BAB 4
HASIL PENELITIAN

4.1 Ringkasan Hasil Penelitian

Penelitian ini dilakukan untuk meningkatkan keamanan pada jaringan *Internet of Things* (IoT) yang semakin luas digunakan dalam kehidupan sehari-hari. Dalam jaringan ini, banyak perangkat saling terhubung dan berpotensi menjadi target penyerangan. Oleh karena itu, diperlukan sistem yang mampu mendeteksi aktivitas tidak normal atau anomali.

Metode yang digunakan dalam penelitian ini meliputi penggunaan dataset *ACI IoT 2023* yang berisi data lalu lintas jaringan rumah pintar. Data diproses melalui tahap pembersihan, encoding label, pemilihan fitur penting, normalisasi, serta pembagian data menjadi data latih dan uji. Untuk mengatasi ketidakseimbangan data antara kelas mayoritas dan minoritas, digunakan teknik SMOTE. Model yang dibangun adalah model *stacking ensemble*, menggunakan algoritma *base learner* yaitu *Decision Tree, Random Forest, Naive Bayes* dan *Logistic Regression*, lalu prediksi dari keempat *base learner* digabungkan untuk dijadikan input untuk model *meta learner* yaitu SVM.

Hasil evaluasi menunjukkan bahwa model stacking ini memiliki akurasi sebesar 86.61% dan nilai ROC AUC sebesar 91.95%. Model ini terbukti cukup baik dalam mendeteksi berbagai jenis anomali, terutama pada data yang besar dan tidak seimbang. Namun, masih terdapat beberapa tantangan dalam mendeteksi kelas minoritas.

4.2 Pembahasan

4.2.1 Pengumpulan Dataset

Penelitian ini menggunakan dataset sekunder bernama *ACI IoT Network Traffic 2023* yang diunduh dari platform Kaggle. Dataset ini merupakan kumpulan *log traffic* jaringan yang merekam berbagai aktivitas komunikasi antar perangkat IoT selama periode lima hari (30 Oktober - 3 November 2023). Pengumpulan data

dilakukan oleh *Internet of Things Research Lab* (IoTRL) di *Army Cyber Institute* (ACI) dalam sebuah lingkungan simulasi rumah yang dirancang khusus untuk meniru kondisi jaringan IoT rumah tangga yang realistis.

Eksperimen pengambilan data dilakukan menggunakan berbagai perangkat yang terhubung secara *wired* dan *wireless*. Lingkungan yang beragam ini juga mencakup berbagai perangkat IoT dan perangkat jaringan reguler yang umum ditemukan pada lingkungan rumah. Untuk mengelola dan mengontrol perangkat IoT, laboratorium mengintegrasikan platform *Home Assistant* sebagai sistem kontrol terpusat. *Home Assistant* merupakan perangkat lunak *open-source* untuk otomasi rumah yang berfungsi mengatur interaksi dan automasi di antara berbagai perangkat dalam jaringan. Pengumpulan data dilakukan menggunakan *network sniffer* dan monitoring points yang ditempatkan secara strategis di seluruh laboratorium untuk menangkap trafik jaringan baik *wired* maupun *wireless*.

Pengumpulan data dilakukan melalui serangkaian serangan siber yang dieksekusi selama lima hari dengan skenario terstruktur yang ditunjukkan pada tabel berikut.

Tabel 4. 1 Skenario Serangan

Hari, Tanggal	Waktu	Jenis Aktivitas	Rincian Serangan
Senin, 30 Oktober 2023	09:01 – 09:05	Reconnaissance	Host Discovery
	09:06 – 09:09		Ping Sweep
	09:12 – 11:04		Ping Sweep
	11:06 – 12:09		OS Scan
	12:10 – 12:16		Vulnerability Scan
	12:20 – 15:56		Port Scan
Selasa, 31 Oktober 2023	09:00 – 16:05	Benign 1	Aktivitas otomatisasi rumah tangga (home assistant)
Rabu, 1 November 2023	09:30 – 09:55	DoS Attack	ICMP Flooding
	09:58 – 10:26		Slowloris
	10:28 – 10:53		SYN Flood

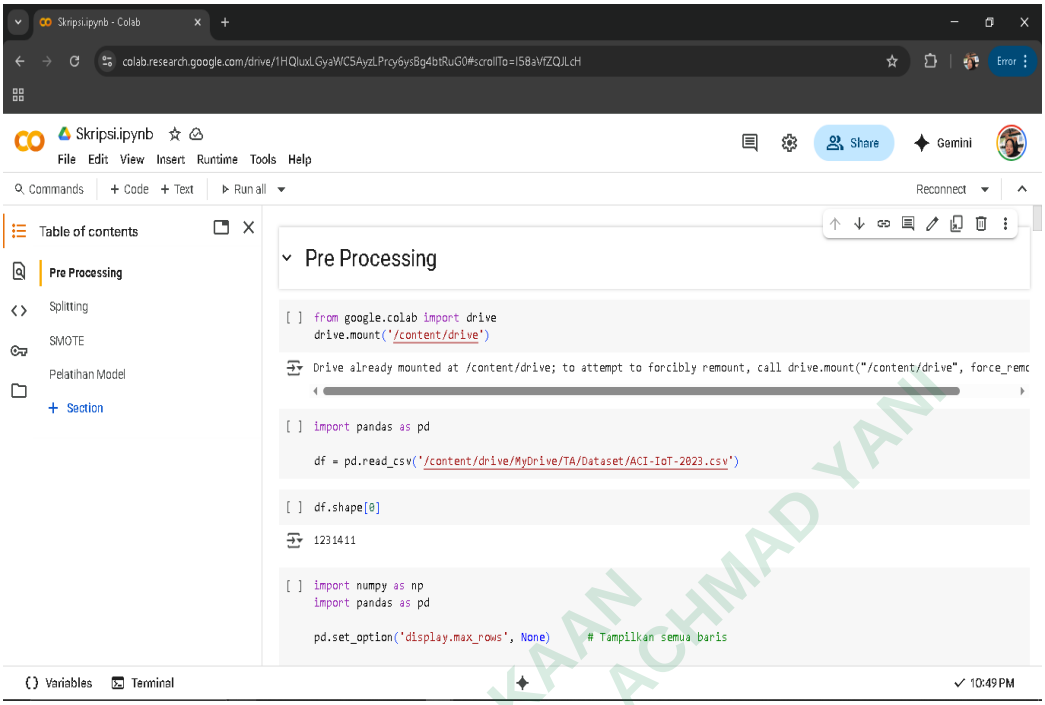
Hari, Tanggal	Waktu	Jenis Aktivitas	Rincian Serangan
	10:54 – 11:21		UDP Flood
	11:23 – 11:49		DNS Flood
Kamis, 2 November 2023	09:30 – 11:44	Brute Force & Spoofing	Dictionary Brute Force
	12:12 – 13:48		ARP Spoofing
Jumat, 3 November 2023	09:00 – 16:05	Benign 2	Aktivitas otomatisasi rumah tangga (home assistant)

Kategori serangan ini dipilih secara khusus untuk mencakup beragam ancaman yang umum ditemukan dalam skenario IoT dunia nyata. Dataset dilengkapi dengan *timesheet* yang memberikan informasi detail mengenai waktu pelaksanaan serangan dan automasi.

Dataset disajikan dalam format CSV dengan nama file *ACI-IoT-2023.csv*. Dataset ini menyediakan representasi data multi-modal yang mencakup pola trafik jaringan, komunikasi antar perangkat, dan karakteristik spesifik perangkat. Setiap baris dalam dataset mewakili satu sesi komunikasi (*network flow*) dan memuat 85 atribut, termasuk satu kolom label. Total terdapat 1.231.411 entri dalam dataset (Bastian dkk., 2023).

4.2.2 *Pre-processing* Data

P*re-processing* data merupakan tahapan yang penting untuk mempersiapkan dataset sebelum digunakan dalam proses pelatihan model. Dataset yang digunakan terdiri dari 1.231.411 entri dengan 85 kolom fitur yang mencakup atribut seperti alamat IP, protokol jaringan, durasi komunikasi, serta label trafik yang menunjukkan jenis aktivitas jaringan. Sebelumnya, seluruh proses *Pre-processing* data pada dataset *ACI IoT Network Traffic 2023* sampai dengan pelatihan dan evaluasi model dilakukan menggunakan *tools google collab* yang dapat ditunjukkan pada gambar 4.1 berikut.



Gambar 4. 1 Tampilan Google Collab

Adapun proses *preprocessing* dilakukan melalui beberapa tahapan sebagai berikut:

1. **Pemeriksaan Nilai Kosong dan Tak Terhingga**

   Langkah pertama dalam preprocessing adalah melakukan pemeriksaan awal untuk mendeteksi keberadaan nilai kosong (NaN) dan nilai tak terhingga (*infinity*) dalam dataset. Nilai *infinity* muncul karena adanya pembagian dengan nol atau eror komputasi, sedangkan NaN mengindikasikan data atau nilai yang hilang. Pemeriksaan dilakukan menggunakan kode berikut:

```
print("Jumlah nilai NaN per kolom:")
print(df.isnull().sum())

print("\nJumlah nilai Inf per kolom:")
print(np.isinf(df.select_dtypes(include=[np.number])).sum())
```

   Hasil pemeriksaan menunjukkan terdapat nilai NaN pada kolom Flow Bytes/s sebanyak 1.009 dan terdapat nilai inf pada kolom Flow Bytes/s sebanyak 915 dan kolom Flow Packets/s sebanyak 1.924 baris sehingga diperlukan penanganan sebagai berikut:

   - Mengubah nilai inf dan – inf menjadi Nan

```
df = df.replace([np.inf, -np.inf], np.nan)
```

- Menghapus baris yang mengandung nilai Nan

```
df = df.dropna()
```

2. Penanganan Duplikasi Data

Langkah kedua adalah mengidentifikasi dan menangani data duplikat. Sebelumnya data diperiksa terlebih dahulu menggunakan kode berikut:

```
df.duplicated().sum()
```

Hasil dari kode tersebut menunjukkan terdapat sebanyak 56.181 baris data duplikat pada dataset yang digunakan. Untuk menjaga kualitas data dan menghindari bias dalam analisis, seluruh baris data duplikat dihapus menggunakan kode berikut:

```
df = df.drop_duplicates()
```

3. Label Encoding

Langkah ketiga adalah melakukan encoding terhadap label kategorikal yang terdapat pada kolom label. Karena algoritma ML tidak dapat memproses data dalam bentuk teks secara langsung, maka label-label tersebut perlu dikonversi ke dalam bentuk numerik. Konversi dilakukan menggunakan metode *Label Encoding* dari *library scikit-learn* menggunakan kode berikut:

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df['Label'] = le.fit_transform(df['Label'])
```

Hasil dari proses encoding ditunjukkan pada tabel 4.2 berikut, yang berisi label asli, jumlah masing-masing label (*value count*), dan hasil konversi menjadi nilai numerik.

Tabel 4. 2 Hasil Label Encoding

Label Asli	Value Count	Label Encode
Port Scan	441260	7
Benign	307288	1
ICMP Flood	225234	4
DNS Flood	46934	2
Vulnerability Scan	39533	11
OS Scan	37524	5
Ping Sweep	35964	6
Slowloris	18537	9
SYN Flood	13857	8
Dictionary Attack	6379	3
UDP Flood	791	10
ARP Spoofing	5	0

4. Seleksi Fitur

Langkah keempat adalah melakukan seleksi fitur dari total 85 Fitur yang tersedia dalam dataset. Fitur dipilih secara manual berdasarkan relevansi terhadap tujuan penelitian. Dari 85 fitur yang tersedia, dipilih 4 fitur utama yang memiliki kontribusi dalam mendeteksi pola anomali pada trafik IoT. Pemilihan dilakukan berdasarkan literature dan pemahaman logika terhadap karakteristik serangan. Adapun fitur yang dipilih beserta alasannya ditunjukkan pada tabel 4.3 berikut:

Tabel 4. 3 Fitur Terpilih

No.	Nama Fitur	Deskripsi	Relevansi dengan Penelitian
1.	Flow Duration	Durasi komunikasi dalam satu flow (detik)	Serangan memiliki pola durasi yang berbeda dari trafik normal. Flow duration yang sangat pendek atau sangat panjang dapat mengindikasikan aktivitas mencurigakan
2.	Total Fwd	Jumlah paket yang	Pola pengiriman paket yang

No.	Nama Fitur	Deskripsi	Relevansi dengan Penelitian
	Packet	dikirim dari source ke destination	tidak wajar dapat menunjukkan adanya serangan
3.	Total Bwd packets	Jumlah paket yang dikirim dari destination ke source	Asimetri pengiriman sering terjadi dalam scenario serangan
4.	Flow Bytes/s	Kecepatan transfer data dalam bytes per detik	Throughput tinggi dapat menunjukkan serangan flooding attacks atau aktivitas transfer data yang mencurigakan

Keempat fitur ini dipilih karena memiliki relevansi dengan jenis serangan yang ada dalam dataset. Penggunaan fitur yang terbatas ini bertujuan untuk menguji efektivitas model dalam mendeteksi anomali dengan *complexity* yang rendah namun tetap relevan terhadap konteks deteksi anomali. Selanjutnya fitur label juga disimpan dalam kolom terpisah sebagai target untuk klasifikasi.

5. Normalisasi

Langkah keenam adalah melakukan normalisasi terhadap data fitur menggunakan metode *min-max scaling*. Normalisasi bertujuan untuk mengubah skala nilai fitur ke dalam rentan [0, 1] agar semua fitur memiliki bobot yang sebanding dan tidak mendominasi proses pelatihan model akibat perbedaan skala. Sebelum dilakukan normalisasi, fitur (x) dan label (y) dipisahkan menggunakan kode berikut:

```
X_features = x.drop('Label', axis=1)  # Semua kolom kecuali Label
y_label = x['Label']                  # Hanya kolom Label
```

Setelah itu proses normalisasi fitur dilakukan dengan *MinMaxScaler* dari *library scikit-learn* dengan kode sebagai berikut:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_normalized = scaler.fit_transform(X_features)
X_normalized_df = pd.DataFrame(X_normalized,
columns=X_features.columns
```

Setelah proses normalisasi dilakukan selanjutnya adalah menggabungkan kembali kolom label ke dalam data frame hasil normalisasi. Sehingga struktur akhir dataset tetap terdiri dari fitur untuk prediksi dan kolom label sebagai target klasifikasi. Berikut adalah kode yang digunakan:

```
X_normalized_df['Label'] = x['Label'].values
X_normalized_df.head()
```

Hasil dari kode tersebut menampilkan lima baris pertama dari dataset hasil normalisasi dengan format yang ditunjukkan pada gambar 4.2 berikut:

	Flow Duration	Total Fwd Packet	Total Bwd packets	Flow Bytes/s	Label
0	0.006234	0.000135	0.000152	1.366401e-05	1
1	0.003374	0.000027	0.000041	1.011432e-06	1
2	0.082538	0.000000	0.000028	8.988343e-10	1
3	0.001186	0.000027	0.000055	2.450140e-06	1
4	0.000040	0.000013	0.000000	0.000000e+00	1

Gambar 4. 2 Hasil Normalisasi

4.2.3 Pembagian Data

Tahapan selanjutnya adalah pembagian dataset menjadi data latih (*training)* dan data uji (*testing)*. Tujuannya adalah agar model dapat dilatih menggunakan sebagian data, lalu dievaluasi performanya terhadap data yang belum pernah dilihat sebelumnya, sehingga dapat mengukur kemampuan generalisasi model terhadap data baru. Pembagian data dilakukan dengan rasio 80% untuk data latih dan 20% untuk data uji menggunakan fungsi train_test_split dari *library scikit-learn*. Proses ini dilakukan dengan kode sebagai berikut:

```
from sklearn.model_selection import train_test_split

X = df.drop('Label', axis=1)
y = df['Label']

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)
```

Hasil dari pembagian data ini adalah sebagai berikut:

- Jumlah data latih (X_train, y_train) : 938.644 baris
- Jumlah data uji (X_test, y_test) : 234.662 baris
- Jumlah fitur (kolom) pada masing-masing subset data: 4 kolom

Pembagian ini untuk memastikan bahwa model mendapatkan cukup data untuk proses pelatihan, tanpa kehilangan kemampuan untuk diuji terhadap data yang belum pernah dilihat.

4.2.4 Penanganan Ketidakesimbangan Data

1. Analisis Distribusi Kelas

Sebelum masuk ke tahapan pelatihan data, dilakukan analisis awal terhadap distribusi kelas pada variabel target (Label) untuk mengetahui apakah terjadi ketidakseimbangan jumlah sampel antar kelas. Berikut adalah kode perintah yang digunakan untuk melihat distribusi kelas:

```
# Cek distribusi kelas sebelum SMOTE
print("Distribusi kelas sebelum SMOTE:")
print(pd.Series(y_train).value_counts().sort_index())
```

Hasil menunjukkan adanya ketidakseimbangan kelas yang cukup signifikan. Beberapa kelas seperti ARP Spoofing (kelas 0) dan UDP Flood (kelas 10) hanya memiliki masing-masing 4 dan 54 sampel, sedangkan kelas lainnya seperti Port Scan dan Benign memiliki puluhan ribu sampel. Ketidakseimbangan ini berpotensi menyebabkan model bias terhdapat kelas mayoritas, yang justru penting dalam konteks deteksi anomali. Berikut adalah distribusi kelas secara keseluruhan sebelum penanganan *imbalanced data* yang dapat dilihat pada tabel 4.4 berikut.

Tabel 4. 4 Distribusi Kelas Sebelum SMOTE

Kelas	Jumlah Sampel	Deskripsi
Kelas 0	4	ARP Spoofing
Kelas 1	245830	Benign
Kelas 2	37547	DNS Flood
Kelas 3	5103	Dictionary Attack

Kelas	Jumlah Sampel	Deskripsi
Kelas 4	180187	ICMP Flood
Kelas 5	30019	OS Scan
Kelas 6	28771	Ping Sweep
Kelas 7	353008	Port Scan
Kelas 8	11086	SYN Flood
Kelas 9	14830	Slowloris
Kelas 10	633	UDP Flood
Kelas 11	31626	Vulnerability Scan

2. Strategi Penanganan dan Penerapan SMOTE

Untuk mengatasi masalah keseimbangan data antar kelas, digunakan metode *Synthetic Minority Over-sampling Technique* (SMOTE) karena mampu menghasilkan sampel sintetis untuk kelas minoritas berdasarkan fitur terdekatnya. Namun, penerapan SMOTE secara menyeluruh terhadap seluruh kelas minoritas dengan menyemakan jumlah kelas mayoritas akan menyebabkan dataset membengkak secara drastis dan tidak efisien. Untuk itu diterapkan strategi sampling secara kustom dengan menetapkan target jumlah data yang disesuaikan per kelas, berdasarkan pertimbangan distribusi data. Berikut adalah strategi sampling yang digunakan:

```
sampling_strategy = {
    2: 50000,
    11: 50000,
    5: 50000,
    6: 50000,
    9: 50000,
    8: 50000,
    3: 20000,
    10: 2000,
    0: 2000
}
```

Strategi ini menargetkan kelas-kelas dengan jumlah data yang tergolong rendah atau sedang, untuk ditingkatkan hingga batas tertentu yang dianggap memadai untuk pembelajaran model

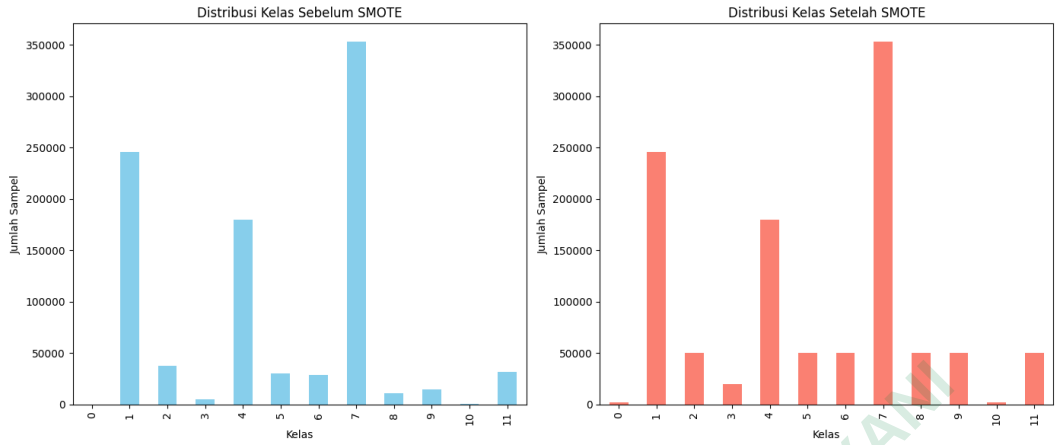
Selanjutnya menerapkan teknik SMOTE dengan parameter k_neighbors=1 agar tetap dapat menghasilkan sampel sintetis meskipun jumlah data asli sangat kecil. Berikut adalah kode perintah yang digunakan untuk menerapkan SMOTE:

```
# SMOTE dengan sampling_strategy custom dan k_neighbors rendah
(karena kelas minoritas sedikit)
smote = SMOTE(sampling_strategy=sampling_strategy,
random_state=42, k_neighbors=1)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train,
y_train)
```

Hasil penerapan SMOTE menunjukkan distribusi kelas menjadi lebih proporsional, terutama untuk kelas-kelas minoritas yang sebelumnya sangat sedikit. Meskipun belum sepenuhnya seimbang dengan kelas mayoritas, penambahan jumlah sampel menggunakan strategi sampling kustom berhasil mengurangi ketidakseimbangan. Distribusi kelas setelah SMOTE dapat dilihat pada tabel 4.5 berikut beserta visualisasinya.

Tabel 4. 5 Distribusi Kelas Setelah SMOTE

Kelas	Jumlah Sampel	Perubahan
Kelas 0	2.000	Ditambah
Kelas 1	245.830	Tetap
Kelas 2	50.000	Ditambah
Kelas 3	20.000	Ditambah
Kelas 4	180.187	Tetap
Kelas 5	50.000	Ditambah
Kelas 6	50.000	Ditambah
Kelas 7	353.008	Tetap
Kelas 8	50.000	Ditambah
Kelas 9	50.000	Ditambah
Kelas 10	2.000	Ditambah
Kelas 11	50.000	Ditambah



Gambar 4. 3 Diagram Distribusi Kelas

Penerapan SMOTE berhasil menyeimbangkan kelas minoritas tanpa mengubah distribusi kelas mayoritas. Dataset hasil SMOTE memiliki total 1.103.025 baris dan siap digunakan untuk proses pelatihan model. Penerapan SMOTE hanya dilakukan pada data *training* saja dan data uji tetap menggunakan distribusi awal untuk memastikan evaluasi yang adil.

4.2.5 Pelatihan Model

Tahapan ini merupakan inti dari penelitian. Pada tahapan ini dilakukan pelatihan model *machine learning* untuk mendeteksi dan mengklasifikasikan trafik jaringan IoT ke dalam 12 kelas, termasuk trafik normal (Benign) dan berbagai jenis serangan atau anomali. Pelatihan model menggunakan pendekatan *ensemble stacking* dengan beberapa *base learner* dan satu *meta learner*. Model dilatih menggunakan data hasil dari oversampling menggunakan teknik SMOTE yaitu X_train_balanced dan y_train_balanced, yang sudah diolah dengan berbagai tahapan agar data representatif dan seimbang antar kelas. Berikut adalah beberapa tahapan dari proses pelatihan model:

1. **Menentukan Strategi Validasi: Stratified k-Fold**

   Sebelum memulai pelatihan, langkah pertama adalah menentukan strategi validasi data. Penelitian ini menggunakan teknik *Stratified k-Fold Cross Validation* dengan 5 fold. Berikut adalah kode perintah yang digunakan:

```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
classes = np.unique(y_train_balanced)
```

Penggunaan parameter shuffle=True digunakan untuk membantu mengacak data sebelum dibagi, dan random_state=42 digunakan untuk menjaga hasil yang konsisten.

2. Inisialisasi Base Learner

Pada tahapan pelatihan ini menggunakan empat model pembelajaran dasar (*base learner*) yaitu:

- *Decision Tree* (DT)
- *Random Forest* (RF)
- *Naïve Bayes* (NB)
- *Logistic Regression* (LR)

Keempat model dasar ini dilatih menggunakan data hasil *pre-processing* yang telah melalui tahap SMOTE untuk menyeimbangkan distribusi kelas. Berikut adalah kode yang digunakan untuk melakukan pelatihan terhadap keempat model dasar:

```
dt = DecisionTreeClassifier(max_depth=10, random_state=42)
rf = RandomForestClassifier(n_estimators=50, max_depth=10,
n_jobs=-1, random_state=42)
nb = GaussianNB()
lr = (LogisticRegression(max_iter=300, solver='saga', n_jobs=-1,
random_state=42, tol=1e-2)

models = {
    'DecisionTree': dt,
    'RandomForest': rf,
    'NaiveBayes': nb,
    'LogisticRegression': lr
}
```

Setiap model diinisialisasi dengan parameter tertentu untuk mengoptimalkan performa dan efisiensi. *Decision Tree* dan *Random Forest* dibatasi kedalaman pohonnya hingga 10 (max_depth=10) guna menghindari *overfitting*. *Random Forest* menggunakan 50 estimator (n_estimators=50) dan diparalelkan dengan n_jobs=-1. *Naive Bayes* digunakan tanpa parameter khusus karena bersifat ringan dan cepat.

Sementara itu, *Logistic Regression* menggunakan solver 'saga', batas iterasi sebanyak 300 (max_iter=300), dan toleransi konvergensi 1e-2 untuk mempercepat pelatihan pada dataset berukuran besar, serta dibungkus dalam skema *OneVsRestClassifier* agar dapat menangani klasifikasi multi-kelas.

3. Mengambil Prediksi dari *Base Learner*

Selanjutnya dalam metode *stacking*, prediksi dari *base learner* digunakan sebagai input untuk *meta learner*. Untuk menghindari kebocoran data dan *overfitting*, prediksi probabilitas diambil menggunakan metode *cross-validation out-of-fold* dengan fungsi cross_val_predict() dengan parameter cv=cv dan method='predict_proba'. Berikut adalah kode perintah yang digunakan untuk mengambil prediksi base learner:

```
# Ambil probabilitas prediksi dari base models
base_models = list(models.values())
base_probas_train = []

for model in base_models:
    proba = cross_val_predict(model, X_train_balanced,
y_train_balanced, cv=cv, method='predict_proba')
    base_probas_train.append(proba)

# Gabungkan menjadi fitur baru untuk meta-learner
X_train_meta_base = np.hstack(base_probas_train)
```

4. Reduksi Dimensi dengan PCA

Karena hasil gabungan prediksi dari keempat *base learner* menghasilkan data berdimensi tinggi (jumlah kelas x jumlah model), akan dilakukan reduksi dimensi menggunakan PCA (*Principal Component Analysis*) sebelum masuk ke *meta learner*. PCA ini bertujuan untuk mengurangi beban komputasi, menghindari *overfitting* dan menyimpan informasi terpenting dari prediksi. Berikut adalah kode yang digunakan:

```
pca = PCA(n_components=8)
X_meta_pca = pca.fit_transform(X_meta)
```

5. Pelatihan Meta Learner: SVM

Selanjutnya adalah pelatihan meta learner menggunakan *Linear Support Vector Machine* (LinearSVC). Tetapi, karena LinearSVC tidak mendukung prediksi probabilitas secara langsung, maka diperlukan *CalibratedClassifierCV* untuk menghasilkan nilai probabilitas yang dibutuhkan untuk perhitungan skor ROC AUC. Berikut kode yang digunakan:

```
from sklearn.svm import LinearSVC
from sklearn.calibration import CalibratedClassifierCV

# Siapkan meta-model: LinearSVC dengan kalibrasi agar bisa
predict_proba
base_svm = LinearSVC(max_iter=2000, random_state=42)
meta_model = CalibratedClassifierCV(base_svm, cv=3)

meta_model.fit(X_meta_pca, y_train_balanced)
```

4.2.6 Evaluasi Model

Setelah semua model berhasil dilatih, tahap berikutnya adalah melakukan evaluasi model untuk mengukur performa dan efisiensi komputasi dari masing-masing algoritma ML baik secara individu (base learner) maupun secara ensemble stacking. Evaluasi dilakukan menggunakan pendekatan *cross validation* yaitu *stratified k-fold* dengan 5 fold, untuk menjaga konsistensi dan menghindari *overfitting*. Metrik yang digunakan adalah *accuracy, precision, recall, F1-score*, dan ROC AUC, dengan pendekatan *macro average* untuk klasifikasi *multiclass*.

1. Evaluasi Base Learner

Evaluasi terhadap masing-masing *base* learner dilakukan menggunakan fungsi cross_val_predict() untuk mendapatkan prediksi dan probabilitas dari setiap fold. Metrix evaluasi yang digunakan yaitu *accuracy, precision, recall, F1-score* dan ROC AUC, dengan pendekatan *macro average* untuk klasifikasi *multiclass*. Berikut ada kode yang digunakan untuk mengambil hasil evaluasi dari keempat *base learner*:

```
    start = time.time()
    y_pred = cross_val_predict(model, X_train_balanced,
y_train_balanced, cv=cv)
    acc = accuracy_score(y_train_balanced, y_pred)
    prec = precision_score(y_train_balanced, y_pred,
average='macro', zero_division=0)
    rec = recall_score(y_train_balanced, y_pred, average='macro',
zero_division=0)
    f1 = f1_score(y_train_balanced, y_pred, average='macro',
zero_division=0)
    roc = roc_auc_score(y_bin, y_proba, average='macro',
multi_class='ovr')
        print(f"ROC AUC    : {roc:.4f}")
    end = time.time()
```

Berdasarkan kode diatas, berikut adalah hasil yang diperoleh dari evaluasi keempat base learner yang ditunjukkan pada tabel 4.6.

Tabel 4. 6 Tabel Evaluasi Matrix Base Learner

Model	Accuracy	Precision	Recall	F1-Score	ROC AUC	Waktu
Decision Tree	0.8200	0.7537	0.6140	0.6066	0.9506	66.68 Detik
Random Forest	0.8225	0.8325	0.6263	0.6257	0.9530	777.04 Detik
Naïve Bayes	0.2825	0.2826	0.3654	0.2094	0.7399	4.60 Detik
Logistic Regression	0.4074	0.0620	0.1161	0.0803	0.6992	139.01 Detik

Dari hasil tabel diatas, dapat dilihat model *Random Forest* menunjukkan performa terbaik diantara keempat *base learner*, dengan akurasi sebesar 82.25% dan ROC AUC sebesar 0.9530. Sedangkan model *Naive Bayes* dan *Logistic Regression* menunjukkan performa yang sangat rendah, dengan nilai F1-score dan ROC AUC yang jauh lebih kecil dibandingkan dengan model lainnya. Kedua model tersebut tetap akan digunakan dalam model *stacking* karena keberagaman model sangat penting dalam pendekatan *ensemble*. Meskipun performanya rendah secara individual, *Naive Bayes*

dan *Logistic Regression* dapat memberikan pola prediksi yang berbeda, apabila dikombinasikan untuk *meta learner* dapat memperkaya representasi dan meningkatkan generalisasi model secara keseluruhan.

2. Evaluasi Model Ensemble (Stacking dengan PCA)

Selanjutnya adalah evaluasi model stacking, yaitu dengan mengabungkan prediksi probabilitas dari keempat *base learner* dan menggunakannya sebagai input ke *meta learner* yaitu SVM (LinearSVC). Untuk mengurangi komputasi akibat banyaknya dimensi, dilakukan reduksi dimensi menggunakan PCA menjadi 8 komponen utama. LinearSVC kemudian dilatih menggunakan *CalibratedClasifierCv* untuk mengkalibrasi model klasifikasi agar dapat menghasilkan prediksi probabilitas yang diperlukan dalam evaluasi ROC AUC. Berikut adalah hasil evaluasi performa model stacking:

- *Accuracy:* 0.8662
- *Precision:* 0.5207
- *Recall:* 0.5212
- *F1-Score:* 0.5052
- *ROC AUC:* 0.9195

Nilai ROC AUC sebesar 0.9195 menunjukkan bahwa model memiliki kemampuan yang sangat baik dalam membedakan kelas-kelas anomali secara umum. Namun, nilai F1-Score-nya sedikit lebih rendah (0.505), karena precision dan recall-nya belum maksimal. Ini menunjukkan bahwa stacking sangat efektif untuk meningkatkan akurasi umum dan generalisasi, meskipun belum sempurna dalam mendeteksi semua kelas secara merata.

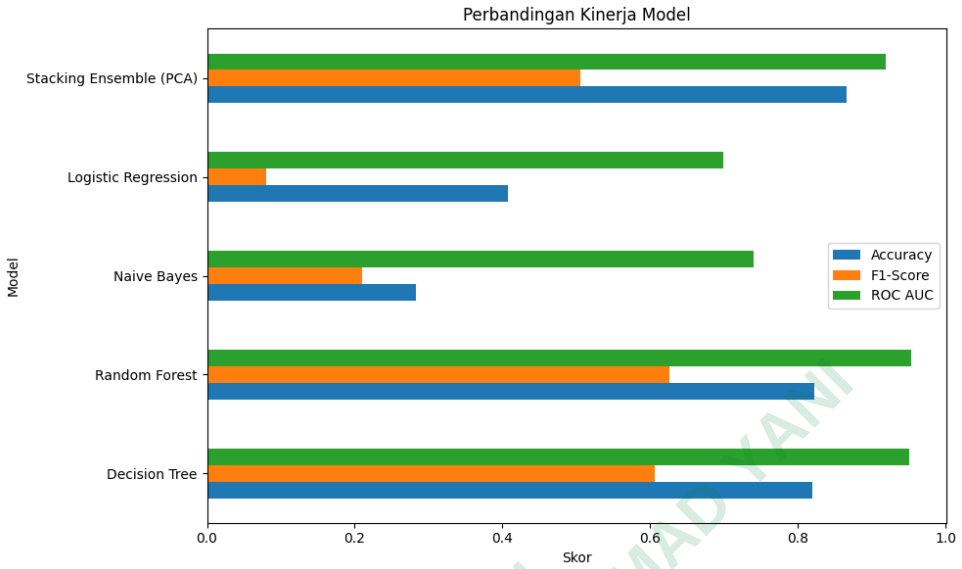
3. Perbandingan Kinerja Model Base Learner dan Ensemble

Berdasarkan hasil evaluasi pada model base learner dan model ensemble stacking, dilakukan analisis perbandingan performa untuk mengetahui kontribusi teknik ensemble terhadap peningkatan akurasi dan kemampuan deteksi anomali. Berikut adalah tabel dan visualisasi perbandingan antara model base learner dan ensemble stacking.

Tabel 4. 7 Perbandingan Base learner dan Ensemble Stacking

Model	Accuracy	Precision	Recall	F1-Score	ROC AUC	Waktu Latih
Decision Tree	82%	75.37%	61.4%	60.66%	95.06%	66.68 Detik
Random Forest	82.25%	83.25%	62.63%	62.57%	95.3%	777.04 Detik
Naive Bayes	28.25%	28.26%	36.54%	20.94%	73.99%	4.60 Detik
Logistic Regression	40.74%	6.2%	11.61%	8.03%	69.92%	139.01 Detik
Stacking Ensemble	**86.62%**	**52.07%**	**52.12%**	**50.52%**	**91.95%**	**219.19 Detik**

Dari hasil pengujian, model Stacking menunjukkan nilai akurasi tertinggi sebesar 86.62%, melampaui seluruh model individu seperti Random Forest (82.25%) dan Decision Tree (82%). Meskipun demikian, nilai F1-Score model stacking justru sedikit lebih rendah (0.5052) dibandingkan Random Forest (0.6257) dan Decision Tree (0.6066), yang mengindikasikan bahwa model stacking lebih kuat dalam prediksi keseluruhan (accuracy), tetapi belum optimal dalam mendeteksi semua kelas secara seimbang khususnya pada kelas minoritas. Nilai ROC AUC pada model stacking mencapai 0.9195, sedikit di bawah Random Forest (0.953) dan Decision Tree (0.951), namun tetap berada dalam kategori tinggi. Artinya, meskipun ROC AUC model stacking sedikit lebih rendah, namun kemampuannya dalam membedakan antar kelas masih sangat baik.



Gambar 4. 4 Grafik Perbandingan

Berdasarkan visualisasi grafik menunjukkan bahwa model ensemble memberikan peningkatan akurasi yang signifikan, meskipun dengan kompleksitas model dan waktu pelatihan yang lebih besar. Model individu seperti Decision Tree memberikan waktu pelatihan yang jauh lebih cepat dengan performa yang masih kompetitif.

4. *Classification Report*

Untuk mengevaluasi kinerja model pada tiap kelas secara spesifik, dilakukan evaluasi menggunakan *classification report*. Evaluasi ini sangat penting dalam konteks klasifikasi *multiclass*, terutama karena adanya ketidakseimbangan distribusi antar kelas. *Classification report* melengkapi metrik seperti *accuracy* dan *macro average*, dengan memberikan gambaran yang lebih rinci tentang keberhasilan model dalam mengenali tiap jenis kelas. Berikut adalah perhitungan metric secara manual:

1) Accuracy

Mengukur seberapa banyak prediksi yang benar dibandingan dengan total prediksi:

$$Accuracy = \frac{Jumlah\ Prediksi\ Benar}{Total\ Prediksi}$$

Berdasarkan data pelatihan sebanyak 234.662 sampel, dan total prediksi benar sebesar 203.276 maka:

$$Accuracy = \frac{203.276}{234.662} = 0.8662$$

2) *Precision*

Mengukur seberapa banyak data positif yang berhasil dikenali oleh model:

$$Precision = \frac{TP}{TP + FP}$$

3) *Recall*

Mengukur seberapa banyak data positif yang berhasil dikenali oleh model:

$$Recall = \frac{TP}{TP + FN}$$

4) *F1-score*

Rata-rata harmonis dari *precision* dan *recall:*

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Sebagai contoh, perhitungan metrik akan dilakukan untuk kelas terbanyak atau kelas mayoritas yaitu kelas 7 (Port Scan). Berikut adalah contoh perhitungannya:

a) TP = 86.938 (jumlah data kelas 7 yang diprediksi sebagai kelas 7)
b) FP = 24.508 (jumlah data bukan kelas 7 tapi diprediksi sebagai kelas 7)
c) FN = 1.314 (jumlah data kelas 7 tapi diprediksi sebagai kelas lain)

$$Precision = \frac{86.938}{86.938 + 24.508} = 0.7800$$

$$Recall = \frac{86.938}{86.938 + 1.314} = 0.9851$$

$$F1 = 2 \times \frac{0.7800 \times 0.9851}{0.7800 + 0.9851} = 0.8706$$

Nilai ini sangat mendekati nilai otomatis yang dihasilkan oleh *Classification report* dari model. Hal ini menunjukkan bahwa hasil evaluasi model telah sesuai secara teoritis dan praktis. Berikut adalah gambaran keseluruhan *classification report* yang ditunjukkan pada gambar 4.5

```
=== Classification Report per Kelas ===
              precision    recall  f1-score   support

           0     0.0000    0.0000    0.0000      2000
           1     0.8599    0.9202    0.8890    245830
           2     0.9073    0.9277    0.9174     50000
           3     0.0000    0.0000    0.0000     20000
           4     0.9823    0.9960    0.9891    180187
           5     0.0000    0.0000    0.0000     50000
           6     0.0000    0.0000    0.0000     50000
           7     0.6862    0.9838    0.8085    353008
           8     0.9360    0.9964    0.9652     50000
           9     0.8512    0.7966    0.8230     50000
          10     0.0000    0.0000    0.0000      2000
          11     0.6875    0.0002    0.0004     50000

    accuracy                         0.8060   1103025
   macro avg     0.4925    0.4684    0.4494   1103025
weighted avg     0.7250    0.8060    0.7411   1103025
```

Gambar 4. 5 Classification Report

Selain metric-metrik diatas, evaluasi juga dilengkapi dengan analisis ROC AUC. ROC AUC mengukur kemampuan model dalam membedakan antara kelas secara keseluruhan. Untuk multi-class classification, nilai ROC AUC dapat dihitung berdasarkan rata-rata AUC dari setiap kelas menggunakan formulasi berikut:

$$ROC\ AUC\ = \frac{1}{N} \sum_{i=1}^{N} \quad AUCi$$

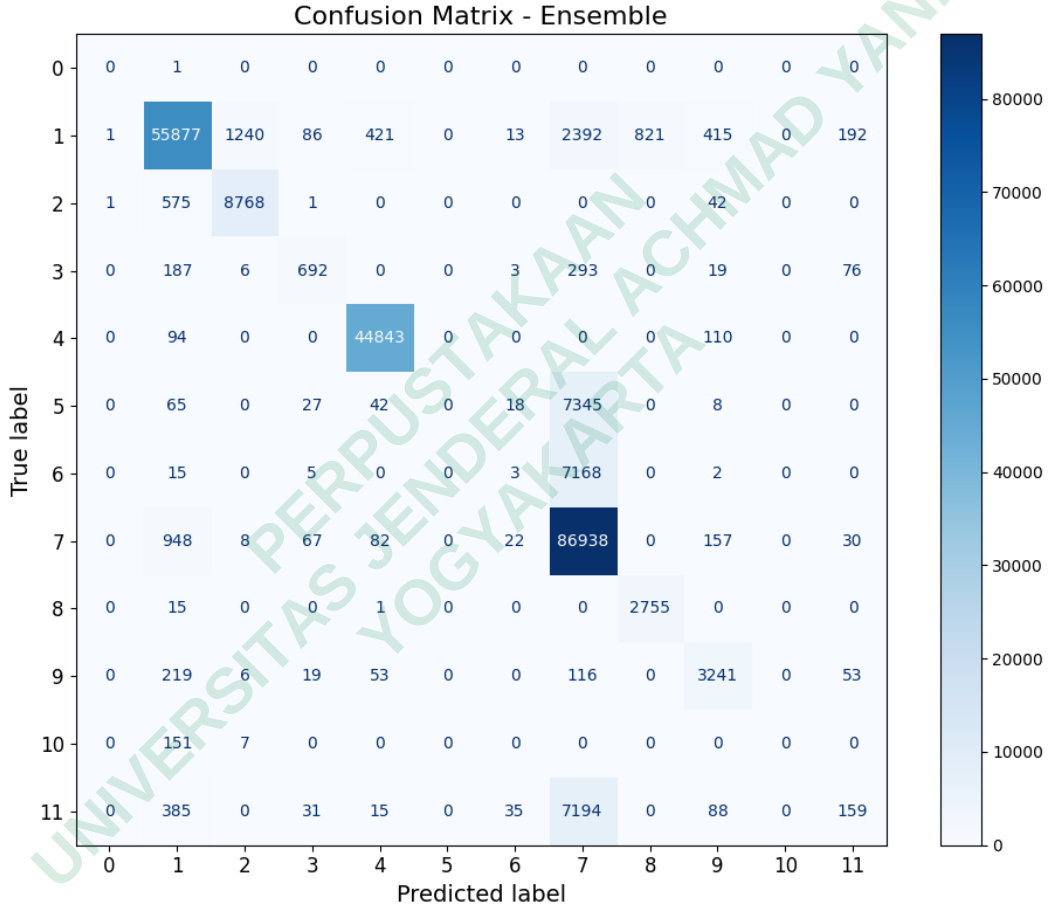
Dengan N adalah jumlah kelas (12 kelas) dan AUC_i adalah nilai AUC masing-masing kelas. Berdasarkan hasil perhitungan:

$$ROC\ AUC\ = \frac{\mathbf{0.89} + \mathbf{0.98} \ldots + \mathbf{0.89} + \mathbf{0.83}}{12} = 0.92$$

Nilai ini menunjukkan bahwa secara rata-rata model memiliki kemampuan diskriminasi yang sangat baik di seluruh kelas, meskipun beberapa kelas minoritas tetap mengalami kesulitan dalam klasifikasi.

5. Visualisasi Confusion Matrix

Confusion matrix digunakan untuk memahami ditribusi kesalahan klasifikasi pada masing-masing kelas. Berikut adalah confusion matrix yang menunjukkan jumlah prediksi benar pada tiap ditunjukkan pada gambar 4.5



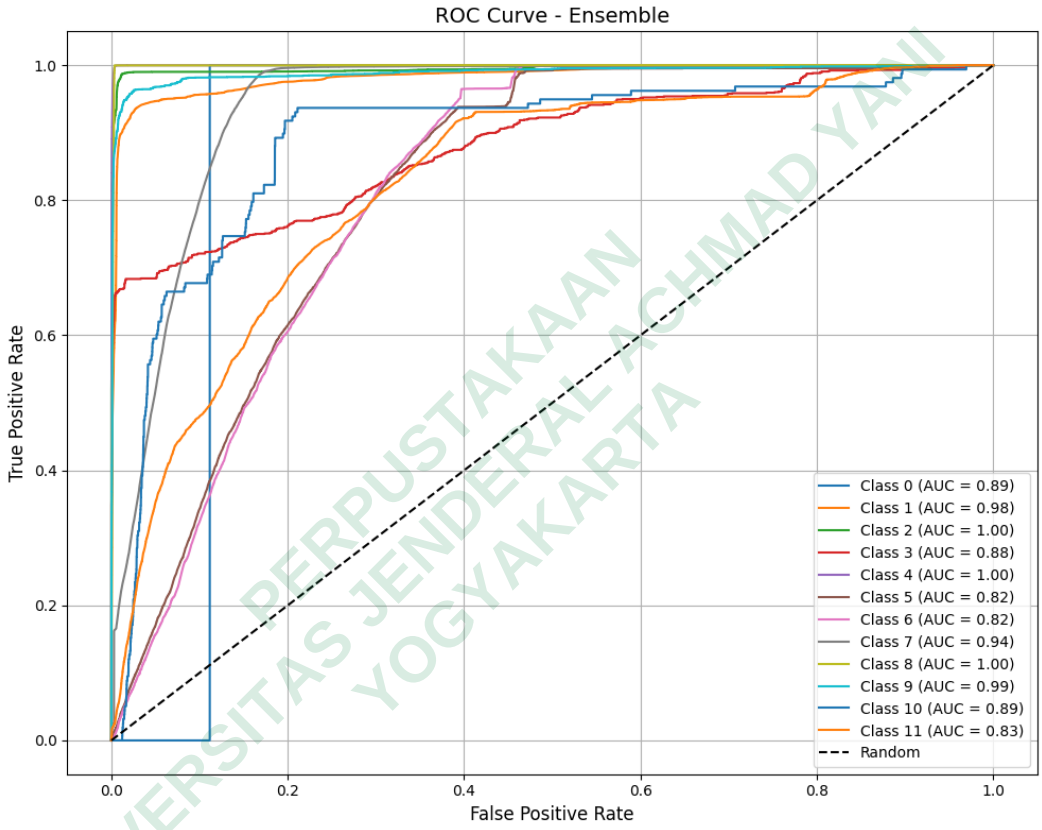
Gambar 4. 6 Confusion Matrix

Model menunjukkan performa yang sangat baik pada kelas mayoritas seperti kelas 7 (Port Scan), kelas 1 (benign), dan kelas 4 (ICMP Flood) dengan prediksi yang dominan benar. Tetapi masih terdapat kesalahan prediksi yang cukup signifikan pada kelas minoritas seperti kelas 0 (ARP Spoofing), kelas 5 (OS Scan) dan kelas 10 (UDP Flood) yang menunjukkan

bahwa model masih kesulitan mengenali pola dari kelas-kelas dengan jumlah data terbatas.

6. Visualisasi Kurva ROC

Kurva ROC menggambarkan performa model dalam membedakan masing-masing kelas terhadap kelas lainnya dengan pendekatan *One-vs-Rest* yang ditunjukkan pada gambar 4.6 berikut.

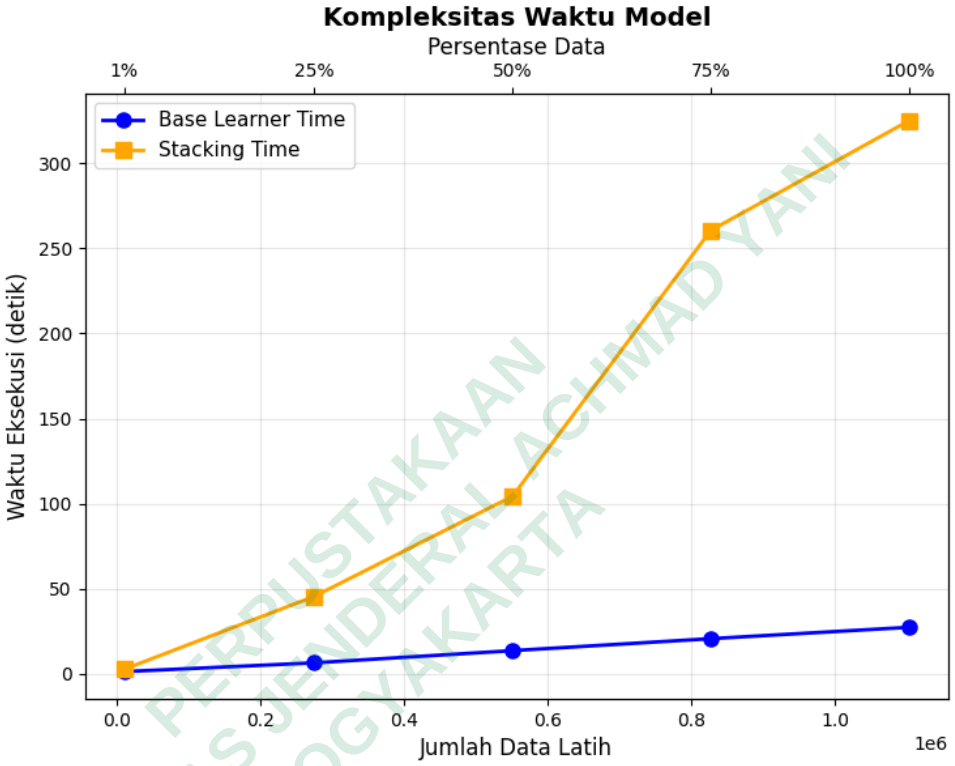


Gambar 4. 7 Kurva ROC

Sebagian kelas besar memiliki nilai AUC yang sangat tinggi, seperti kelas 2, 4, dan 8 dengan AUC = 1.00 kelas 9 AUC = 0.99, kelas 0 dan 10 AUC = 0.89. Tetapi masih terdapat beberapa kelas seperti kelas 5, 6 dan 11 memiliki nilai AUC dibawah 0.83 yang menunjukkan adanya keterbatasan model dalam membedakkan pola dari masing-masing kelas tersebut.

7. Analisis Efisiensi Komputasi dan Skalabilitas Model

Untuk mengevaluasi efisiensi komputasi, dilakukan perbandingan waktu pelatihan antar model. Yang dapat dilihat pada gambar berikut



Gambar 4. 8 Efisiensi Komputasi

Gambar menunjukkan perbandingan waktu eksekusi antara model *base learner* dan model *stacking* pada lima skala data yang berbeda, mulai dari 1% hingga 100% dari total data pelatihan 1 juta entri. Tujuan dari analisis ini adalah untuk menilai sejauh mana kompleksitas komputasi model meningkat seiring dengan bertambahnya jumlah data, serta mengevaluasi skalabilitas model terhadap dataset berskala besar.

Model base learner terbukti lebih efisien secara komputasi dan cocok digunakan pada sistem dengan keterbatasan sumber daya atau kebutuhan real-time. Model stacking memiliki *overhead* komputasi yang lebih tinggi, namun hal tersebut sebanding dengan peningkatan akurasi yang signifikan, sehingga lebih ideal untuk implementasi. Berdasarkan grafik

waktu eksekusi, titik terbaik antara performa dan efisiensi waktu berada pada penggunaan data sebesar 25–50%, di mana peningkatan akurasi sudah tercapai dengan waktu pelatihan yang masih wajar. Dari sisi skalabilitas, base learner lebih stabil dan efisien karena waktu pelatihannya meningkat secara linier terhadap jumlah data, sementara model stacking kurang *scalable* karena struktur pelatihan yang kompleks dan proses validasi silang yang memakan waktu.