

Topik : 4.4. Finalisasi Prototipe

Objective : Siapkan pipeline end-to-end: dataset → FL → DP → hasil

Task : Uji ulang semua modul, pastikan reproducible



LOCAL (manual)

1. Dataset (Load Data)

```
17
18 # =====
19 # 0) File input
20 # =====
21 DINSOS_CSV = "dinsos.csv"
22 DUKCAPIL_CSV = "dukcapil.csv"
23 KEMENKES_CSV = "kemenkes.csv"
24
25 # =====
26 # 1) Load CSV
27 # =====
28 def load_or_fail(path):
29     p = Path(path)
30     if not p.exists():
31         raise FileNotFoundError(f"File tidak ditemukan: {path}")
32     return pd.read_csv(p)
33
34 dinsos = load_or_fail(DINSOS_CSV)
35 dukcapil = load_or_fail(DUKCAPIL_CSV)
36 kemenkes = load_or_fail(KEMENKES_CSV)
37
```

2. Labeling

```
38 # =====
39 # 2) Labeling rules
40 # =====
41 def label_dinsos(row):
42     jt = row.get("jumlah_tanggungan", np.nan)
43     ph = row.get("penghasilan", np.nan)
44     kr = row.get("kondisi_rumah", "")
45     if pd.isna(jt): jt = 0
46     if pd.isna(ph): ph = 0
47     if ph < 2_000_000: return 1
48     if (2_000_000 <= ph < 3_500_000) and jt >= 4: return 1
49     if (kr in ("tidak layak", "semi permanen", "sangat sederhana")) and (ph < 5_000_000): return 1
50     if (jt >= 5) and (ph < 6_000_000): return 1
51     if ph >= 8_000_000: return 0
52     return 0
53
54 def label_dukcapil(row):
55     u = row.get("umur", np.nan)
56     sp = row.get("status_pekerjaan", "")
57     st = row.get("status_pernikahan", "")
58     if pd.isna(u): u = 0
59     if u > 65: return 1
60     if sp in ("pengangguran", "buruh", "pekerja informal"): return 1
61     if (sp=="wirausaha") and (u>55): return 1
62     if (st in ("cerai", "janda", "duda")) and (spi=="pegawai tetap"): return 1
63     if (sp=="pegawai tetap") and (25<u<55) and (st=="menikah"): return 0
64     if (u<25) and (sp=="wirausaha"): return 0
65     return 0
66
67 def label_kemenkes(row):
68     rp = row.get("riwayat_penyakit", "")
69     sg = row.get("status_gizi", "")
70     t = row.get("tinggi_cm", np.nan)
71     b = row.get("berat_kg", np.nan)
72     bmi = None
73     if pd.notna(t) and pd.notna(b) and t > 0:
74         bmi = b / ((t/100.0)**2)
75     if rp in ("kronis", "jantung", "asma", "diabetes", "disabilitas"): return 1
76     if sg in ("kurang", "stunting", "gizi buruk"): return 1
77     if (bmi is not None) and (bmi < 18.5): return 1
78     if (bmi is not None) and ((bmi < 17) or (bmi > 35)): return 1
79     return 0
80
81 # Apply label
82 dinsos["layak_subsid"] = dinsos.apply(label_dinsos, axis=1)
83 dukcapil["layak_subsid"] = dukcapil.apply(label_dukcapil, axis=1)
84 kemenkes["layak_subsid"] = kemenkes.apply(label_kemenkes, axis=1)
85
```

3. Preprocessing

```
187 # =====
188 # 3) Preprocess (one-hot + min-max scaling global)
189 # =====
190 cat_cols_all = ["kondisi_rumah", "status_pekerjaan", "status_pernikahan", "riwayat_penyakit", "status_gizi"]
191 num_cols_all = ["jumlah_tanggungan", "penghasilan", "umur", "tinggi_cm", "berat_kg"]
192
193 for df in (dinsos, dukcapil, kemenkes):
194     if ("tinggi_cm", "berat_kg").issubset(df.columns):
195         df["BMI"] = df["berat_kg"] / ((df["tinggi_cm"] / 100.0) ** 2)
196     else:
197         df["BMI"] = np.nan
198     if "BMI" not in num_cols_all:
199         num_cols_all.append("BMI")
200
201 def union_categories(series_list):
202     cats = set()
203     for s in series_list:
204         if s is not None:
205             for v in s.dropna().unique().tolist():
206                 if isinstance(v, str): cats.add(v)
207     return sorted(list(cats))
208
209 global_vocab = {
210     "kondisi_rumah": union_categories([dinsos.get("kondisi_rumah")]),
211     "status_pekerjaan": union_categories([dukcapil.get("status_pekerjaan")]),
212     "status_pernikahan": union_categories([dukcapil.get("status_pernikahan")]),
213     "riwayat_penyakit": union_categories([kemenkes.get("riwayat_penyakit")]),
214     "status_gizi": union_categories([kemenkes.get("status_gizi")]),
215 }
216
217 def encode_and_scale(df):
218     for c in num_cols_all:
219         if c not in df: df[c] = np.nan
220     for c in cat_cols_all:
221         if c not in df: df[c] = np.nan
222     oh_parts = []
223     for col, vocab in global_vocab.items():
224         for v in vocab:
225             oh_parts.append(("{}_{}".format(col, v), (df[col] == v).astype(float)))
226     oh_df = pd.DataFrame([na.values for na, a in oh_parts], index=df.index) if oh_parts else pd.DataFrame(index=df.index)
227     num_df = df[num_cols_all].copy()
228     for c in num_cols_all:
229         if not np.isfinite(num_df[c]).any(): num_df[c] = 0.0
230         else: num_df[c] = num_df[c].fillna(num_df[c].median())
231     X_raw = pd.concat([num_df, oh_df], axis=1)
232     y = df["layak_subsidir"].astype(int).values
233     return X_raw, y
234
235 X_dinsos_raw, y_dinsos = encode_and_scale(dinsos)
236 X_dukcapil_raw, y_dukcapil = encode_and_scale(dukcapil)
237 X_kemenkes_raw, y_kemenkes = encode_and_scale(kemenkes)
238
239 all_X = pd.concat([X_dinsos_raw, X_dukcapil_raw, X_kemenkes_raw], axis=0)
240 mins, maxs = all_X.min(axis=0), all_X.max(axis=0)
241 rng = (maxs - mins).replace(0, 1.0)
242 def scale_like_global(X): return (X - mins) / rng
243
244 X_dinsos = scale_like_global(X_dinsos_raw).fillna(0.0)
245 X_dukcapil = scale_like_global(X_dukcapil_raw).fillna(0.0)
246 X_kemenkes = scale_like_global(X_kemenkes_raw).fillna(0.0)
247
248 FEATURE_COLS = list(X_dinsos.columns)
```

4. Dataset per klien

```
249 # =====
250 # 4) Dataset per klien
251 # =====
252 LOCAL_EPOCHS = 1
253 BATCH_SIZE = 64
254 DP_NUM_MICROBATCHES = BATCH_SIZE
255
256 def to_ds(X, y, local_epochs, shuffle=True):
257     X = X.values.astype("float32")
258     y = y.astype("float32").reshape(-1, 1)
259     ds = tf.data.Dataset.from_tensor_slices((X, y))
260     if shuffle:
261         ds = ds.shuffle(buffer_size=min(len(y), 2848), reshuffle_each_iteration=True)
262     ds = ds.batch(BATCH_SIZE, drop_remainder=True).repeat(local_epochs).prefetch(tf.data.AUTOTUNE)
263     return ds
264
265 client_data = [(X_dinsos, y_dinsos), (X_dukcapil, y_dukcapil), (X_kemenkes, y_kemenkes)]
266 client_sizes = [len(y_dinsos), len(y_dukcapil), len(y_kemenkes)]
267
268 def make_client_datasets(local_epochs, shuffle=True):
269     return [to_ds(X, y, local_epochs, shuffle) for (X, y) in client_data]
```

5. Model

```
270 # =====
271 # 5) Model
272 # =====
273
274 def build_model(input_dim):
275     i = tf.keras.Input(shape=(input_dim,))
276     x = tf.keras.layers.Dense(64, activation="relu")(i)
277     x = tf.keras.layers.Dense(32, activation="relu")(x)
278     o = tf.keras.layers.Dense(1, activation="sigmoid")(x)
279     return tf.keras.Model(i, o)
280
281 def get_weights(model): return [w.copy() for w in model.get_weights()]
282 def set_weights(model, weights): model.set_weights([w.copy() for w in weights])
283
284 def evaluate_global(model):
285     X_all = np.vstack([X_dinsos.values, X_dukcapil.values, X_kemenkes.values]).astype("float32")
286     y_all = np.concatenate([y_dinsos, y_dukcapil, y_kemenkes]).astype("float32").reshape(-1, 1)
287     loss = tf.keras.losses.BinaryCrossentropy(from_logits=False)
288     y_pred = model.predict(X_all, batch_size=256, verbose=0)
289     l = float(loss(y_all, y_pred).numpy())
290     acc = float(np.mean((y_pred >= 0.5) == y_all))
291     return l, acc
```

6. Federated learning

```
193 # -----
194 # 6) Federated Training Loop (funct1)
195 # -----
196 def clip_by_l2_norm(tensors, clip):
197     s = 0.0
198     for t in tensors: s += np.sum(np.square(t))
199     norm = float(np.sqrt(s)) + 1e-12
200     if norm <= clip: return tensors, 1.0
201     factor = clip / norm
202     return [t * factor for t in tensors], factor
203
204 def federated_train(noise_multiplier, rounds=5):
205     DP_CLIENT_LR = 0.05
206     DP_L2_NORM_CLIP_CLIENT = 1.5
207     SERVER_CLIP = 5.0
208     SERVER_LR = 1.0
209
210     tf.keras.utils.set_random_seed(42)
211     np.random.seed(42)
212
213     global_model = build_model(len(FEATURE_COLS))
214     global_model.compile(optimizer="sgd", loss="binary_crossentropy", metrics=["accuracy"])
215     global_weights = get_weights(global_model)
216
217     acc_log, loss_log = [], []
218
219     for round_idx in range(1, rounds+1):
220         client_ds = make_client_datasets(local_epochs=LOCAL_EPOCHS, shuffle=True)
221         clipped_weighted_sums = None
222         total_weight = 0.0
223
224         for k, ds in enumerate(client_ds):
225             local_model = build_model(len(FEATURE_COLS))
226             set_weights(local_model, global_weights)
227             opt = DPKerasSGDOptimizer(
228                 l2_norm_clip=DP_L2_NORM_CLIP_CLIENT,
229                 noise_multiplier=noise_multiplier,
230                 num_microbatches=BATCH_SIZE,
231                 learning_rate=DP_CLIENT_LR,
232                 momentum=0.9
233             )
234             loss = tf.keras.losses.BinaryCrossentropy(from_logits=False, reduction=tf.keras.losses.Reduction.NONE)
235             local_model.compile(optimizer=opt, loss=loss, metrics=["accuracy"])
236             local_model.fit(ds, epochs=1, verbose=0)
237
238             w_local = local_model.get_weights()
239             delta = [w1 - w2 for w1, w2 in zip(w_local, global_weights)]
240             delta_clipped, _ = clip_by_l2_norm(delta, SERVER_CLIP)
241
242             weight_k = float(client_sizes[k])
243             if clipped_weighted_sums is None:
244                 clipped_weighted_sums = [d * weight_k for d in delta_clipped]
245             else:
246                 for i in range(len(delta_clipped)):
247                     clipped_weighted_sums[i] += delta_clipped[i] * weight_k
248             total_weight += weight_k
249
250         avg_delta = [cw / (total_weight+1e-12) for cw in clipped_weighted_sums]
251         avg_delta = [SERVER_LR * d for d in avg_delta]
252         global_weights = [wg + d for wg, d in zip(global_weights, avg_delta)]
253         set_weights(global_model, global_weights)
254
255         gl_loss, gl_acc = evaluate_global(global_model)
256         acc_log.append(gl_acc)
257         loss_log.append(gl_loss)
258         print(f"[noise_multiplier:.3f] Round {round_idx} | acc={gl_acc:.4f} | loss={gl_loss:.4f}")
259
260     try:
261         n = max(client_sizes)
262         eps, _ = compute_dp_sgd_privacy_lib.compute_dp_sgd_privacy(
263             n=n,
264             batch_size=BATCH_SIZE,
265             noise_multiplier=noise_multiplier,
266             epochs=LOCAL_EPOCHS,
267             delta=1e-5
268         )
269     except Exception:
270         eps = np.nan
271
272     return np.mean(acc_log), np.mean(loss_log), eps, acc_log, loss_log
273
```

7. Differential Privacy

```
local_model = build_model(len(FEATURE_COLS))
set_weights(local_model, global_weights)
opt = DPKerasSGDOptimizer(
    l2_norm_clip=DP_L2_NORM_CLIP_CLIENT,
    noise_multiplier=noise_multiplier,
    num_microbatches=BATCH_SIZE,
    learning_rate=DP_CLIENT_LR,
    momentum=0.9
)

try:
    n = max(client_sizes)
    eps, _ = compute_dp_sgd_privacy_lib.compute_dp_sgd_privacy(
        n=n,
        batch_size=BATCH_SIZE,
        noise_multiplier=noise_multiplier,
        epochs=LOCAL_EPOCHS,
        delta=1e-5
    )
```

8. Evaluasi

```
274 # -----
275 # 7) Evaluasi Trade-off
276 # -----
277 noise_list = [0.01, 0.05, 0.1, 0.5, 1.0]
278 results = []
279 rounds = 5
280
281 for nm in noise_list:
282     acc, loss, eps, acc_log, loss_log = federated_train(noise_multiplier=nm, rounds=rounds)
283     results.append({"NoiseMultiplier": nm, "Accuracy": acc, "Loss": loss, "Epsilon": eps})
284
285 df_results = pd.DataFrame(results)
286 print("\nHasil Evaluasi Trade-off:")
287 print(df_results)
```

9. Plot Trade-off

```
288 # -----
289 # 8) Plot Trade-off (Accuracy vs Epsilon)
290 # -----
291 # -----
292 plt.figure(figsize=(7,5))
293 sns.lineplot(data=df_results, x="Epsilon", y="Accuracy", marker="o")
294 for i, row in df_results.iterrows():
295     plt.text(row["Epsilon"], row["Accuracy"], f"nm={row['NoiseMultiplier']}", fontsize=9)
296
297 plt.xlabel("Epsilon (ε) → Privasi ↓")
298 plt.ylabel("Accuracy → Utility ↑")
299 plt.title("Trade-off Federated Learning (DP-SGD)")
300 plt.grid(True)
301 plt.tight_layout()
302 plt.savefig("tradeoff_fl_accuracy_epsilon.png", dpi=150)
303 print("Grafik trade-off disimpan sebagai tradeoff_fl_accuracy_epsilon.png")
304
```

Pipeline end-to-end :

1. Dataset (load data)

Tahap pertama Adalah mengambil data mentah dari tiga client :

- Dinsos → berisi informasi jumlah_tanggungan, penghasilan, kondisi rumah
- Dukcapil → berisi informasi umur, status_pekerjaan, status_pernikahan
- Kemenkes → berisi informasi Riwayat_penyakit, status_gizi, tinggi, berat

Data ini di-load dari file CSV menggunakan pandas.read_csv(). Jika file tidak ada, program berhenti agar reproducibility terjaga.

2. Labeling

Data mentah tidak memiliki target langsung. Maka dibuat aturan untuk menentukan label layak_subsidy (1 = layak, 0 = tidak layak).

- Output tahap ini : kolom baru layak_subsidy pada tiap dataset

3. Pre-processing

Agar model dapat dilatih :

- Numerik : nilai yang hilang diisi mendian, lalu di-*scale* pakai min-max normalisasi global
- Kategori : diubah menjadi one-hot encoding berdasarkan vocab gabungan (supaya konsisten antar klien)
- Fitur Tambahan : BMI dihitung dari tinggi & berat badan

Output : matriks fitur numerik dan vector label yang sudah bersih

4. Dataset per Klien

Setiap sumber data dianggap sebagai client dalam Federated Learning.

- Data tiap klien dikonversi ke tf.data.Dataset kemudian dilakukan batching, shuffling, repeat sesuai dengan local epochs
- Tiga dataset klien siap dipakai dalam loop training

Hal ini mensimulasikan kondisi nyata Dimana data tidak dikumpulllkan disatu tempat, melainkan dilatih secara terdistribusi

5. Model

Menggunakan arsitektur :

- Input layer sesuai jumlah fitur
- Hidden layer : 64 neuron (ReLU) → 32 neuron (ReLU)
- Output layer : 1 neuron sigmoid (binary classification)

Penerapan model yang cukup ringan sehingga cocok untuk melakukan federated learning.

6. Federated Learning

Implementasi manual Federated Averaging (FedAvg) :

- Tiap klien dimulai dari bobot global
- Klien melatih model local pada datasetnya dengan optimizer DP-SGD
- Delta bobot hasil training di-clip dengan L2 norm agar tidak terlalu besar

- Server melakukan agregasi delta bobot dari semua klien (weighted average berdasarkan ukuran dataset)
- Update bobot global → looping untuk beberapa round

7. Differential privacy

Untuk menjamin privasi data tiap klien, maka digunakan :

- DP-SGD Optimizer → menambahkan *noise Gaussian* ke gradien, plus clipping gradien
- Menggunakan parameter penting
 - o L2_norm_clip → batas clipping gradien
 - o Noise_multiplier → besar noise yang ditambahkan
- Estimasi privasi dihitung dengan menggunakan `compute_dp_sgd_privacy` → menghasilkan nilai ϵ (epsilon), semakin kecil artinya privasi lebih kuat

8. Evaluasi

Setelah setiap federated round :

- Model global dievaluasi pada gabungan semua data klien
- Metrics : accuracy dan loss
- Disimpan log per round agar bisa dianalisis konsistensinya
- Dicatat pula nilai rata – rata akurasi/loss untuk tiap noise multiplier

9. Plot Trade-off

Hasil evaluasi divisualisasikan :

- Grafik Accuracy vs Epsilon (ϵ) → menunjukkan trade-off antara utility(akurasi) dan privacy (ϵ).
- Tiap titik diberi label nm = noise_multiplier
- Grafik disimpan dalam file PNG agar reproducible

MENGGUNAKAN TFF

0. Load data

```
18 # =====
19 # 0) Load CSV
20 # =====
21 def load_or_fail(path):
22     p = Path(path)
23     if not p.exists():
24         raise FileNotFoundError(f"File tidak ditemukan: {path}")
25     return pd.read_csv(p)
26
27 dinsos = load_or_fail("dinsos.csv")
28 dukcapil = load_or_fail("dukcapil.csv")
29 kemenkes = load_or_fail("kemenkes.csv")
30
```

1. Aturan Labeling

```
# =====
# 1) Aturan Labeling
# =====
def label_dinsos(row):
    jt = row.get("jumlah_tanggungan", np.nan)
    ph = row.get("penghasilan", np.nan)
    kr = row.get("kondisi_rumah", "")
    if pd.isna(jt): jt = 0
    if pd.isna(ph): ph = 0
    if ph < 2_000_000: return 1
    if (2_000_000 <= ph < 3_500_000) and jt >= 4: return 1
    if (kr in {"tidak layak", "semi permanen", "sangat sederhana"}) and (ph < 5_000_000): return 1
    if (jt >= 5) and (ph < 5_000_000): return 1
    if ph >= 5_000_000: return 0
    return 0

def label_dukcapil(row):
    u = row.get("umur", np.nan)
    sp = row.get("status_pekerjaan", "")
    st = row.get("status_pernikahan", "")
    if pd.isna(u): u = 0
    if u > 65: return 1
    if sp in {"pengangguran", "buruh", "pekerja informal"}: return 1
    if (sp=="wirausaha") and (u>55): return 1
    if (st in {"cerai", "janda", "duda"}) and (spi=="pegawai tetap"): return 1
    if (sp=="pegawai tetap") and (25<u<55) and (st=="menikah"): return 0
    if (u<25) and (sp=="wirausaha"): return 0
    return 0

def label_kemenkes(row):
    rp = row.get("riwayat_penyakit", "")
    sg = row.get("status_gizi", "")
    t = row.get("tinggi_cm", np.nan)
    b = row.get("berat_kg", np.nan)
    bmi = None
    if pd.notna(t) and pd.notna(b) and t > 0:
        bmi = b / ((t/100.0) ** 2)
    if rp in {"kronis", "jantung", "asma", "diabetes", "disabilitas"}: return 1
    if sg in {"kurang", "stunting", "gizi buruk"}: return 1
    if (bmi is not None) and (bmi < 18.5): return 1
    if (bmi is not None) and ((bmi < 17) or (bmi > 35)): return 1
    return 0

dinsos["layak_subsidy"] = dinsos.apply(label_dinsos, axis=1)
dukcapil["layak_subsidy"] = dukcapil.apply(label_dukcapil, axis=1)
kemenkes["layak_subsidy"] = kemenkes.apply(label_kemenkes, axis=1)
```

2. Pre-processing

```
78 # =====
79 # 2) Preprocessing (one-hot + min-max global)
80 # =====
81 cat_cols_all = ["kondisi_rumah", "status_pekerjaan", "status_pernikahan", "riwayat_penyakit", "status_gizi"]
82 num_cols_all = ["jumlah_tanggungan", "penghasilan", "umur", "tinggi_cm", "berat_kg"]
83
84 for df in (dinsos, dukcapil, kemenkes):
85     if ("tinggi_cm", "berat_kg").issubset(df.columns):
86         df["BMI"] = df["berat_kg"] / ((df["tinggi_cm"] / 100.0) ** 2)
87     else:
88         df["BMI"] = np.nan
89     if "BMI" not in num_cols_all:
90         num_cols_all.append("BMI")
91
92 def union_categories(series_list):
93     cats = set()
94     for s in series_list:
95         if s is not None:
96             for v in s.dropna().unique().tolist():
97                 if isinstance(v, str): cats.add(v)
98     return sorted(list(cats))
99
100 global_vocab = {
101     "kondisi_rumah": union_categories([dinsos.get("kondisi_rumah")]),
102     "status_pekerjaan": union_categories([dukcapil.get("status_pekerjaan")]),
103     "status_pernikahan": union_categories([dukcapil.get("status_pernikahan")]),
104     "riwayat_penyakit": union_categories([kemenkes.get("riwayat_penyakit")]),
105     "status_gizi": union_categories([kemenkes.get("status_gizi")]),
106 }
107
108 def encode_and_scale(df: pd.DataFrame):
109     for c in num_cols_all:
110         if c not in df: df[c] = np.nan
111     for c in cat_cols_all:
112         if c not in df: df[c] = np.nan
113
114     oh_parts = []
115     for col, vocab in global_vocab.items():
116         for v in vocab:
117             oh_parts.append((f"{col}_{v}", (df[col] == v).astype(float)))
118     oh_df = pd.DataFrame([n:a.values for n,a in oh_parts], index=df.index)
119
120     num_df = df[num_cols_all].copy()
121     for c in num_cols_all:
122         if not np.isfinite(num_df[c]).any(): num_df[c] = 0.0
123         else: num_df[c] = num_df[c].fillna(num_df[c].median())
124
125     X_raw = pd.concat([num_df, oh_df], axis=1)
126     y = df["layak_subsidir"].astype(int).values
127     return X_raw, y
128
129 X_dinsos_raw, y_dinsos = encode_and_scale(dinsos)
130 X_dukcapil_raw, y_dukcapil = encode_and_scale(dukcapil)
131 X_kemenkes_raw, y_kemenkes = encode_and_scale(kemenkes)
132
133 all_X = pd.concat([X_dinsos_raw, X_dukcapil_raw, X_kemenkes_raw], axis=0)
134 mins, maxs = all_X.min(axis=0), all_X.max(axis=0)
135 rng = (maxs - mins).replace(0, 1.0)
136 def scale_like_global(x): return (x - mins) / rng
137
138 X_dinsos = scale_like_global(X_dinsos_raw).fillna(0.0)
139 X_dukcapil = scale_like_global(X_dukcapil_raw).fillna(0.0)
140 X_kemenkes = scale_like_global(X_kemenkes_raw).fillna(0.0)
141
142 FEATURE_COLS = list(X_dinsos.columns)
```

3. Datasets

```
8 # =====
9 # 3) Dataset Federated untuk TFF
10 # =====
11
12 def to_tf_dataset(X, y, batch_size=64):
13     Xf = X.astype("float32").values
14     yf = y.astype("float32").reshape(-1,1)
15     return tf.data.Dataset.from_tensor_slices((Xf, yf)).batch(batch_size)
16
17 clients = [
18     ("dinsos", to_tf_dataset(X_dinsos, y_dinsos, len(y_dinsos))),
19     ("dukcapil", to_tf_dataset(X_dukcapil, y_dukcapil, len(y_dukcapil))),
20     ("kemenkes", to_tf_dataset(X_kemenkes, y_kemenkes, len(y_kemenkes))),
21 ]
22 total_clients = len(clients)
```

4. Model TFF

```
159 # =====
160 # 4) Model TFF
161 # =====
162
163 def model_fn():
164     model = tf.keras.Sequential([
165         tf.keras.layers.Input(shape=(len(FEATURE_COLS),)),
166         tf.keras.layers.Dense(64, activation="relu"),
167         tf.keras.layers.Dense(32, activation="relu"),
168         tf.keras.layers.Dense(1, activation="sigmoid"),
169     ])
170     return tff.learning.models.from_keras_model(
171         keras_model=model,
172         input_spec=clients[0][1].element_spec,
173         loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),
174         metrics=[tf.keras.metrics.BinaryAccuracy(name="binary_accuracy")],
175     )
```


5. DP Aggregator

```
175
176 # =====
177 # 5) Server-level DP Aggregator
178 # =====
179 from tensorflow_privacy.privacy.dp_query import gaussian_query
180 from tensorflow_federated.python.aggregators import differential_privacy
181
182 def build_process_with_server_dp(noise_multiplier, clients_per_round):
183     # Best GaussianSumQuery (differential TF-Privacy 0.9.0)
184     dp_query = gaussian_query.GaussianSumQuery(
185         ll_norm_clip=1.0,
186         stddev=noise_multiplier
187     )
188
189     # Bungkus query jadi factory (TF 0.87.0 API)
190     dp_factory = differential_privacy.DifferentiallyPrivateFactory(dp_query)
191
192     return tff.learning.algorithms.build_unweighted_fed_avg(
193         model_fn=model_fn,
194         client_optimizer_fn=tff.learning.optimizers.build_sgdm(
195             learning_rate=0.05, momentum=0.9),
196         server_optimizer_fn=tff.learning.optimizers.build_sgdm(
197             learning_rate=1.0, momentum=0.9),
198         model_aggregator=dp_factory
199     )
200
201
```

6. Training

```
202
203 # =====
204 # 6) Training loop (perbaiki sampling client)
205 # =====
206 def run_experiment(noise_multiplier, rounds=10, clients_per_round=None):
207     if clients_per_round is None:
208         clients_per_round = total_clients
209
210     learning_process = build_process_with_server_dp(noise_multiplier, clients_per_round)
211     state = learning_process.initialize()
212
213     logs = []
214     for e in range(1, rounds+1):
215         # ambil random sample fix sejumlah clients per round
216         idx = np.random.choice(range(total_clients), size=clients_per_round, replace=False)
217         sampled = [clients[i][1] for i in idx]
218
219         state, metrics = learning_process.next(state, sampled)
220
221         train_acc = float(metrics['client_work']['train']['binary_accuracy'])
222         train_loss = float(metrics['client_work']['train']['loss'])
223
224         print(f"[noise_multiplier: {noise_multiplier}] Round {e:02d} "
225               f"train_acc={train_acc:.4f} | train_loss={train_loss:.4f}")
226
227         logs.append({
228             "Round": e,
229             "NoiseMultiplier": noise_multiplier,
230             "train_binary_accuracy": train_acc,
231             "train_loss": train_loss,
232         })
233     return pd.DataFrame(logs)
234
```

7. Hasil

```
235
236 # =====
237 # 7) Jalankan semua noise
238 # =====
239 NOISE_LIST = [0.01, 0.05, 0.1, 0.5, 1.0]
240 ROUNDS = 5
241 all_logs = []
242
243 for nm in NOISE_LIST:
244     df_log = run_experiment(
245         noise_multiplier=nm,
246         rounds=ROUNDS,
247         clients_per_round=total_clients # semua client ikut
248     )
249     all_logs.append(df_log)
250
251 df_log_all = pd.concat(all_logs, ignore_index=True)
252
253 print("\n=== Rata-rata Akurasi per NoiseMultiplier ===")
254 avg_acc = df_log_all.groupby("NoiseMultiplier")["train_binary_accuracy"].mean().reset_index()
255 print(avg_acc)
256
```

8. Plot

```
256
257 # =====
258 # 8) Plot
259 # =====
260 plt.figure(figsize=(9,5))
261 sns.lineplot(data=df_log_all, x="Round", y="train_binary_accuracy", hue="NoiseMultiplier", marker="o")
262 plt.ylabel("Train Accuracy")
263 plt.title("Federated Learning (TFF 8.87.0 + Server DP Aggregator)")
264 plt.grid(True)
265 plt.tight_layout()
266 plt.savefig("tff_serverdp_accuracy.png", dpi=150)
267 print("Grafik disisipkan: tff_serverdp_accuracy.png")
268
```

1. Load Dataset

- Membaca tiga dataset dari CSV (dinsos.csv, dukcapil.csv, kemenkes.csv).
- Fungsi `load_or_fail` dipakai agar jika file tidak ada, langsung error (`FileNotFoundError`).
- Dataset ini jadi sumber data mentah untuk setiap *client* federated learning.

2. Aturan Labeling

- Menentukan label target (`layak_subsidisi`) berdasarkan logika aturan sosial/ekonomi/kesehatan.
- Ada tiga fungsi khusus:
 - o `label_dinsos`: berdasar penghasilan, jumlah tanggungan, kondisi rumah.
 - o `label_dukcapil`: berdasar umur, status pekerjaan, status pernikahan.
 - o `label_kemenkes`: berdasar riwayat penyakit, status gizi, serta perhitungan BMI.
- tiap dataset punya kolom biner `layak_subsidisi` (0 = tidak, 1 = ya).

3. Preprocessing (Encoding + Scaling)

- Menyatukan fitur numerik & kategorikal agar bisa dipakai model.
- Langkah:
 1. Numerik: (`jumlah_tanggungan`, `penghasilan`, `umur`, `tinggi_cm`, `berat_kg`, BMI) → normalisasi dengan min-max global (pakai gabungan semua dataset supaya skala seragam).
 2. Kategorikal: (`kondisi_rumah`, `status_pekerjaan`, dll.) → *di-one-hot encoding* berdasarkan vocab global (union dari semua kategori di semua dataset).
- Output: X (fitur terstandarisasi) dan y (label biner).

4. Dataset Federated

- Ubah data tiap institusi jadi format `tf.data.Dataset` agar kompatibel dengan TFF.
- Fungsi `to_tf_dataset` → menghasilkan dataset batched (X, y).
- Dikelompokkan sebagai *client*:
 - o Client 1: `dinsos`
 - o Client 2: `dukcapil`
 - o Client 3: `kemenkes`

5. Model TFF

- Definisi model ML yang akan dilatih secara federated.
- Model = Keras Sequential:
 - o Dense(64, relu)
 - o Dense(32, relu)
 - o Dense(1, sigmoid)
- Dibungkus dengan `tff.learning.models.from_keras_model` agar bisa dipakai di federated averaging.

6. Server-level DP Aggregator

- Menjamin privasi pada tahap agregasi model di server.
- Pakai GaussianSumQuery dari tensorflow-privacy → tambahkan *noise* Gaussian pada update agregasi.
- Dimasukkan ke TFF dengan DifferentiallyPrivateFactory.
- Hasilnya dipakai dalam `tff.learning.algorithms.build_unweighted_fed_avg`.

7. Training Loop

- Melatih model federated dengan sampling client dan logging metrik.
- Tahapan tiap ronde:
 1. Pilih subset client (random sampling).
 2. Panggil `learning_process.next(state, sampled)` untuk update global.
 3. Catat train accuracy dan loss tiap ronde.
- Bisa diulang dengan berbagai nilai noise multiplier (nm) untuk membandingkan trade-off.

8. Evaluasi & Plot

- Analisis hasil percobaan.
- Setelah training untuk semua nm selesai:
 - o Hitung rata-rata akurasi tiap noise multiplier.
 - o Plot kurva akurasi per ronde (dengan seaborn lineplot).

PERBANDINGAN MANUAL FEDERATED DAN TFF

1. Framework yang dipakai

- CODE 1
 - Manual Federated Learning (pure NumPy + Keras), loop federasi dibuat manual, tidak pakai TFF.
 - DP dilakukan di client-side dengan DPKerasSGDOptimizer (DP-SGD di level optimizer tiap klien).
- CODE 2
 - TensorFlow Federated (TFF 0.87.0) dipakai untuk membangun algoritme federated (build_unweighted_fed_avg).
 - DP dilakukan di server-side dengan DifferentiallyPrivateFactory(GaussianSumQuery) (DP Aggregator).

2. Posisi Differential Privacy

- CODE 1 → Client-level DP-SGD
 - Noise ditambahkan saat update gradien di setiap klien.
 - Gunakan DPKerasSGDOptimizer, l2_norm_clip, noise_multiplier.
 - Ada perhitungan epsilon dengan compute_dp_sgd_privacy_lib.
- CODE 2 → Server-level DP Aggregator
 - Noise ditambahkan saat agregasi model update di server.
 - Gunakan GaussianSumQuery + DifferentiallyPrivateFactory.
 - Tidak ada perhitungan epsilon eksplisit, hanya log akurasi/loss.

3. Training Loop

- CODE 1 (manual)
 - Loop round federasi ditulis sendiri (for round_idx in range(rounds)).
 - Model global diupdate manual (delta, clipping, averaging).
 - Harus handle federated averaging secara eksplisit.
- CODE 2 (TFF)
 - Training loop lebih ringkas, karena learning_process = build_unweighted_fed_avg(...) sudah mengatur federated averaging.
 - Hanya perlu memanggil state, metrics = learning_process.next(state, sampled).

4. Evaluasi Trade-off

- CODE 1
 - Setelah training, dihitung ϵ (epsilon) via `compute_dp_sgd_privacy_lib`.
 - Plot: Accuracy vs Epsilon (langsung lihat trade-off privasi vs utilitas).
- CODE 2
 - Hanya menghitung rata-rata akurasi per noise multiplier.
 - Plot: Accuracy vs Round untuk tiap NoiseMultiplier.
 - Tidak ada epsilon.

5. Output

- CODE 1 menghasilkan:
 - DataFrame dengan kolom: NoiseMultiplier, Accuracy, Loss, Epsilon.
 - Grafik: *tradeoff_fl_accuracy_epsilon.png* (Accuracy vs Epsilon).
- CODE 2 menghasilkan:
 - Rata-rata akurasi per NoiseMultiplier.
 - Grafik: *tff_serverdp_accuracy.png* (Accuracy vs Round, warna berdasarkan NoiseMultiplier).

Ringkasnya

- CODE 1 = manual FedAvg + DP-SGD di klien, ada perhitungan ϵ .
- CODE 2 = TFF FedAvg + DP Aggregator di server, hanya akurasi/loss.

TABEL PERBANDINGAN :

Aspek	Code 1	Code 2
Framework	Manual federated learning (NumPy + Keras)	TensorFlow Federated
Posisi Differential Privacy	Client-side DP-SGD → noise ditambahkan saat training klien	Server-side DP Aggregator → noise ditambahkan saat agregasi
DP Tools	DPKerasSGDOptimizer (optimizer dengan clipping + noise) + compute_dp_sgd_privacy_lib	GaussianSumQuery + DifferentiallyPrivateFactory
Clipping & Noise	L2 clipping di level gradien tiap klien	L2 clipping di level update global sebelum agregasi
Federated Averaging	Ditulis manual: delta weights dihitung & dirata-rata sendiri	Ditangani otomatis oleh tff.learning.algorithms.build_unweighted_fed_avg
Training Loop	Manual loop, update global model dilakukan sendiri	Gunakan learning_process.next(state, sampled)
Evaluasi	Hitung: Accuracy, Loss, dan ϵ (epsilon)	Hitung: Accuracy, Loss (tanpa epsilon)
Plot Trade-off	Accuracy vs Epsilon → menilai utility vs privacy	Accuracy vs Round (per NoiseMultiplier)