precision_score, recall_score, f1_score
from transformers import BertTokenizer, BertForSequenceClassification
from torch.optim import AdamW
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
from imblearn.over_sampling import SMOTE  # ✅ GANTI: Import SMOTE
from collections import Counter

# ============================================================
# 2️⃣ Load Dataset
# ============================================================
```

```
df = pd.read_excel("/content/3-Data Normalize.xlsx")

TEXT_COLUMN = "normalize"
LABEL_COLUMN = "label"

# mapping label → angka
label_map = {label: idx for idx, label in
enumerate(df[LABEL_COLUMN].unique())}
df["label_id"] = df[LABEL_COLUMN].map(label_map)
print("Label mapping:", label_map)

texts = df[TEXT_COLUMN].astype(str).tolist()
labels = df["label_id"].astype(int).tolist()

# ✅ Tampilkan distribusi kelas asli
print("\n===== Distribusi Kelas Original =====")
label_counts = Counter(labels)
for label, count in label_counts.items():
    label_name = [k for k, v in label_map.items() if v == label][0]
    print(f"{label_name}: {count} ({count/len(labels)*100:.2f}%)")

tokenizer = BertTokenizer.from_pretrained("indobenchmark/indobert-
base-p1")
encodings = tokenizer(
    texts, truncation=True, padding=True, max_length=128,
return_tensors="pt"
)

input_ids = encodings["input_ids"]
attention_mask = encodings["attention_mask"]
labels_tensor = torch.tensor(labels)

# ============================================================
# 3️⃣ Parameter Training
# ============================================================
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
num_epochs = 100
num_folds = 5
batch_size = 32
lr = 1e-5
patience = 10
use_smote = True

# ============================================================
# 4️⃣ Early Stopping Class
# ============================================================
class EarlyStopping:
    def __init__(self, patience=10, min_delta=0, verbose=True):
        self.patience = patience
        self.min_delta = min_delta
        self.verbose = verbose
        self.counter = 0
        self.best_loss = None
        self.early_stop = False
        self.best_epoch = 0

    def __call__(self, val_loss, epoch):
        if self.best_loss is None:
            self.best_loss = val_loss
            self.best_epoch = epoch
        elif val_loss > self.best_loss - self.min_delta:
            self.counter += 1
            if self.verbose:
                print(f"EarlyStopping counter:
{self.counter}/{self.patience}")
```

```
            if self.counter >= self.patience:
                self.early_stop = True
        else:
            self.best_loss = val_loss
            self.best_epoch = epoch
            self.counter = 0

# ========================================================
# 5️⃣ K-Fold Cross Validation dengan Early Stopping + SMOTE
# ========================================================
kfold = KFold(n_splits=num_folds, shuffle=True, random_state=42)

fold_train_acc, fold_val_acc = [], []
fold_prec, fold_rec, fold_f1 = [], [], []
all_preds, all_labels = [], []

train_acc_per_epoch, val_acc_per_epoch = [], []
train_loss_per_epoch, val_loss_per_epoch = [], []
epochs_completed_per_fold = []
best_epoch_per_fold = []

# ✅ TAMBAHAN: Tracking info SMOTE per fold
smote_info_per_fold = []

for fold, (train_idx, val_idx) in
enumerate(kfold.split(range(len(labels)))):
    print(f"\n{'='*60}")
    print(f"========== Fold {fold+1} ==========")
    print(f"{'='*60}")

    # ✅ LANGKAH 1: Ambil data train dan validation
    train_input_ids = input_ids[train_idx]
    train_attention_mask = attention_mask[train_idx]
    train_labels = labels_tensor[train_idx]

    val_input_ids = input_ids[val_idx]
    val_attention_mask = attention_mask[val_idx]
    val_labels = labels_tensor[val_idx]

    print(f"\n--- Distribusi Sebelum SMOTE ---")
    train_label_counts = Counter(train_labels.numpy())
    for label, count in sorted(train_label_counts.items()):
        label_name = [k for k, v in label_map.items() if v ==
label][0]
        print(f"  Train {label_name}: {count}")

    # ✅ LANGKAH 2: Terapkan SMOTE pada Training Set
    if use_smote:
        print(f"\n⚙️ Menerapkan SMOTE pada Training Set...")
        print(f"   SMOTE akan membuat data sintetis untuk kelas
minoritas")

        # Gabungkan input_ids dan attention_mask untuk SMOTE
        original_shape = train_input_ids.shape

        # Reshape untuk SMOTE: (num_samples, features)
        train_data = torch.cat([
            train_input_ids,
            train_attention_mask
        ], dim=1).numpy()

        # Apply SMOTE
        # k_neighbors=5 artinya menggunakan 5 tetangga terdekat untuk
interpolasi
```

```
        # Pastikan k_neighbors <= jumlah sampel kelas minoritas - 1
        min_class_count = min(train_label_counts.values())
        k_neighbors = min(5, min_class_count - 1) if min_class_count >
1 else 1

        smote = SMOTE(random_state=42, k_neighbors=k_neighbors)
        train_data_resampled, train_labels_resampled =
smote.fit_resample(
            train_data,
            train_labels.numpy()
        )

        # Split kembali menjadi input_ids dan attention_mask
        seq_length = original_shape[1]
        train_input_ids = torch.tensor(train_data_resampled[:,
:seq_length])
        train_attention_mask = torch.tensor(train_data_resampled[:,
seq_length:])
        train_labels = torch.tensor(train_labels_resampled)

        print(f"\n--- Distribusi Setelah SMOTE ---")
        train_label_counts_after = Counter(train_labels.numpy())
        for label, count in sorted(train_label_counts_after.items()):
            label_name = [k for k, v in label_map.items() if v ==
label][0]
            print(f"  Train {label_name}: {count}")

        print(f"\n📊 Ukuran dataset:")
        print(f"  Training: {len(train_labels)} samples (setelah
SMOTE)")
        print(f"  Validation: {len(val_labels)} samples (TIDAK di-
SMOTE)")

        # Simpan info SMOTE
        before_count = len(train_idx)
        after_count = len(train_labels)
        smote_info = {
            'fold': fold + 1,
            'before': before_count,
            'after': after_count,
            'addition': after_count - before_count,
            'addition_pct': ((after_count - before_count) /
before_count) * 100
        }
        smote_info_per_fold.append(smote_info)
        print(f"  ✅ Data sintetis bertambah:
{smote_info['addition']} sampel ({smote_info['addition_pct']:.1f}%)")

    # ✅ LANGKAH 3: Buat DataLoader
    train_dataset = TensorDataset(train_input_ids,
train_attention_mask, train_labels)
    val_dataset = TensorDataset(val_input_ids, val_attention_mask,
val_labels)

    train_loader = DataLoader(train_dataset,
sampler=RandomSampler(train_dataset), batch_size=batch_size)
    val_loader = DataLoader(val_dataset,
sampler=SequentialSampler(val_dataset), batch_size=batch_size)

    # ✅ LANGKAH 4: Inisialisasi Model
    model = BertForSequenceClassification.from_pretrained(
        "indobenchmark/indobert-base-p1",
        num_labels=len(label_map)
    ).to(device)
```

```
    optimizer = AdamW(model.parameters(), lr=lr)
    early_stopping = EarlyStopping(patience=patience, verbose=True)

    fold_train_acc_epoch, fold_val_acc_epoch = [], []
    fold_train_loss_epoch, fold_val_loss_epoch = [], []

    # ✅ LANGKAH 5: Training Loop dengan Early Stopping
    for epoch in range(num_epochs):
        # ---- Training ----
        model.train()
        total_loss, correct, total = 0, 0, 0
        for batch in tqdm(train_loader, desc=f"Training Epoch
{epoch+1}"):
            optimizer.zero_grad()
            input_ids_batch, attention_mask_batch, labels_batch =
[b.to(device) for b in batch]
            outputs = model(input_ids_batch,
attention_mask=attention_mask_batch, labels=labels_batch)
            loss = outputs.loss
            logits = outputs.logits
            loss.backward()
            optimizer.step()

            total_loss += loss.item()
            preds = torch.argmax(logits, dim=1)
            correct += (preds == labels_batch).sum().item()
            total += labels_batch.size(0)

        train_acc = correct / total
        avg_train_loss = total_loss / len(train_loader)

        # ---- Validation (pada data ASLI, tanpa SMOTE) ----
        model.eval()
        val_loss, val_correct, val_total = 0, 0, 0
        val_preds, val_true = [], []
        with torch.no_grad():
            for batch in val_loader:
                input_ids_batch, attention_mask_batch, labels_batch =
[b.to(device) for b in batch]
                outputs = model(input_ids_batch,
attention_mask=attention_mask_batch, labels=labels_batch)
                val_loss += outputs.loss.item()
                preds = torch.argmax(outputs.logits, dim=1)
                val_correct += (preds == labels_batch).sum().item()
                val_total += labels_batch.size(0)
                val_preds.extend(preds.cpu().numpy())
                val_true.extend(labels_batch.cpu().numpy())

        val_acc = val_correct / val_total
        avg_val_loss = val_loss / len(val_loader)

        fold_train_acc_epoch.append(train_acc)
        fold_val_acc_epoch.append(val_acc)
        fold_train_loss_epoch.append(avg_train_loss)
        fold_val_loss_epoch.append(avg_val_loss)

        print(f"Epoch {epoch+1}/{num_epochs} - Train Loss:
{avg_train_loss:.4f}, Val Loss: {avg_val_loss:.4f}, "
              f"Train Acc: {train_acc:.4f}, Val Acc: {val_acc:.4f}")

        # ✅ LANGKAH 6: Early Stopping memantau validation loss
        early_stopping(avg_val_loss, epoch+1)

        if early_stopping.early_stop:
```

```
            print(f"⚠️ Early stopping triggered at epoch {epoch+1}")
            print(f"✅ Best validation loss was
{early_stopping.best_loss:.4f} at epoch {early_stopping.best_epoch}")
            break

    epochs_completed = len(fold_train_acc_epoch)
    epochs_completed_per_fold.append(epochs_completed)
    best_epoch_per_fold.append(early_stopping.best_epoch)
    print(f"📊 Fold {fold+1} completed {epochs_completed} epochs
(best at epoch {early_stopping.best_epoch})")

    # Metrics per fold
    precision = precision_score(val_true, val_preds,
average="weighted", zero_division=0)
    recall = recall_score(val_true, val_preds, average="weighted",
zero_division=0)
    f1 = f1_score(val_true, val_preds, average="weighted",
zero_division=0)

    fold_train_acc.append(train_acc)
    fold_val_acc.append(val_acc)
    fold_prec.append(precision)
    fold_rec.append(recall)
    fold_f1.append(f1)

    all_preds.extend(val_preds)
    all_labels.extend(val_true)

    train_acc_per_epoch.append(fold_train_acc_epoch)
    val_acc_per_epoch.append(fold_val_acc_epoch)
    train_loss_per_epoch.append(fold_train_loss_epoch)
    val_loss_per_epoch.append(fold_val_loss_epoch)

    print(f"Fold {fold+1} - Train Acc: {train_acc:.4f} | Val Acc:
{val_acc:.4f} | "
          f"Prec: {precision:.4f} | Rec: {recall:.4f} | F1: {f1:.4f}")

# ============================================================
# 6️⃣ Informasi SMOTE (Synthetic Minority Over-sampling Technique)
# ============================================================
if use_smote:
    print("\n===== 📊 Informasi SMOTE =====")
    smote_df = pd.DataFrame(smote_info_per_fold)
    print(smote_df.to_string(index=False))
    print(f"\nRata-rata penambahan data sintetis:
{smote_df['addition'].mean():.1f} sampel
({smote_df['addition_pct'].mean():.1f}%)")
    print(f"Total data training sebelum SMOTE:
{smote_df['before'].mean():.0f}")
    print(f"Total data training setelah SMOTE:
{smote_df['after'].mean():.0f}")

# ============================================================
# 7️⃣ Informasi Early Stopping
# ============================================================
print("\n===== ⏱️ Informasi Early Stopping =====")
print(f"Epoch minimum: {min(epochs_completed_per_fold)}")
print(f"Epoch maksimum: {max(epochs_completed_per_fold)}")
print(f"Rata-rata epoch: {np.mean(epochs_completed_per_fold):.2f}")
print(f"Epoch per fold: {epochs_completed_per_fold}")
print(f"\nBest epoch per fold: {best_epoch_per_fold}")
print(f"Rata-rata best epoch: {np.mean(best_epoch_per_fold):.2f}")
print(f"Best epoch minimum: {min(best_epoch_per_fold)}")
print(f"Best epoch maksimum: {max(best_epoch_per_fold)}")
```

```
# ============================================================
# 8️⃣ Rata-rata Akhir 5 Fold
# ============================================================
print("\n===== 📈 Rata-Rata Hasil 5 Fold =====")
print(f"SMOTE: {'AKTIF' if use_smote else 'TIDAK AKTIF'}")
print(f"Train Acc: {np.mean(fold_train_acc):.4f}")
print(f"Val Acc:   {np.mean(fold_val_acc):.4f}")
print(f"Precision: {np.mean(fold_prec):.4f}")
print(f"Recall:    {np.mean(fold_rec):.4f}")
print(f"F1 Score:  {np.mean(fold_f1):.4f}")

# ============================================================
# 9️⃣ Grafik Akurasi & Loss Rata-rata per Epoch (SMOOTH - TANPA
MARKER)
# ============================================================
min_epochs = min(epochs_completed_per_fold)

train_acc_trimmed = [fold[:min_epochs] for fold in
train_acc_per_epoch]
val_acc_trimmed = [fold[:min_epochs] for fold in val_acc_per_epoch]
train_loss_trimmed = [fold[:min_epochs] for fold in
train_loss_per_epoch]
val_loss_trimmed = [fold[:min_epochs] for fold in val_loss_per_epoch]

avg_train_acc = np.mean(train_acc_trimmed, axis=0)
avg_val_acc = np.mean(val_acc_trimmed, axis=0)
avg_train_loss = np.mean(train_loss_trimmed, axis=0)
avg_val_loss = np.mean(val_loss_trimmed, axis=0)

epochs_range = range(1, min_epochs + 1)

# ✅ Grafik Akurasi (SMOOTH - tanpa marker)
plt.figure(figsize=(10,5))
plt.plot(epochs_range, avg_train_acc, label="Train Accuracy",
linewidth=2.5, color='#1f77b4')
plt.plot(epochs_range, avg_val_acc, label="Validation Accuracy",
linewidth=2.5, color='#ff7f0e')
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Accuracy", fontsize=12)
smote_status = "dengan SMOTE" if use_smote else "tanpa SMOTE"
plt.title(f"Rata-rata Akurasi per Epoch (5 Fold, Early Stop
{smote_status})", fontsize=14)
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig("avg_accuracy_epoch_smote.png", dpi=300)
plt.show()

# ✅ Grafik Loss (SMOOTH - tanpa marker)
plt.figure(figsize=(10,5))
plt.plot(epochs_range, avg_train_loss, label="Train Loss",
linewidth=2.5, color='#1f77b4')
plt.plot(epochs_range, avg_val_loss, label="Validation Loss",
linewidth=2.5, color='#ff7f0e')
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Loss", fontsize=12)
plt.title(f"Rata-rata Loss per Epoch (5 Fold, Early Stop
{smote_status})", fontsize=14)
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig("avg_loss_epoch_smote.png", dpi=300)
plt.show()
```

```
# ============================================================
# 🔟 Confusion Matrix & Classification Report
# ============================================================
cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
xticklabels=label_map.keys(), yticklabels=label_map.keys())
plt.xlabel("Predicted", fontsize=12)
plt.ylabel("True", fontsize=12)
plt.title("Confusion Matrix (Keseluruhan - dengan SMOTE)",
fontsize=14)
plt.tight_layout()
plt.savefig("confusion_matrix_overall_smote.png", dpi=300)
plt.show()

target_names = list(label_map.keys())

# ✅ Custom classification report dengan nilai bulat (persentase)
report_dict = classification_report(all_labels, all_preds,
target_names=target_names, output_dict=True, zero_division=0)

print("\n===== 📋 Classification Report Final (dengan SMOTE) =====")
print(f"{'Class':<15} {'Precision':<12} {'Recall':<12} {'F1-
Score':<12} {'Support':<10}")
print("=" * 65)

# Print per kelas
for label_name in target_names:
    prec = report_dict[label_name]['precision'] * 100
    rec = report_dict[label_name]['recall'] * 100
    f1 = report_dict[label_name]['f1-score'] * 100
    sup = int(report_dict[label_name]['support'])
    print(f"{label_name:<15}
{prec:>6.0f}%       {rec:>6.0f}%       {f1:>6.0f}%       {sup:>6}")

print("-" * 65)

# Print accuracy
acc = report_dict['accuracy'] * 100
total_support = int(report_dict['weighted avg']['support'])
print(f"{'Accuracy':<15} {'':<12} {'':<12}
{acc:>6.0f}%       {total_support:>6}")

print("-" * 65)

# Print macro avg
macro_prec = report_dict['macro avg']['precision'] * 100
macro_rec = report_dict['macro avg']['recall'] * 100
macro_f1 = report_dict['macro avg']['f1-score'] * 100
print(f"{'Macro Avg':<15}
{macro_prec:>6.0f}%       {macro_rec:>6.0f}%       {macro_f1:>6.0f}%
    {total_support:>6}")

# Print weighted avg
weighted_prec = report_dict['weighted avg']['precision'] * 100
weighted_rec = report_dict['weighted avg']['recall'] * 100
weighted_f1 = report_dict['weighted avg']['f1-score'] * 100
print(f"{'Weighted Avg':<15}
{weighted_prec:>6.0f}%       {weighted_rec:>6.0f}%       {weighted_f1:
>6.0f}%       {total_support:>6}")

# Simpan ke CSV
report_df = pd.DataFrame(report_dict).transpose()
report_df_pct = report_df.copy()
```

```
for col in ['precision', 'recall', 'f1-score']:
    if col in report_df_pct.columns:
        report_df_pct[col] = (report_df_pct[col] *
100).round(0).astype(int).astype(str) + '%'
report_df_pct.to_csv("classification_report_overall_smote.csv",
index=True)

print(f"\n✅ Panjang sumbu X pada grafik: {min_epochs} epoch")
print(f"✅ Semua hasil (grafik & laporan) sudah tersimpan dan tampil
di output Colab.")
print(f"✅ SMOTE berhasil diterapkan untuk membuat data sintetis bagi
kelas minoritas.")
```

Lampiran 14. *Code* Skenario 5

```
# ============================================================
# 1️⃣ Import Library
# ============================================================
import torch
import numpy as np
import pandas as pd
from torch.utils.data import DataLoader, TensorDataset, RandomSampler,
SequentialSampler
from sklearn.model_selection import KFold
from sklearn.metrics import classification_report, confusion_matrix,
precision_score, recall_score, f1_score
from transformers import BertTokenizer, BertForSequenceClassification
from torch.optim import AdamW
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter

# ============================================================
# 2️⃣ Load Dataset
# ============================================================
df = pd.read_excel("/content/3-Data Normalize.xlsx")

TEXT_COLUMN = "normalize"
LABEL_COLUMN = "label"

# mapping label → angka
label_map = {label: idx for idx, label in
enumerate(df[LABEL_COLUMN].unique())}
df["label_id"] = df[LABEL_COLUMN].map(label_map)
print("Label mapping:", label_map)

texts = df[TEXT_COLUMN].astype(str).tolist()
labels = df["label_id"].astype(int).tolist()

# ✅ Tampilkan distribusi kelas asli
print("\n===== Distribusi Kelas Original =====")
label_counts = Counter(labels)
for label, count in label_counts.items():
    label_name = [k for k, v in label_map.items() if v == label][0]
    print(f"{label_name}: {count} ({count/len(labels)*100:.2f}%)")

tokenizer = BertTokenizer.from_pretrained("indobenchmark/indobert-
base-p1")
encodings = tokenizer(
    texts, truncation=True, padding=True, max_length=128,
return_tensors="pt"
```

```
)

input_ids = encodings["input_ids"]
attention_mask = encodings["attention_mask"]
labels_tensor = torch.tensor(labels)

# ============================================================
# 3️⃣ Parameter Training
# ============================================================
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
num_epochs = 100
num_folds = 5
batch_size = 16
lr = 1e-5
patience = 10
use_rus = True  # ✅ GANTI: Aktifkan RUS

# ============================================================
# 4️⃣ Early Stopping Class
# ============================================================
class EarlyStopping:
    def __init__(self, patience=10, min_delta=0, verbose=True):
        self.patience = patience
        self.min_delta = min_delta
        self.verbose = verbose
        self.counter = 0
        self.best_loss = None
        self.early_stop = False
        self.best_epoch = 0

    def __call__(self, val_loss, epoch):
        if self.best_loss is None:
            self.best_loss = val_loss
            self.best_epoch = epoch
        elif val_loss > self.best_loss - self.min_delta:
            self.counter += 1
            if self.verbose:
                print(f"EarlyStopping counter:
{self.counter}/{self.patience}")
            if self.counter >= self.patience:
                self.early_stop = True
        else:
            self.best_loss = val_loss
            self.best_epoch = epoch
            self.counter = 0

# ============================================================
# 5️⃣ K-Fold Cross Validation dengan Early Stopping + RUS
# ============================================================
kfold = KFold(n_splits=num_folds, shuffle=True, random_state=42)

fold_train_acc, fold_val_acc = [], []
fold_prec, fold_rec, fold_f1 = [], [], []
all_preds, all_labels = [], []

train_acc_per_epoch, val_acc_per_epoch = [], []
train_loss_per_epoch, val_loss_per_epoch = [], []
epochs_completed_per_fold = []
best_epoch_per_fold = []

# ✅ TAMBAHAN: Tracking info RUS per fold
rus_info_per_fold = []

for fold, (train_idx, val_idx) in
```

```
enumerate(kfold.split(range(len(labels)))):
    print(f"\n{'='*60}")
    print(f"========== Fold {fold+1} ==========")
    print(f"{'='*60}")

    # ✅ LANGKAH 1: Ambil data train dan validation
    train_input_ids = input_ids[train_idx]
    train_attention_mask = attention_mask[train_idx]
    train_labels = labels_tensor[train_idx]

    val_input_ids = input_ids[val_idx]
    val_attention_mask = attention_mask[val_idx]
    val_labels = labels_tensor[val_idx]

    print(f"\n--- Distribusi Sebelum RUS ---")
    train_label_counts = Counter(train_labels.numpy())
    for label, count in sorted(train_label_counts.items()):
        label_name = [k for k, v in label_map.items() if v ==
label][0]
        print(f"  Train {label_name}: {count}")

    # ✅ LANGKAH 2: Terapkan RUS pada Training Set
    if use_rus:
        print(f"\n⚙️ Menerapkan Random Under Sampling pada Training
Set...")

        # Gabungkan input_ids dan attention_mask untuk RUS
        # Karena RUS butuh format 2D, kita perlu flatten dan simpan
shape
        original_shape = train_input_ids.shape

        # Reshape untuk RUS: (num_samples, features)
        train_data = torch.cat([
            train_input_ids,
            train_attention_mask
        ], dim=1).numpy()

        # Apply RUS (GANTI dari ROS ke RUS)
        rus = RandomUnderSampler(random_state=42)
        train_data_resampled, train_labels_resampled =
rus.fit_resample(
            train_data,
            train_labels.numpy()
        )

        # Split kembali menjadi input_ids dan attention_mask
        seq_length = original_shape[1]
        train_input_ids = torch.tensor(train_data_resampled[:,
:seq_length])
        train_attention_mask = torch.tensor(train_data_resampled[:,
seq_length:])
        train_labels = torch.tensor(train_labels_resampled)

        print(f"\n--- Distribusi Setelah RUS ---")
        train_label_counts_after = Counter(train_labels.numpy())
        for label, count in sorted(train_label_counts_after.items()):
            label_name = [k for k, v in label_map.items() if v ==
label][0]
            print(f"  Train {label_name}: {count}")

        print(f"\n📊 Ukuran dataset:")
        print(f"  Training: {len(train_labels)} samples (setelah
RUS)")
        print(f"  Validation: {len(val_labels)} samples (TIDAK di-
```

```
RUS)")

        # Simpan info RUS
        before_count = len(train_idx)
        after_count = len(train_labels)
        rus_info = {
            'fold': fold + 1,
            'before': before_count,
            'after': after_count,
            'reduction': before_count - after_count,
            'reduction_pct': ((before_count - after_count) /
before_count) * 100
        }
        rus_info_per_fold.append(rus_info)
        print(f"  ⚠️ Data berkurang: {rus_info['reduction']} sampel
({rus_info['reduction_pct']:.1f}%)")

    # ✅ LANGKAH 3: Buat DataLoader
    train_dataset = TensorDataset(train_input_ids,
train_attention_mask, train_labels)
    val_dataset = TensorDataset(val_input_ids, val_attention_mask,
val_labels)

    train_loader = DataLoader(train_dataset,
sampler=RandomSampler(train_dataset), batch_size=batch_size)
    val_loader = DataLoader(val_dataset,
sampler=SequentialSampler(val_dataset), batch_size=batch_size)

    # ✅ LANGKAH 4: Inisialisasi Model
    model = BertForSequenceClassification.from_pretrained(
        "indobenchmark/indobert-base-p1",
        num_labels=len(label_map)
    ).to(device)

    optimizer = AdamW(model.parameters(), lr=lr)
    early_stopping = EarlyStopping(patience=patience, verbose=True)

    fold_train_acc_epoch, fold_val_acc_epoch = [], []
    fold_train_loss_epoch, fold_val_loss_epoch = [], []

    # ✅ LANGKAH 5: Training Loop dengan Early Stopping
    for epoch in range(num_epochs):
        # ---- Training ----
        model.train()
        total_loss, correct, total = 0, 0, 0
        for batch in tqdm(train_loader, desc=f"Training Epoch
{epoch+1}"):
            optimizer.zero_grad()
            input_ids_batch, attention_mask_batch, labels_batch =
[b.to(device) for b in batch]
            outputs = model(input_ids_batch,
attention_mask=attention_mask_batch, labels=labels_batch)
            loss = outputs.loss
            logits = outputs.logits
            loss.backward()
            optimizer.step()

            total_loss += loss.item()
            preds = torch.argmax(logits, dim=1)
            correct += (preds == labels_batch).sum().item()
            total += labels_batch.size(0)

        train_acc = correct / total
        avg_train_loss = total_loss / len(train_loader)
```

```
        # ---- Validation (pada data ASLI, tanpa RUS) ----
        model.eval()
        val_loss, val_correct, val_total = 0, 0, 0
        val_preds, val_true = [], []
        with torch.no_grad():
            for batch in val_loader:
                input_ids_batch, attention_mask_batch, labels_batch =
[b.to(device) for b in batch]
                outputs = model(input_ids_batch,
attention_mask=attention_mask_batch, labels=labels_batch)
                val_loss += outputs.loss.item()
                preds = torch.argmax(outputs.logits, dim=1)
                val_correct += (preds == labels_batch).sum().item()
                val_total += labels_batch.size(0)
                val_preds.extend(preds.cpu().numpy())
                val_true.extend(labels_batch.cpu().numpy())

        val_acc = val_correct / val_total
        avg_val_loss = val_loss / len(val_loader)

        fold_train_acc_epoch.append(train_acc)
        fold_val_acc_epoch.append(val_acc)
        fold_train_loss_epoch.append(avg_train_loss)
        fold_val_loss_epoch.append(avg_val_loss)

        print(f"Epoch {epoch+1}/{num_epochs} - Train Loss:
{avg_train_loss:.4f}, Val Loss: {avg_val_loss:.4f}, "
              f"Train Acc: {train_acc:.4f}, Val Acc: {val_acc:.4f}")

        # ✅ LANGKAH 6: Early Stopping memantau validation loss
        early_stopping(avg_val_loss, epoch+1)

        if early_stopping.early_stop:
            print(f"⚠️ Early stopping triggered at epoch {epoch+1}")
            print(f"✅ Best validation loss was
{early_stopping.best_loss:.4f} at epoch {early_stopping.best_epoch}")
            break

    epochs_completed = len(fold_train_acc_epoch)
    epochs_completed_per_fold.append(epochs_completed)
    best_epoch_per_fold.append(early_stopping.best_epoch)
    print(f"📊 Fold {fold+1} completed {epochs_completed} epochs
(best at epoch {early_stopping.best_epoch})")

    # Metrics per fold
    precision = precision_score(val_true, val_preds,
average="weighted", zero_division=0)
    recall = recall_score(val_true, val_preds, average="weighted",
zero_division=0)
    f1 = f1_score(val_true, val_preds, average="weighted",
zero_division=0)

    fold_train_acc.append(train_acc)
    fold_val_acc.append(val_acc)
    fold_prec.append(precision)
    fold_rec.append(recall)
    fold_f1.append(f1)

    all_preds.extend(val_preds)
    all_labels.extend(val_true)

    train_acc_per_epoch.append(fold_train_acc_epoch)
    val_acc_per_epoch.append(fold_val_acc_epoch)
```

```
    train_loss_per_epoch.append(fold_train_loss_epoch)
    val_loss_per_epoch.append(fold_val_loss_epoch)

    print(f"Fold {fold+1} - Train Acc: {train_acc:.4f} | Val Acc:
{val_acc:.4f} | "
          f"Prec: {precision:.4f} | Rec: {recall:.4f} | F1: {f1:.4f}")

# ============================================================
# 6 Informasi RUS (Random Under Sampling)
# ============================================================
if use_rus:
    print("\n===== 📊 Informasi Random Under Sampling (RUS) =====")
    rus_df = pd.DataFrame(rus_info_per_fold)
    print(rus_df.to_string(index=False))
    print(f"\nRata-rata pengurangan data:
{rus_df['reduction'].mean():.1f} sampel
({rus_df['reduction_pct'].mean():.1f}%)")
    print(f"Total data training sebelum RUS:
{rus_df['before'].mean():.0f}")
    print(f"Total data training setelah RUS:
{rus_df['after'].mean():.0f}")

# ============================================================
# 7 Informasi Early Stopping
# ============================================================
print("\n===== ⏱ Informasi Early Stopping =====")
print(f"Epoch minimum: {min(epochs_completed_per_fold)}")
print(f"Epoch maksimum: {max(epochs_completed_per_fold)}")
print(f"Rata-rata epoch: {np.mean(epochs_completed_per_fold):.2f} ±
{np.std(epochs_completed_per_fold):.2f}")
print(f"Epoch per fold: {epochs_completed_per_fold}")
print(f"\nBest epoch per fold: {best_epoch_per_fold}")
print(f"Rata-rata best epoch: {np.mean(best_epoch_per_fold):.2f} ±
{np.std(best_epoch_per_fold):.2f}")
print(f"Best epoch minimum: {min(best_epoch_per_fold)}")
print(f"Best epoch maksimum: {max(best_epoch_per_fold)}")

# ============================================================
# 8 Rata-rata Akhir 5 Fold
# ============================================================
print("\n===== 📈 Rata-Rata Hasil 5 Fold =====")
print(f"Random Under Sampling: {'AKTIF' if use_rus else 'TIDAK
AKTIF'}")
print(f"Train Acc: {np.mean(fold_train_acc):.4f}")
print(f"Val Acc:   {np.mean(fold_val_acc):.4f}")
print(f"Precision: {np.mean(fold_prec):.4f}")
print(f"Recall:    {np.mean(fold_rec):.4f}")
print(f"F1 Score:  {np.mean(fold_f1):.4f}")

# ============================================================
# 9 Grafik Akurasi & Loss Rata-rata per Epoch (TANPA MARKER)
# ============================================================
min_epochs = min(epochs_completed_per_fold)

train_acc_trimmed = [fold[:min_epochs] for fold in
train_acc_per_epoch]
val_acc_trimmed = [fold[:min_epochs] for fold in val_acc_per_epoch]
train_loss_trimmed = [fold[:min_epochs] for fold in
train_loss_per_epoch]
val_loss_trimmed = [fold[:min_epochs] for fold in val_loss_per_epoch]

avg_train_acc = np.mean(train_acc_trimmed, axis=0)
avg_val_acc = np.mean(val_acc_trimmed, axis=0)
avg_train_loss = np.mean(train_loss_trimmed, axis=0)
```

```
avg_val_loss = np.mean(val_loss_trimmed, axis=0)

epochs_range = range(1, min_epochs + 1)

# ✅ Grafik Akurasi (TANPA MARKER, garis mulus)
plt.figure(figsize=(10,5))
plt.plot(epochs_range, avg_train_acc, label="Train Accuracy",
linewidth=2.5, color='#1f77b4')
plt.plot(epochs_range, avg_val_acc, label="Validation Accuracy",
linewidth=2.5, color='#ff7f0e')
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Accuracy", fontsize=12)
rus_status = "dengan RUS" if use_rus else "tanpa RUS"
plt.title(f"Rata-rata Akurasi per Epoch (5 Fold, Early Stop
{rus_status})", fontsize=14)
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig("avg_accuracy_epoch_rus.png", dpi=300)
plt.show()

# ✅ Grafik Loss (TANPA MARKER, garis mulus)
plt.figure(figsize=(10,5))
plt.plot(epochs_range, avg_train_loss, label="Train Loss",
linewidth=2.5, color='#1f77b4')
plt.plot(epochs_range, avg_val_loss, label="Validation Loss",
linewidth=2.5, color='#ff7f0e')
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Loss", fontsize=12)
plt.title(f"Rata-rata Loss per Epoch (5 Fold, Early Stop
{rus_status})", fontsize=14)
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig("avg_loss_epoch_rus.png", dpi=300)
plt.show()

# ============================================================
# 🔟 Confusion Matrix & Classification Report
# ============================================================
cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
xticklabels=label_map.keys(), yticklabels=label_map.keys())
plt.xlabel("Predicted", fontsize=12)
plt.ylabel("True", fontsize=12)
plt.title("Confusion Matrix (Keseluruhan - dengan RUS)", fontsize=14)
plt.tight_layout()
plt.savefig("confusion_matrix_overall_rus.png", dpi=300)
plt.show()

target_names = list(label_map.keys())

# ✅ MODIFIKASI: Buat custom classification report dengan nilai bulat
(persentase)
report_dict = classification_report(all_labels, all_preds,
target_names=target_names, output_dict=True, zero_division=0)

print("\n===== 📋 Classification Report Final (dengan RUS) =====")
print(f"{'Class':<15} {'Precision':<12} {'Recall':<12} {'F1-
Score':<12} {'Support':<10}")
print("=" * 65)

# Print per kelas
```

```
for label_name in target_names:
    prec = report_dict[label_name]['precision'] * 100  # Convert ke
persentase
    rec = report_dict[label_name]['recall'] * 100
    f1 = report_dict[label_name]['f1-score'] * 100
    sup = int(report_dict[label_name]['support'])
    print(f"{label_name:<15}
{prec:>6.0f}%       {rec:>6.0f}%       {f1:>6.0f}%       {sup:>6}")

print("-" * 65)

# Print accuracy
acc = report_dict['accuracy'] * 100
total_support = int(report_dict['weighted avg']['support'])
print(f"{'Accuracy':<15} {'':<12} {'':<12}
{acc:>6.0f}%       {total_support:>6}")

print("-" * 65)

# Print macro avg
macro_prec = report_dict['macro avg']['precision'] * 100
macro_rec = report_dict['macro avg']['recall'] * 100
macro_f1 = report_dict['macro avg']['f1-score'] * 100
print(f"{'Macro Avg':<15}
{macro_prec:>6.0f}%       {macro_rec:>6.0f}%       {macro_f1:>6.0f}%
    {total_support:>6}")

# Print weighted avg
weighted_prec = report_dict['weighted avg']['precision'] * 100
weighted_rec = report_dict['weighted avg']['recall'] * 100
weighted_f1 = report_dict['weighted avg']['f1-score'] * 100
print(f"{'Weighted Avg':<15}
{weighted_prec:>6.0f}%       {weighted_rec:>6.0f}%       {weighted_f1:
>6.0f}%       {total_support:>6}")

# Simpan ke CSV dengan format persentase bulat
report_df = pd.DataFrame(report_dict).transpose()
report_df_pct = report_df.copy()
for col in ['precision', 'recall', 'f1-score']:
    if col in report_df_pct.columns:
        report_df_pct[col] = (report_df_pct[col] *
100).round(0).astype(int).astype(str) + '%'
report_df_pct.to_csv("classification_report_overall_rus.csv",
index=True)

print(f"\n✅ Panjang sumbu X pada grafik: {min_epochs} epoch")
print(f"✅ Semua hasil (grafik & laporan) sudah tersimpan dan tampil
di output Colab.")
print(f"✅ RUS berhasil diterapkan pada setiap fold untuk
menyeimbangkan kelas minoritas dan mayoritas.")
```

Lampiran 15. *Code* Skenario 6

```
# ============================================================
# 1️⃣ Import Library
# ============================================================
import torch
import numpy as np
import pandas as pd
from torch.utils.data import DataLoader, TensorDataset, RandomSampler,
SequentialSampler
from sklearn.model_selection import KFold
from sklearn.metrics import classification_report, confusion_matrix,
precision_score, recall_score, f1_score
from transformers import BertTokenizer, BertForSequenceClassification
from torch.optim import AdamW
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter

# ============================================================
# 2️⃣ Load Dataset
# ============================================================
df = pd.read_excel("/content/3-Data Normalize.xlsx")

TEXT_COLUMN = "normalize"
LABEL_COLUMN = "label"

# mapping label → angka
label_map = {label: idx for idx, label in
enumerate(df[LABEL_COLUMN].unique())}
df["label_id"] = df[LABEL_COLUMN].map(label_map)
print("Label mapping:", label_map)

texts = df[TEXT_COLUMN].astype(str).tolist()
labels = df["label_id"].astype(int).tolist()

# ✅ Tampilkan distribusi kelas asli
print("\n===== Distribusi Kelas Original =====")
label_counts = Counter(labels)
for label, count in label_counts.items():
    label_name = [k for k, v in label_map.items() if v == label][0]
    print(f"{label_name}: {count} ({count/len(labels)*100:.2f}%)")

tokenizer = BertTokenizer.from_pretrained("indobenchmark/indobert-
base-p1")
encodings = tokenizer(
    texts, truncation=True, padding=True, max_length=128,
return_tensors="pt"
)

input_ids = encodings["input_ids"]
attention_mask = encodings["attention_mask"]
labels_tensor = torch.tensor(labels)

# ============================================================
# 3️⃣ Parameter Training
# ============================================================
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
num_epochs = 100
num_folds = 5
batch_size = 32
lr = 1e-5
```

```
patience = 10
use_rus = True  # ✅ GANTI: Aktifkan RUS

# ============================================================
# 4️⃣ Early Stopping Class
# ============================================================
class EarlyStopping:
    def __init__(self, patience=10, min_delta=0, verbose=True):
        self.patience = patience
        self.min_delta = min_delta
        self.verbose = verbose
        self.counter = 0
        self.best_loss = None
        self.early_stop = False
        self.best_epoch = 0

    def __call__(self, val_loss, epoch):
        if self.best_loss is None:
            self.best_loss = val_loss
            self.best_epoch = epoch
        elif val_loss > self.best_loss - self.min_delta:
            self.counter += 1
            if self.verbose:
                print(f"EarlyStopping counter:
{self.counter}/{self.patience}")
            if self.counter >= self.patience:
                self.early_stop = True
        else:
            self.best_loss = val_loss
            self.best_epoch = epoch
            self.counter = 0

# ============================================================
# 5️⃣ K-Fold Cross Validation dengan Early Stopping + RUS
# ============================================================
kfold = KFold(n_splits=num_folds, shuffle=True, random_state=42)

fold_train_acc, fold_val_acc = [], []
fold_prec, fold_rec, fold_f1 = [], [], []
all_preds, all_labels = [], []

train_acc_per_epoch, val_acc_per_epoch = [], []
train_loss_per_epoch, val_loss_per_epoch = [], []
epochs_completed_per_fold = []
best_epoch_per_fold = []

# ✅ TAMBAHAN: Tracking info RUS per fold
rus_info_per_fold = []

for fold, (train_idx, val_idx) in
enumerate(kfold.split(range(len(labels)))):
    print(f"\n{'='*60}")
    print(f"========== Fold {fold+1} ==========")
    print(f"{'='*60}")

    # ✅ LANGKAH 1: Ambil data train dan validation
    train_input_ids = input_ids[train_idx]
    train_attention_mask = attention_mask[train_idx]
    train_labels = labels_tensor[train_idx]

    val_input_ids = input_ids[val_idx]
    val_attention_mask = attention_mask[val_idx]
    val_labels = labels_tensor[val_idx]
```

```
    print(f"\n--- Distribusi Sebelum RUS ---")
    train_label_counts = Counter(train_labels.numpy())
    for label, count in sorted(train_label_counts.items()):
        label_name = [k for k, v in label_map.items() if v ==
label][0]
        print(f"  Train {label_name}: {count}")

    # ✅ LANGKAH 2: Terapkan RUS pada Training Set
    if use_rus:
        print(f"\n⚙️ Menerapkan Random Under Sampling pada Training
Set...")

        # Gabungkan input_ids dan attention_mask untuk RUS
        # Karena RUS butuh format 2D, kita perlu flatten dan simpan
shape
        original_shape = train_input_ids.shape

        # Reshape untuk RUS: (num_samples, features)
        train_data = torch.cat([
            train_input_ids,
            train_attention_mask
        ], dim=1).numpy()

        # Apply RUS (GANTI dari ROS ke RUS)
        rus = RandomUnderSampler(random_state=42)
        train_data_resampled, train_labels_resampled =
rus.fit_resample(
            train_data,
            train_labels.numpy()
        )

        # Split kembali menjadi input_ids dan attention_mask
        seq_length = original_shape[1]
        train_input_ids = torch.tensor(train_data_resampled[:,
:seq_length])
        train_attention_mask = torch.tensor(train_data_resampled[:,
seq_length:])
        train_labels = torch.tensor(train_labels_resampled)

        print(f"\n--- Distribusi Setelah RUS ---")
        train_label_counts_after = Counter(train_labels.numpy())
        for label, count in sorted(train_label_counts_after.items()):
            label_name = [k for k, v in label_map.items() if v ==
label][0]
            print(f"  Train {label_name}: {count}")

        print(f"\n📊 Ukuran dataset:")
        print(f"  Training: {len(train_labels)} samples (setelah
RUS)")
        print(f"  Validation: {len(val_labels)} samples (TIDAK di-
RUS)")

        # Simpan info RUS
        before_count = len(train_idx)
        after_count = len(train_labels)
        rus_info = {
            'fold': fold + 1,
            'before': before_count,
            'after': after_count,
            'reduction': before_count - after_count,
            'reduction_pct': ((before_count - after_count) /
before_count) * 100
        }
        rus_info_per_fold.append(rus_info)
```

```
        print(f"  ⚠️ Data berkurang: {rus_info['reduction']} sampel
({rus_info['reduction_pct']:.1f}%)")

    # ✅ LANGKAH 3: Buat DataLoader
    train_dataset = TensorDataset(train_input_ids,
train_attention_mask, train_labels)
    val_dataset = TensorDataset(val_input_ids, val_attention_mask,
val_labels)

    train_loader = DataLoader(train_dataset,
sampler=RandomSampler(train_dataset), batch_size=batch_size)
    val_loader = DataLoader(val_dataset,
sampler=SequentialSampler(val_dataset), batch_size=batch_size)

    # ✅ LANGKAH 4: Inisialisasi Model
    model = BertForSequenceClassification.from_pretrained(
        "indobenchmark/indobert-base-p1",
        num_labels=len(label_map)
    ).to(device)

    optimizer = AdamW(model.parameters(), lr=lr)
    early_stopping = EarlyStopping(patience=patience, verbose=True)

    fold_train_acc_epoch, fold_val_acc_epoch = [], []
    fold_train_loss_epoch, fold_val_loss_epoch = [], []

    # ✅ LANGKAH 5: Training Loop dengan Early Stopping
    for epoch in range(num_epochs):
        # ---- Training ----
        model.train()
        total_loss, correct, total = 0, 0, 0
        for batch in tqdm(train_loader, desc=f"Training Epoch
{epoch+1}"):
            optimizer.zero_grad()
            input_ids_batch, attention_mask_batch, labels_batch =
[b.to(device) for b in batch]
            outputs = model(input_ids_batch,
attention_mask=attention_mask_batch, labels=labels_batch)
            loss = outputs.loss
            logits = outputs.logits
            loss.backward()
            optimizer.step()

            total_loss += loss.item()
            preds = torch.argmax(logits, dim=1)
            correct += (preds == labels_batch).sum().item()
            total += labels_batch.size(0)

        train_acc = correct / total
        avg_train_loss = total_loss / len(train_loader)

        # ---- Validation (pada data ASLI, tanpa RUS) ----
        model.eval()
        val_loss, val_correct, val_total = 0, 0, 0
        val_preds, val_true = [], []
        with torch.no_grad():
            for batch in val_loader:
                input_ids_batch, attention_mask_batch, labels_batch =
[b.to(device) for b in batch]
                outputs = model(input_ids_batch,
attention_mask=attention_mask_batch, labels=labels_batch)
                val_loss += outputs.loss.item()
                preds = torch.argmax(outputs.logits, dim=1)
                val_correct += (preds == labels_batch).sum().item()
```

```
                val_total += labels_batch.size(0)
                val_preds.extend(preds.cpu().numpy())
                val_true.extend(labels_batch.cpu().numpy())

        val_acc = val_correct / val_total
        avg_val_loss = val_loss / len(val_loader)

        fold_train_acc_epoch.append(train_acc)
        fold_val_acc_epoch.append(val_acc)
        fold_train_loss_epoch.append(avg_train_loss)
        fold_val_loss_epoch.append(avg_val_loss)

        print(f"Epoch {epoch+1}/{num_epochs} - Train Loss:
{avg_train_loss:.4f}, Val Loss: {avg_val_loss:.4f}, "
              f"Train Acc: {train_acc:.4f}, Val Acc: {val_acc:.4f}")

        # ✅ LANGKAH 6: Early Stopping memantau validation loss
        early_stopping(avg_val_loss, epoch+1)

        if early_stopping.early_stop:
            print(f"⚠️ Early stopping triggered at epoch {epoch+1}")
            print(f"✅ Best validation loss was
{early_stopping.best_loss:.4f} at epoch {early_stopping.best_epoch}")
            break

    epochs_completed = len(fold_train_acc_epoch)
    epochs_completed_per_fold.append(epochs_completed)
    best_epoch_per_fold.append(early_stopping.best_epoch)
    print(f"📊 Fold {fold+1} completed {epochs_completed} epochs
(best at epoch {early_stopping.best_epoch})")

    # Metrics per fold
    precision = precision_score(val_true, val_preds,
average="weighted", zero_division=0)
    recall = recall_score(val_true, val_preds, average="weighted",
zero_division=0)
    f1 = f1_score(val_true, val_preds, average="weighted",
zero_division=0)

    fold_train_acc.append(train_acc)
    fold_val_acc.append(val_acc)
    fold_prec.append(precision)
    fold_rec.append(recall)
    fold_f1.append(f1)

    all_preds.extend(val_preds)
    all_labels.extend(val_true)

    train_acc_per_epoch.append(fold_train_acc_epoch)
    val_acc_per_epoch.append(fold_val_acc_epoch)
    train_loss_per_epoch.append(fold_train_loss_epoch)
    val_loss_per_epoch.append(fold_val_loss_epoch)

    print(f"Fold {fold+1} - Train Acc: {train_acc:.4f} | Val Acc:
{val_acc:.4f} | "
          f"Prec: {precision:.4f} | Rec: {recall:.4f} | F1: {f1:.4f}")

# ============================================================
# 6️⃣ Informasi RUS (Random Under Sampling)
# ============================================================
if use_rus:
    print("\n===== 📊 Informasi Random Under Sampling (RUS) =====")
    rus_df = pd.DataFrame(rus_info_per_fold)
    print(rus_df.to_string(index=False))
```

```
    print(f"\nRata-rata pengurangan data:
{rus_df['reduction'].mean():.1f} sampel
({rus_df['reduction_pct'].mean():.1f}%)")
    print(f"Total data training sebelum RUS:
{rus_df['before'].mean():.0f}")
    print(f"Total data training setelah RUS:
{rus_df['after'].mean():.0f}")

# ============================================================
# 7️⃣ Informasi Early Stopping
# ============================================================
print("\n===== ⏱️ Informasi Early Stopping =====")
print(f"Epoch minimum: {min(epochs_completed_per_fold)}")
print(f"Epoch maksimum: {max(epochs_completed_per_fold)}")
print(f"Rata-rata epoch: {np.mean(epochs_completed_per_fold):.2f} ±
{np.std(epochs_completed_per_fold):.2f}")
print(f"Epoch per fold: {epochs_completed_per_fold}")
print(f"\nBest epoch per fold: {best_epoch_per_fold}")
print(f"Rata-rata best epoch: {np.mean(best_epoch_per_fold):.2f} ±
{np.std(best_epoch_per_fold):.2f}")
print(f"Best epoch minimum: {min(best_epoch_per_fold)}")
print(f"Best epoch maksimum: {max(best_epoch_per_fold)}")

# ============================================================
# 8️⃣ Rata-rata Akhir 5 Fold
# ============================================================
print("\n===== 📈 Rata-Rata Hasil 5 Fold =====")
print(f"Random Under Sampling: {'AKTIF' if use_rus else 'TIDAK
AKTIF'}")
print(f"Train Acc: {np.mean(fold_train_acc):.4f}")
print(f"Val Acc:   {np.mean(fold_val_acc):.4f}")
print(f"Precision: {np.mean(fold_prec):.4f}")
print(f"Recall:    {np.mean(fold_rec):.4f}")
print(f"F1 Score:  {np.mean(fold_f1):.4f}")

# ============================================================
# 9️⃣ Grafik Akurasi & Loss Rata-rata per Epoch (TANPA MARKER)
# ============================================================
min_epochs = min(epochs_completed_per_fold)

train_acc_trimmed = [fold[:min_epochs] for fold in
train_acc_per_epoch]
val_acc_trimmed = [fold[:min_epochs] for fold in val_acc_per_epoch]
train_loss_trimmed = [fold[:min_epochs] for fold in
train_loss_per_epoch]
val_loss_trimmed = [fold[:min_epochs] for fold in val_loss_per_epoch]

avg_train_acc = np.mean(train_acc_trimmed, axis=0)
avg_val_acc = np.mean(val_acc_trimmed, axis=0)
avg_train_loss = np.mean(train_loss_trimmed, axis=0)
avg_val_loss = np.mean(val_loss_trimmed, axis=0)

epochs_range = range(1, min_epochs + 1)

# ✅ Grafik Akurasi (TANPA MARKER, garis mulus)
plt.figure(figsize=(10,5))
plt.plot(epochs_range, avg_train_acc, label="Train Accuracy",
linewidth=2.5, color='#1f77b4')
plt.plot(epochs_range, avg_val_acc, label="Validation Accuracy",
linewidth=2.5, color='#ff7f0e')
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Accuracy", fontsize=12)
rus_status = "dengan RUS" if use_rus else "tanpa RUS"
plt.title(f"Rata-rata Akurasi per Epoch (5 Fold, Early Stop
```

```
{rus_status})", fontsize=14)
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig("avg_accuracy_epoch_rus.png", dpi=300)
plt.show()

# ✅ Grafik Loss (TANPA MARKER, garis mulus)
plt.figure(figsize=(10,5))
plt.plot(epochs_range, avg_train_loss, label="Train Loss",
linewidth=2.5, color='#1f77b4')
plt.plot(epochs_range, avg_val_loss, label="Validation Loss",
linewidth=2.5, color='#ff7f0e')
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Loss", fontsize=12)
plt.title(f"Rata-rata Loss per Epoch (5 Fold, Early Stop
{rus_status})", fontsize=14)
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig("avg_loss_epoch_rus.png", dpi=300)
plt.show()

# ============================================================
# 🔟 Confusion Matrix & Classification Report
# ============================================================
cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
xticklabels=label_map.keys(), yticklabels=label_map.keys())
plt.xlabel("Predicted", fontsize=12)
plt.ylabel("True", fontsize=12)
plt.title("Confusion Matrix (Keseluruhan - dengan RUS)", fontsize=14)
plt.tight_layout()
plt.savefig("confusion_matrix_overall_rus.png", dpi=300)
plt.show()

target_names = list(label_map.keys())

# ✅ MODIFIKASI: Buat custom classification report dengan nilai bulat
(persentase)
report_dict = classification_report(all_labels, all_preds,
target_names=target_names, output_dict=True, zero_division=0)

print("\n===== 📋 Classification Report Final (dengan RUS) =====")
print(f"{'Class':<15} {'Precision':<12} {'Recall':<12} {'F1-
Score':<12} {'Support':<10}")
print("=" * 65)

# Print per kelas
for label_name in target_names:
    prec = report_dict[label_name]['precision'] * 100  # Convert ke
persentase
    rec = report_dict[label_name]['recall'] * 100
    f1 = report_dict[label_name]['f1-score'] * 100
    sup = int(report_dict[label_name]['support'])
    print(f"{label_name:<15}
{prec:>6.0f}%       {rec:>6.0f}%       {f1:>6.0f}%       {sup:>6}")

print("-" * 65)

# Print accuracy
acc = report_dict['accuracy'] * 100
total_support = int(report_dict['weighted avg']['support'])
```

```
print(f"{'Accuracy':<15} {'':<12} {'':<12}
{acc:>6.0f}%       {total_support:>6}")

print("-" * 65)

# Print macro avg
macro_prec = report_dict['macro avg']['precision'] * 100
macro_rec = report_dict['macro avg']['recall'] * 100
macro_f1 = report_dict['macro avg']['f1-score'] * 100
print(f"{'Macro Avg':<15}
{macro_prec:>6.0f}%       {macro_rec:>6.0f}%       {macro_f1:>6.0f}%
    {total_support:>6}")

# Print weighted avg
weighted_prec = report_dict['weighted avg']['precision'] * 100
weighted_rec = report_dict['weighted avg']['recall'] * 100
weighted_f1 = report_dict['weighted avg']['f1-score'] * 100
print(f"{'Weighted Avg':<15}
{weighted_prec:>6.0f}%       {weighted_rec:>6.0f}%       {weighted_f1:
>6.0f}%       {total_support:>6}")

# Simpan ke CSV dengan format persentase bulat
report_df = pd.DataFrame(report_dict).transpose()
report_df_pct = report_df.copy()
for col in ['precision', 'recall', 'f1-score']:
    if col in report_df_pct.columns:
        report_df_pct[col] = (report_df_pct[col] *
100).round(0).astype(int).astype(str) + '%'
report_df_pct.to_csv("classification_report_overall_rus.csv",
index=True)

print(f"\n✅ Panjang sumbu X pada grafik: {min_epochs} epoch")
print(f"✅ Semua hasil (grafik & laporan) sudah tersimpan dan tampil
di output Colab.")
print(f"✅ RUS berhasil diterapkan pada setiap fold untuk
menyeimbangkan kelas minoritas dan mayoritas.")
```

Lampiran 16. *Code* Skenario 7

```
# ============================================================
# 1️⃣ Import Library
# ============================================================
import torch
import numpy as np
import pandas as pd
from torch.utils.data import DataLoader, TensorDataset, RandomSampler,
SequentialSampler
from sklearn.model_selection import KFold
from sklearn.metrics import classification_report, confusion_matrix,
precision_score, recall_score, f1_score
from transformers import BertTokenizer, BertForSequenceClassification
from torch.optim import AdamW
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
from imblearn.over_sampling import RandomOverSampler
from collections import Counter

# ============================================================
# 2️⃣ Load Dataset
# ============================================================
df = pd.read_excel("/content/3-Data Normalize.xlsx")
```

```
TEXT_COLUMN = "normalize"
LABEL_COLUMN = "label"

# mapping label → angka
label_map = {label: idx for idx, label in
enumerate(df[LABEL_COLUMN].unique())}
df["label_id"] = df[LABEL_COLUMN].map(label_map)
print("Label mapping:", label_map)

texts = df[TEXT_COLUMN].astype(str).tolist()
labels = df["label_id"].astype(int).tolist()

# ✅ Tampilkan distribusi kelas asli
print("\n===== Distribusi Kelas Original =====")
label_counts = Counter(labels)
for label, count in label_counts.items():
    label_name = [k for k, v in label_map.items() if v == label][0]
    print(f"{label_name}: {count} ({count/len(labels)*100:.2f}%)")

tokenizer = BertTokenizer.from_pretrained("indobenchmark/indobert-
base-p1")
encodings = tokenizer(
        texts, truncation=True, padding=True, max_length=128,
return_tensors="pt"
)

input_ids = encodings["input_ids"]
attention_mask = encodings["attention_mask"]
labels_tensor = torch.tensor(labels)

# ============================================================
# 3️⃣ Parameter Training
# ============================================================
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
num_epochs = 100
num_folds = 5
batch_size = 16
lr = 1e-5
patience = 10
use_ros = True

# ============================================================
# 4️⃣ Early Stopping Class
# ============================================================
class EarlyStopping:
    def __init__(self, patience=10, min_delta=0, verbose=True):
        self.patience = patience
        self.min_delta = min_delta
        self.verbose = verbose
        self.counter = 0
        self.best_loss = None
        self.early_stop = False
        self.best_epoch = 0

    def __call__(self, val_loss, epoch):
        if self.best_loss is None:
            self.best_loss = val_loss
            self.best_epoch = epoch
        elif val_loss > self.best_loss - self.min_delta:
            self.counter += 1
            if self.verbose:
                print(f"EarlyStopping counter:
{self.counter}/{self.patience}")
            if self.counter >= self.patience:
                self.early_stop = True
```

```
        else:
            self.best_loss = val_loss
            self.best_epoch = epoch
            self.counter = 0

# ============================================================
# 5️⃣ K-Fold Cross Validation dengan Early Stopping + ROS
# ============================================================
kfold = KFold(n_splits=num_folds, shuffle=True, random_state=42)

fold_train_acc, fold_val_acc = [], []
fold_prec, fold_rec, fold_f1 = [], [], []
all_preds, all_labels = [], []

train_acc_per_epoch, val_acc_per_epoch = [], []
train_loss_per_epoch, val_loss_per_epoch = [], []
epochs_completed_per_fold = []
best_epoch_per_fold = []

# ✅ TAMBAHAN: Tracking info ROS per fold
ros_info_per_fold = []

for           fold,           (train_idx,           val_idx)           in
enumerate(kfold.split(range(len(labels)))):
    print(f"\n{'='*60}")
    print(f"========== Fold {fold+1} ==========")
    print(f"{'='*60}")

    # ✅ LANGKAH 1: Ambil data train dan validation
    train_input_ids = input_ids[train_idx]
    train_attention_mask = attention_mask[train_idx]
    train_labels = labels_tensor[train_idx]

    val_input_ids = input_ids[val_idx]
    val_attention_mask = attention_mask[val_idx]
    val_labels = labels_tensor[val_idx]

    print(f"\n--- Distribusi Sebelum ROS ---")
    train_label_counts = Counter(train_labels.numpy())
    for label, count in sorted(train_label_counts.items()):
        label_name = [k for k, v in label_map.items() if v == label][0]
        print(f"  Train {label_name}: {count}")

    # ✅ LANGKAH 2: Terapkan ROS pada Training Set
    if use_ros:
          print(f"\n⚙️ Menerapkan Random Over Sampling pada Training
Set...")

        # Gabungkan input_ids dan attention_mask untuk ROS
        original_shape = train_input_ids.shape

        # Reshape untuk ROS: (num_samples, features)
        train_data = torch.cat([
            train_input_ids,
            train_attention_mask
        ], dim=1).numpy()

        # Apply ROS
        ros = RandomOverSampler(random_state=42)
                    train_data_resampled,    train_labels_resampled    =
ros.fit_resample(
            train_data,
            train_labels.numpy()
        )
```

```
        # Split kembali menjadi input_ids dan attention_mask
        seq_length = original_shape[1]
             train_input_ids   =   torch.tensor(train_data_resampled[:,
:seq_length])
          train_attention_mask  =  torch.tensor(train_data_resampled[:,
seq_length:])
        train_labels = torch.tensor(train_labels_resampled)

        print(f"\n--- Distribusi Setelah ROS ---")
        train_label_counts_after = Counter(train_labels.numpy())
        for label, count in sorted(train_label_counts_after.items()):
             label_name  =  [k  for  k,  v  in  label_map.items()  if  v  ==
label][0]
            print(f"  Train {label_name}: {count}")

        print(f"\n📊 Ukuran dataset:")
       print(f"  Training: {len(train_labels)} samples (setelah ROS)")
      print(f"  Validation: {len(val_labels)} samples (TIDAK di-ROS)")

        # Simpan info ROS
        before_count = len(train_idx)
        after_count = len(train_labels)
        ros_info = {
            'fold': fold + 1,
            'before': before_count,
            'after': after_count,
            'addition': after_count - before_count,
                    'addition_pct':  ((after_count  -  before_count)  /
before_count) * 100
        }
        ros_info_per_fold.append(ros_info)
         print(f"  ✅ Data bertambah: {ros_info['addition']} sampel
({ros_info['addition_pct']:.1f}%)")

    # ✅ LANGKAH 3: Buat DataLoader
             train_dataset      =      TensorDataset(train_input_ids,
train_attention_mask, train_labels)
     val_dataset  =  TensorDataset(val_input_ids,  val_attention_mask,
val_labels)

                train_loader       =       DataLoader(train_dataset,
sampler=RandomSampler(train_dataset), batch_size=batch_size)
                  val_loader       =       DataLoader(val_dataset,
sampler=SequentialSampler(val_dataset), batch_size=batch_size)

    # ✅ LANGKAH 4: Inisialisasi Model
    model = BertForSequenceClassification.from_pretrained(
        "indobenchmark/indobert-base-p1",
        num_labels=len(label_map)
    ).to(device)

    optimizer = AdamW(model.parameters(), lr=lr)
    early_stopping = EarlyStopping(patience=patience, verbose=True)

    fold_train_acc_epoch, fold_val_acc_epoch = [], []
    fold_train_loss_epoch, fold_val_loss_epoch = [], []

    # ✅ LANGKAH 5: Training Loop dengan Early Stopping
    for epoch in range(num_epochs):
        # ---- Training ----
        model.train()
        total_loss, correct, total = 0, 0, 0
            for  batch  in  tqdm(train_loader,  desc=f"Training  Epoch
```

```
{epoch+1}"):
            optimizer.zero_grad()
               input_ids_batch, attention_mask_batch, labels_batch =
[b.to(device) for b in batch]
                                outputs     =     model(input_ids_batch,
attention_mask=attention_mask_batch, labels=labels_batch)
            loss = outputs.loss
            logits = outputs.logits
            loss.backward()
            optimizer.step()

            total_loss += loss.item()
            preds = torch.argmax(logits, dim=1)
            correct += (preds == labels_batch).sum().item()
            total += labels_batch.size(0)

        train_acc = correct / total
        avg_train_loss = total_loss / len(train_loader)

        # ---- Validation (pada data ASLI, tanpa ROS) ----
        model.eval()
        val_loss, val_correct, val_total = 0, 0, 0
        val_preds, val_true = [], []
        with torch.no_grad():
            for batch in val_loader:
                input_ids_batch, attention_mask_batch, labels_batch =
[b.to(device) for b in batch]
                                    outputs    =    model(input_ids_batch,
attention_mask=attention_mask_batch, labels=labels_batch)
                val_loss += outputs.loss.item()
                preds = torch.argmax(outputs.logits, dim=1)
                val_correct += (preds == labels_batch).sum().item()
                val_total += labels_batch.size(0)
                val_preds.extend(preds.cpu().numpy())
                val_true.extend(labels_batch.cpu().numpy())

        val_acc = val_correct / val_total
        avg_val_loss = val_loss / len(val_loader)

        fold_train_acc_epoch.append(train_acc)
        fold_val_acc_epoch.append(val_acc)
        fold_train_loss_epoch.append(avg_train_loss)
        fold_val_loss_epoch.append(avg_val_loss)

              print(f"Epoch   {epoch+1}/{num_epochs}   -   Train   Loss:
{avg_train_loss:.4f}, Val Loss: {avg_val_loss:.4f}, "
              f"Train Acc: {train_acc:.4f}, Val Acc: {val_acc:.4f}")

        # ✅ LANGKAH 6: Early Stopping memantau validation loss
        early_stopping(avg_val_loss, epoch+1)

        if early_stopping.early_stop:
            print(f"⚠️ Early stopping triggered at epoch {epoch+1}")
                        print(f"✅    Best    validation    loss    was
{early_stopping.best_loss:.4f} at epoch {early_stopping.best_epoch}")
            break

    epochs_completed = len(fold_train_acc_epoch)
    epochs_completed_per_fold.append(epochs_completed)
    best_epoch_per_fold.append(early_stopping.best_epoch)
    print(f"📊 Fold {fold+1} completed {epochs_completed} epochs (best
at epoch {early_stopping.best_epoch})")

    # Metrics per fold
```

```
            precision     =     precision_score(val_true,     val_preds,
average="weighted", zero_division=0)
      recall  =  recall_score(val_true,  val_preds,  average="weighted",
zero_division=0)
         f1    =    f1_score(val_true,    val_preds,    average="weighted",
zero_division=0)

    fold_train_acc.append(train_acc)
    fold_val_acc.append(val_acc)
    fold_prec.append(precision)
    fold_rec.append(recall)
    fold_f1.append(f1)

    all_preds.extend(val_preds)
    all_labels.extend(val_true)

    train_acc_per_epoch.append(fold_train_acc_epoch)
    val_acc_per_epoch.append(fold_val_acc_epoch)
    train_loss_per_epoch.append(fold_train_loss_epoch)
    val_loss_per_epoch.append(fold_val_loss_epoch)

     print(f"Fold  {fold+1}  -  Train  Acc:  {train_acc:.4f}  |  Val  Acc:
{val_acc:.4f} | "
          f"Prec: {precision:.4f} | Rec: {recall:.4f} | F1: {f1:.4f}")

# ============================================================
# 6 Informasi ROS (Random Over Sampling)
# ============================================================
if use_ros:
    print("\n===== 📊 Informasi Random Over Sampling (ROS) =====")
    ros_df = pd.DataFrame(ros_info_per_fold)
    print(ros_df.to_string(index=False))
                    print(f"\nRata-rata          penambahan          data:
{ros_df['addition'].mean():.1f}                                  sampel
({ros_df['addition_pct'].mean():.1f}%)")
              print(f"Total      data      training      sebelum      ROS:
{ros_df['before'].mean():.0f}")
              print(f"Total      data      training      setelah      ROS:
{ros_df['after'].mean():.0f}")

# ============================================================
# 7 Informasi Early Stopping
# ============================================================
print("\n===== ⏱ Informasi Early Stopping =====")
print(f"Epoch minimum: {min(epochs_completed_per_fold)}")
print(f"Epoch maksimum: {max(epochs_completed_per_fold)}")
print(f"Rata-rata epoch: {np.mean(epochs_completed_per_fold):.2f}")
print(f"Epoch per fold: {epochs_completed_per_fold}")
print(f"\nBest epoch per fold: {best_epoch_per_fold}")
print(f"Rata-rata best epoch: {np.mean(best_epoch_per_fold):.2f}")
print(f"Best epoch minimum: {min(best_epoch_per_fold)}")
print(f"Best epoch maksimum: {max(best_epoch_per_fold)}")

# ============================================================
# 8 Rata-rata Akhir 5 Fold
# ============================================================
print("\n===== 📈 Rata-Rata Hasil 5 Fold =====")
print(f"Random Over Sampling: {'AKTIF' if use_ros else 'TIDAK AKTIF'}")
print(f"Train Acc: {np.mean(fold_train_acc):.4f}")
print(f"Val Acc:   {np.mean(fold_val_acc):.4f}")
print(f"Precision: {np.mean(fold_prec):.4f}")
print(f"Recall:    {np.mean(fold_rec):.4f}")
print(f"F1 Score:  {np.mean(fold_f1):.4f}")
```

```
# ============================================================
# 9️⃣ Grafik Akurasi & Loss Rata-rata per Epoch (TANPA MARKER - SMOOTH)
# ============================================================
min_epochs = min(epochs_completed_per_fold)

train_acc_trimmed = [fold[:min_epochs] for fold in train_acc_per_epoch]
val_acc_trimmed = [fold[:min_epochs] for fold in val_acc_per_epoch]
train_loss_trimmed = [fold[:min_epochs] for fold in
train_loss_per_epoch]
val_loss_trimmed = [fold[:min_epochs] for fold in val_loss_per_epoch]

avg_train_acc = np.mean(train_acc_trimmed, axis=0)
avg_val_acc = np.mean(val_acc_trimmed, axis=0)
avg_train_loss = np.mean(train_loss_trimmed, axis=0)
avg_val_loss = np.mean(val_loss_trimmed, axis=0)

epochs_range = range(1, min_epochs + 1)

# ✅ Grafik Akurasi (TANPA MARKER - garis mulus)
plt.figure(figsize=(10,5))
plt.plot(epochs_range, avg_train_acc, label="Train Accuracy",
linewidth=2.5, color='#1f77b4')
plt.plot(epochs_range, avg_val_acc, label="Validation Accuracy",
linewidth=2.5, color='#ff7f0e')
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Accuracy", fontsize=12)
ros_status = "dengan ROS" if use_ros else "tanpa ROS"
plt.title(f"Rata-rata Akurasi per Epoch (5 Fold, Early Stop
{ros_status})", fontsize=14)
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig("avg_accuracy_epoch_ros.png", dpi=300)
plt.show()

# ✅ Grafik Loss (TANPA MARKER - garis mulus)
plt.figure(figsize=(10,5))
plt.plot(epochs_range, avg_train_loss, label="Train Loss",
linewidth=2.5, color='#1f77b4')
plt.plot(epochs_range, avg_val_loss, label="Validation Loss",
linewidth=2.5, color='#ff7f0e')
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Loss", fontsize=12)
plt.title(f"Rata-rata Loss per Epoch (5 Fold, Early Stop {ros_status})",
fontsize=14)
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig("avg_loss_epoch_ros.png", dpi=300)
plt.show()

# ============================================================
# 🔟 Confusion Matrix & Classification Report
# ============================================================
cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
xticklabels=label_map.keys(), yticklabels=label_map.keys())
plt.xlabel("Predicted", fontsize=12)
plt.ylabel("True", fontsize=12)
plt.title("Confusion Matrix (Keseluruhan - dengan ROS)", fontsize=14)
plt.tight_layout()
plt.savefig("confusion_matrix_overall_ros.png", dpi=300)
plt.show()
```

```
target_names = list(label_map.keys())

# ✅ MODIFIKASI: Buat custom classification report dengan nilai bulat
(persentase)
report_dict     =     classification_report(all_labels,     all_preds,
target_names=target_names, output_dict=True, zero_division=0)

print("\n===== 📋 Classification Report Final (dengan ROS) =====")
print(f"{'Class':<15}     {'Precision':<12}     {'Recall':<12}     {'F1-
Score':<12} {'Support':<10}")
print("=" * 65)

# Print per kelas
for label_name in target_names:
     prec = report_dict[label_name]['precision'] * 100  # Convert ke
persentase
    rec = report_dict[label_name]['recall'] * 100
    f1 = report_dict[label_name]['f1-score'] * 100
    sup = int(report_dict[label_name]['support'])
                                                print(f"{label_name:<15}
{prec:>6.0f}%       {rec:>6.0f}%       {f1:>6.0f}%       {sup:>6}")

print("-" * 65)

# Print accuracy
acc = report_dict['accuracy'] * 100
total_support = int(report_dict['weighted avg']['support'])
print(f"{'Accuracy':<15}                {'':<12}                {'':<12}
{acc:>6.0f}%       {total_support:>6}")

print("-" * 65)

# Print macro avg
macro_prec = report_dict['macro avg']['precision'] * 100
macro_rec = report_dict['macro avg']['recall'] * 100
macro_f1 = report_dict['macro avg']['f1-score'] * 100
print(f"{'Macro                                                    Avg':<15}
{macro_prec:>6.0f}%       {macro_rec:>6.0f}%       {macro_f1:>6.0f}%
    {total_support:>6}")

# Print weighted avg
weighted_prec = report_dict['weighted avg']['precision'] * 100
weighted_rec = report_dict['weighted avg']['recall'] * 100
weighted_f1 = report_dict['weighted avg']['f1-score'] * 100
print(f"{'Weighted                                                 Avg':<15}
{weighted_prec:>6.0f}%       {weighted_rec:>6.0f}%       {weighted_f1:
>6.0f}%       {total_support:>6}")

# Simpan ke CSV dengan format persentase bulat
report_df = pd.DataFrame(report_dict).transpose()
report_df_pct = report_df.copy()
for col in ['precision', 'recall', 'f1-score']:
    if col in report_df_pct.columns:
                    report_df_pct[col]    =    (report_df_pct[col]    *
100).round(0).astype(int).astype(str) + '%'
report_df_pct.to_csv("classification_report_overall_ros.csv",
index=True)

print(f"\n✅ Panjang sumbu X pada grafik: {min_epochs} epoch")
print(f"✅ Semua hasil (grafik & laporan) sudah tersimpan dan tampil
di output Colab.")
print(f"✅   ROS   berhasil   diterapkan   pada   setiap   fold   untuk
menyeimbangkan kelas minoritas dan mayoritas.")
```

Lampiran 17. *Code* Skenario 8

```
# ========================================================
# 1️⃣ Import Library
# ========================================================
import torch
import numpy as np
import pandas as pd
from torch.utils.data import DataLoader, TensorDataset, RandomSampler,
SequentialSampler
from sklearn.model_selection import KFold
from sklearn.metrics import classification_report, confusion_matrix,
precision_score, recall_score, f1_score
from transformers import BertTokenizer, BertForSequenceClassification
from torch.optim import AdamW
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
from imblearn.over_sampling import RandomOverSampler
from collections import Counter

# ========================================================
# 2️⃣ Load Dataset
# ========================================================
df = pd.read_excel("/content/3-Data Normalize.xlsx")

TEXT_COLUMN = "normalize"
LABEL_COLUMN = "label"

# mapping label → angka
label_map = {label: idx for idx, label in
enumerate(df[LABEL_COLUMN].unique())}
df["label_id"] = df[LABEL_COLUMN].map(label_map)
print("Label mapping:", label_map)

texts = df[TEXT_COLUMN].astype(str).tolist()
labels = df["label_id"].astype(int).tolist()

# ✅ Tampilkan distribusi kelas asli
print("\n===== Distribusi Kelas Original =====")
label_counts = Counter(labels)
for label, count in label_counts.items():
    label_name = [k for k, v in label_map.items() if v == label][0]
    print(f"{label_name}: {count} ({count/len(labels)*100:.2f}%)")

tokenizer = BertTokenizer.from_pretrained("indobenchmark/indobert-
base-p1")
encodings = tokenizer(
    texts, truncation=True, padding=True, max_length=128,
return_tensors="pt"
)

input_ids = encodings["input_ids"]
attention_mask = encodings["attention_mask"]
labels_tensor = torch.tensor(labels)

# ========================================================
# 3️⃣ Parameter Training
# ========================================================
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
num_epochs = 100
```

```
num_folds = 5
batch_size = 32
lr = 1e-5
patience = 10
use_ros = True

# ============================================================
# 4️⃣ Early Stopping Class
# ============================================================
class EarlyStopping:
    def __init__(self, patience=10, min_delta=0, verbose=True):
        self.patience = patience
        self.min_delta = min_delta
        self.verbose = verbose
        self.counter = 0
        self.best_loss = None
        self.early_stop = False
        self.best_epoch = 0

    def __call__(self, val_loss, epoch):
        if self.best_loss is None:
            self.best_loss = val_loss
            self.best_epoch = epoch
        elif val_loss > self.best_loss - self.min_delta:
            self.counter += 1
            if self.verbose:
                print(f"EarlyStopping counter:
{self.counter}/{self.patience}")
            if self.counter >= self.patience:
                self.early_stop = True
        else:
            self.best_loss = val_loss
            self.best_epoch = epoch
            self.counter = 0

# ============================================================
# 5️⃣ K-Fold Cross Validation dengan Early Stopping + ROS
# ============================================================
kfold = KFold(n_splits=num_folds, shuffle=True, random_state=42)

fold_train_acc, fold_val_acc = [], []
fold_prec, fold_rec, fold_f1 = [], [], []
all_preds, all_labels = [], []

train_acc_per_epoch, val_acc_per_epoch = [], []
train_loss_per_epoch, val_loss_per_epoch = [], []
epochs_completed_per_fold = []
best_epoch_per_fold = []

# ✅ TAMBAHAN: Tracking info ROS per fold
ros_info_per_fold = []

for fold, (train_idx, val_idx) in
enumerate(kfold.split(range(len(labels)))):
    print(f"\n{'='*60}")
    print(f"========== Fold {fold+1} ==========")
    print(f"{'='*60}")

    # ✅ LANGKAH 1: Ambil data train dan validation
    train_input_ids = input_ids[train_idx]
    train_attention_mask = attention_mask[train_idx]
    train_labels = labels_tensor[train_idx]

    val_input_ids = input_ids[val_idx]
```

```
    val_attention_mask = attention_mask[val_idx]
    val_labels = labels_tensor[val_idx]

    print(f"\n--- Distribusi Sebelum ROS ---")
    train_label_counts = Counter(train_labels.numpy())
    for label, count in sorted(train_label_counts.items()):
        label_name = [k for k, v in label_map.items() if v ==
label][0]
        print(f"  Train {label_name}: {count}")

    # ✅ LANGKAH 2: Terapkan ROS pada Training Set
    if use_ros:
        print(f"\n⚙️ Menerapkan Random Over Sampling pada Training
Set...")

        # Gabungkan input_ids dan attention_mask untuk ROS
        original_shape = train_input_ids.shape

        # Reshape untuk ROS: (num_samples, features)
        train_data = torch.cat([
            train_input_ids,
            train_attention_mask
        ], dim=1).numpy()

        # Apply ROS
        ros = RandomOverSampler(random_state=42)
        train_data_resampled, train_labels_resampled =
ros.fit_resample(
            train_data,
            train_labels.numpy()
        )

        # Split kembali menjadi input_ids dan attention_mask
        seq_length = original_shape[1]
        train_input_ids = torch.tensor(train_data_resampled[:,
:seq_length])
        train_attention_mask = torch.tensor(train_data_resampled[:,
seq_length:])
        train_labels = torch.tensor(train_labels_resampled)

        print(f"\n--- Distribusi Setelah ROS ---")
        train_label_counts_after = Counter(train_labels.numpy())
        for label, count in sorted(train_label_counts_after.items()):
            label_name = [k for k, v in label_map.items() if v ==
label][0]
            print(f"  Train {label_name}: {count}")

        print(f"\n📊 Ukuran dataset:")
        print(f"  Training: {len(train_labels)} samples (setelah
ROS)")
        print(f"  Validation: {len(val_labels)} samples (TIDAK di-
ROS)")

        # Simpan info ROS
        before_count = len(train_idx)
        after_count = len(train_labels)
        ros_info = {
            'fold': fold + 1,
            'before': before_count,
            'after': after_count,
            'addition': after_count - before_count,
            'addition_pct': ((after_count - before_count) /
before_count) * 100
        }
        ros_info_per_fold.append(ros_info)
```

```
        print(f"  ✅ Data bertambah: {ros_info['addition']} sampel
({ros_info['addition_pct']:.1f}%)")

    # ✅ LANGKAH 3: Buat DataLoader
    train_dataset = TensorDataset(train_input_ids,
train_attention_mask, train_labels)
    val_dataset = TensorDataset(val_input_ids, val_attention_mask,
val_labels)

    train_loader = DataLoader(train_dataset,
sampler=RandomSampler(train_dataset), batch_size=batch_size)
    val_loader = DataLoader(val_dataset,
sampler=SequentialSampler(val_dataset), batch_size=batch_size)

    # ✅ LANGKAH 4: Inisialisasi Model
    model = BertForSequenceClassification.from_pretrained(
        "indobenchmark/indobert-base-p1",
        num_labels=len(label_map)
    ).to(device)

    optimizer = AdamW(model.parameters(), lr=lr)
    early_stopping = EarlyStopping(patience=patience, verbose=True)

    fold_train_acc_epoch, fold_val_acc_epoch = [], []
    fold_train_loss_epoch, fold_val_loss_epoch = [], []

    # ✅ LANGKAH 5: Training Loop dengan Early Stopping
    for epoch in range(num_epochs):
        # ---- Training ----
        model.train()
        total_loss, correct, total = 0, 0, 0
        for batch in tqdm(train_loader, desc=f"Training Epoch
{epoch+1}"):
            optimizer.zero_grad()
            input_ids_batch, attention_mask_batch, labels_batch =
[b.to(device) for b in batch]
            outputs = model(input_ids_batch,
attention_mask=attention_mask_batch, labels=labels_batch)
            loss = outputs.loss
            logits = outputs.logits
            loss.backward()
            optimizer.step()

            total_loss += loss.item()
            preds = torch.argmax(logits, dim=1)
            correct += (preds == labels_batch).sum().item()
            total += labels_batch.size(0)

        train_acc = correct / total
        avg_train_loss = total_loss / len(train_loader)

        # ---- Validation (pada data ASLI, tanpa ROS) ----
        model.eval()
        val_loss, val_correct, val_total = 0, 0, 0
        val_preds, val_true = [], []
        with torch.no_grad():
            for batch in val_loader:
                input_ids_batch, attention_mask_batch, labels_batch =
[b.to(device) for b in batch]
                outputs = model(input_ids_batch,
attention_mask=attention_mask_batch, labels=labels_batch)
                val_loss += outputs.loss.item()
                preds = torch.argmax(outputs.logits, dim=1)
                val_correct += (preds == labels_batch).sum().item()
```

```
                val_total += labels_batch.size(0)
                val_preds.extend(preds.cpu().numpy())
                val_true.extend(labels_batch.cpu().numpy())

        val_acc = val_correct / val_total
        avg_val_loss = val_loss / len(val_loader)

        fold_train_acc_epoch.append(train_acc)
        fold_val_acc_epoch.append(val_acc)
        fold_train_loss_epoch.append(avg_train_loss)
        fold_val_loss_epoch.append(avg_val_loss)

        print(f"Epoch {epoch+1}/{num_epochs} - Train Loss:
{avg_train_loss:.4f}, Val Loss: {avg_val_loss:.4f}, "
              f"Train Acc: {train_acc:.4f}, Val Acc: {val_acc:.4f}")

        # ✅ LANGKAH 6: Early Stopping memantau validation loss
        early_stopping(avg_val_loss, epoch+1)

        if early_stopping.early_stop:
            print(f"⚠️ Early stopping triggered at epoch {epoch+1}")
            print(f"✅ Best validation loss was
{early_stopping.best_loss:.4f} at epoch {early_stopping.best_epoch}")
            break

    epochs_completed = len(fold_train_acc_epoch)
    epochs_completed_per_fold.append(epochs_completed)
    best_epoch_per_fold.append(early_stopping.best_epoch)
    print(f"📊 Fold {fold+1} completed {epochs_completed} epochs
(best at epoch {early_stopping.best_epoch})")

    # Metrics per fold
    precision = precision_score(val_true, val_preds,
average="weighted", zero_division=0)
    recall = recall_score(val_true, val_preds, average="weighted",
zero_division=0)
    f1 = f1_score(val_true, val_preds, average="weighted",
zero_division=0)

    fold_train_acc.append(train_acc)
    fold_val_acc.append(val_acc)
    fold_prec.append(precision)
    fold_rec.append(recall)
    fold_f1.append(f1)

    all_preds.extend(val_preds)
    all_labels.extend(val_true)

    train_acc_per_epoch.append(fold_train_acc_epoch)
    val_acc_per_epoch.append(fold_val_acc_epoch)
    train_loss_per_epoch.append(fold_train_loss_epoch)
    val_loss_per_epoch.append(fold_val_loss_epoch)

    print(f"Fold {fold+1} - Train Acc: {train_acc:.4f} | Val Acc:
{val_acc:.4f} | "
          f"Prec: {precision:.4f} | Rec: {recall:.4f} | F1: {f1:.4f}")

# ============================================================
# 6️⃣ Informasi ROS (Random Over Sampling)
# ============================================================
if use_ros:
    print("\n===== 📊 Informasi Random Over Sampling (ROS) =====")
    ros_df = pd.DataFrame(ros_info_per_fold)
    print(ros_df.to_string(index=False))
```

```
    print(f"\nRata-rata penambahan data:
{ros_df['addition'].mean():.1f} sampel
({ros_df['addition_pct'].mean():.1f}%)")
    print(f"Total data training sebelum ROS:
{ros_df['before'].mean():.0f}")
    print(f"Total data training setelah ROS:
{ros_df['after'].mean():.0f}")

# ============================================================
# 7️⃣ Informasi Early Stopping
# ============================================================
print("\n===== ⏱️ Informasi Early Stopping =====")
print(f"Epoch minimum: {min(epochs_completed_per_fold)}")
print(f"Epoch maksimum: {max(epochs_completed_per_fold)}")
print(f"Rata-rata epoch: {np.mean(epochs_completed_per_fold):.2f}")
print(f"Epoch per fold: {epochs_completed_per_fold}")
print(f"\nBest epoch per fold: {best_epoch_per_fold}")
print(f"Rata-rata best epoch: {np.mean(best_epoch_per_fold):.2f}")
print(f"Best epoch minimum: {min(best_epoch_per_fold)}")
print(f"Best epoch maksimum: {max(best_epoch_per_fold)}")

# ============================================================
# 8️⃣ Rata-rata Akhir 5 Fold
# ============================================================
print("\n===== 📈 Rata-Rata Hasil 5 Fold =====")
print(f"Random Over Sampling: {'AKTIF' if use_ros else 'TIDAK
AKTIF'}")
print(f"Train Acc: {np.mean(fold_train_acc):.4f}")
print(f"Val Acc:   {np.mean(fold_val_acc):.4f}")
print(f"Precision: {np.mean(fold_prec):.4f}")
print(f"Recall:    {np.mean(fold_rec):.4f}")
print(f"F1 Score:  {np.mean(fold_f1):.4f}")

# ============================================================
# 9️⃣ Grafik Akurasi & Loss Rata-rata per Epoch (TANPA MARKER -
SMOOTH)
# ============================================================
min_epochs = min(epochs_completed_per_fold)

train_acc_trimmed = [fold[:min_epochs] for fold in
train_acc_per_epoch]
val_acc_trimmed = [fold[:min_epochs] for fold in val_acc_per_epoch]
train_loss_trimmed = [fold[:min_epochs] for fold in
train_loss_per_epoch]
val_loss_trimmed = [fold[:min_epochs] for fold in val_loss_per_epoch]

avg_train_acc = np.mean(train_acc_trimmed, axis=0)
avg_val_acc = np.mean(val_acc_trimmed, axis=0)
avg_train_loss = np.mean(train_loss_trimmed, axis=0)
avg_val_loss = np.mean(val_loss_trimmed, axis=0)

epochs_range = range(1, min_epochs + 1)

# ✅ Grafik Akurasi (TANPA MARKER - garis mulus)
plt.figure(figsize=(10,5))
plt.plot(epochs_range, avg_train_acc, label="Train Accuracy",
linewidth=2.5, color='#1f77b4')
plt.plot(epochs_range, avg_val_acc, label="Validation Accuracy",
linewidth=2.5, color='#ff7f0e')
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Accuracy", fontsize=12)
ros_status = "dengan ROS" if use_ros else "tanpa ROS"
plt.title(f"Rata-rata Akurasi per Epoch (5 Fold, Early Stop
{ros_status})", fontsize=14)
```

```
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig("avg_accuracy_epoch_ros.png", dpi=300)
plt.show()

# ✅ Grafik Loss (TANPA MARKER - garis mulus)
plt.figure(figsize=(10,5))
plt.plot(epochs_range, avg_train_loss, label="Train Loss",
linewidth=2.5, color='#1f77b4')
plt.plot(epochs_range, avg_val_loss, label="Validation Loss",
linewidth=2.5, color='#ff7f0e')
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Loss", fontsize=12)
plt.title(f"Rata-rata Loss per Epoch (5 Fold, Early Stop
{ros_status})", fontsize=14)
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig("avg_loss_epoch_ros.png", dpi=300)
plt.show()

# ============================================================
# 🔟 Confusion Matrix & Classification Report
# ============================================================
cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
xticklabels=label_map.keys(), yticklabels=label_map.keys())
plt.xlabel("Predicted", fontsize=12)
plt.ylabel("True", fontsize=12)
plt.title("Confusion Matrix (Keseluruhan - dengan ROS)", fontsize=14)
plt.tight_layout()
plt.savefig("confusion_matrix_overall_ros.png", dpi=300)
plt.show()

target_names = list(label_map.keys())

# ✅ MODIFIKASI: Buat custom classification report dengan nilai bulat
(persentase)
report_dict = classification_report(all_labels, all_preds,
target_names=target_names, output_dict=True, zero_division=0)

print("\n===== 📋 Classification Report Final (dengan ROS) =====")
print(f"{'Class':<15} {'Precision':<12} {'Recall':<12} {'F1-
Score':<12} {'Support':<10}")
print("=" * 65)

# Print per kelas
for label_name in target_names:
    prec = report_dict[label_name]['precision'] * 100  # Convert ke
persentase
    rec = report_dict[label_name]['recall'] * 100
    f1 = report_dict[label_name]['f1-score'] * 100
    sup = int(report_dict[label_name]['support'])
    print(f"{label_name:<15}
{prec:>6.0f}%       {rec:>6.0f}%       {f1:>6.0f}%       {sup:>6}")

print("-" * 65)

# Print accuracy
acc = report_dict['accuracy'] * 100
total_support = int(report_dict['weighted avg']['support'])
print(f"{'Accuracy':<15} {'':<12} {'':<12}
```

```
{acc:>6.0f}%       {total_support:>6}")

print("-" * 65)

# Print macro avg
macro_prec = report_dict['macro avg']['precision'] * 100
macro_rec = report_dict['macro avg']['recall'] * 100
macro_f1 = report_dict['macro avg']['f1-score'] * 100
print(f"{'Macro Avg':<15}
{macro_prec:>6.0f}%       {macro_rec:>6.0f}%       {macro_f1:>6.0f}%
    {total_support:>6}")

# Print weighted avg
weighted_prec = report_dict['weighted avg']['precision'] * 100
weighted_rec = report_dict['weighted avg']['recall'] * 100
weighted_f1 = report_dict['weighted avg']['f1-score'] * 100
print(f"{'Weighted Avg':<15}
{weighted_prec:>6.0f}%       {weighted_rec:>6.0f}%       {weighted_f1:
>6.0f}%       {total_support:>6}")

# Simpan ke CSV dengan format persentase bulat
report_df = pd.DataFrame(report_dict).transpose()
report_df_pct = report_df.copy()
for col in ['precision', 'recall', 'f1-score']:
    if col in report_df_pct.columns:
        report_df_pct[col] = (report_df_pct[col] *
100).round(0).astype(int).astype(str) + '%'
report_df_pct.to_csv("classification_report_overall_ros.csv",
index=True)

print(f"\n✅ Panjang sumbu X pada grafik: {min_epochs} epoch")
print(f"✅ Semua hasil (grafik & laporan) sudah tersimpan dan tampil
di output Colab.")
print(f"✅ ROS berhasil diterapkan pada setiap fold untuk
menyeimbangkan kelas minoritas dan mayoritas.")
```