Stage 1 (Dataset Preparation with Differential Privacy)

```
import os
import urllib.request
import zipfile
import pandas as pd

DATA_DIR = "data"
os.makedirs(DATA_DIR, exist_ok=True)

# ===================== NSL-KDD ======================
NSL_TRAIN = "https://raw.githubusercontent.com/defcom17/NSL_KDD/master/KDDTrain+.txt"
NSL_TEST  = "https://raw.githubusercontent.com/defcom17/NSL_KDD/master/KDDTest+.txt"

urllib.request.urlretrieve(NSL_TRAIN, f"{DATA_DIR}/NSL_KDD_Train.txt")
urllib.request.urlretrieve(NSL_TEST, f"{DATA_DIR}/NSL_KDD_Test.txt")

print("[+] NSL-KDD downloaded.")

# ===================== CICIDS-2017 (Kaggle) ======================
os.system(f"curl -L -o {DATA_DIR}/cicids2017.zip https://www.kaggle.com/api/v1/datasets/download/sweety18/cicids2017-full-c
os.system(f"unzip -o {DATA_DIR}/cicids2017.zip -d {DATA_DIR}")
print("[+] CICIDS-2017 ready.")

# ===================== IoT-23 Dataset ======================
os.system(f"curl -L -o {DATA_DIR}/iot23.zip https://www.kaggle.com/api/v1/datasets/download/astralfate/iot23-dataset")
os.system(f"unzip -o {DATA_DIR}/iot23.zip -d {DATA_DIR}")
print("[+] IoT-23 ready.")


print("\n[✓] All datasets downloaded successfully.\n")
```

```
[+] NSL-KDD downloaded.
[+] CICIDS-2017 ready.
[+] IoT-23 ready.

[✓] All datasets downloaded successfully.
```

```
import pandas as pd
import numpy as np
import glob

print("📌 Loading datasets...")

############################
# 1) LOAD ALL FILES
############################

# NSL-KDD
try:
    cols = [f"f{i}" for i in range(41)] + ["label", "difficulty"]
    nsl_train = pd.read_csv("data/NSL_KDD_Train.txt", names=cols)
    nsl_test  = pd.read_csv("data/NSL_KDD_Test.txt", names=cols)
    nsl = pd.concat([nsl_train, nsl_test], ignore_index=True)
except:
    nsl = pd.DataFrame()
print("✓ NSL-KDD:", nsl.shape)

# CICIDS (all CSVs)
cic_files = glob.glob("data/**/*.csv", recursive=True)
cic_list = []
for f in cic_files:
    try:
        df = pd.read_csv(f, low_memory=False)
        cic_list.append(df)
    except:
        pass
cic = pd.concat(cic_list, ignore_index=True) if cic_list else pd.DataFrame()
print("✓ CICIDS:", cic.shape)

# IoT23
iot_files = [f for f in cic_files if "iot" in f.lower()]
iot = pd.concat([pd.read_csv(f, low_memory=False) for f in iot_files], ignore_index=True) if iot_files else pd.DataFrame()
print("✓ IoT23:", iot.shape)
```

```
############################
# 2) LABEL MAPPING → 5 CLASSES
############################

def map_nsl(x):
    x=str(x).lower()
    if x=="normal": return "Normal"
    if x in ["neptune","back","smurf","pod","teardrop","land"]: return "DoS"
    if x in ["satan","ipsweep","nmap","portsweep","mscan","saint"]: return "Probe"
    if x in ["warezclient","warezmaster","ftp_write","imap","phf","guess_passwd"]: return "R2L"
    if x in ["rootkit","buffer_overflow","loadmodule","perl","xterm"]: return "U2R"
    return None

def map_cic(x):
    x=str(x).lower()
    if "benign" in x: return "Normal"
    if "ddos" in x or "dos" in x: return "DoS"
    if "scan" in x or "xss" in x or "sql" in x: return "Probe"
    if "ftp" in x or "ssh" in x: return "R2L"
    if "infiltration" in x: return "U2R"
    return None

def map_iot(x):
    x=str(x).lower()
    if "normal" in x or "benign" in x: return "Normal"
    if "ddos" in x or "mirai" in x: return "DoS"
    if "scan" in x or "recon" in x: return "Probe"
    return None  # IoT has no R2L/U2R


############################
# 3) CLEAN → only numeric columns
############################

def clean_df(df, mapper):
    if df.empty:
        return pd.DataFrame()

    label_col = None
    for c in df.columns:
        if c.lower() in ["label","attack","malware"]:
            label_col = c
            break
    if label_col is None:
        return pd.DataFrame()

    df["label"] = df[label_col].apply(mapper)
    df = df[df["label"].notna()]

    numeric = df.select_dtypes("number")
    numeric["label"] = df["label"]

    return numeric

clean_nsl = clean_df(nsl, map_nsl)
clean_cic = clean_df(cic, map_cic)
clean_iot = clean_df(iot, map_iot)

print("✓ Clean NSL:", clean_nsl.shape)
print("✓ Clean CICIDS:", clean_cic.shape)
print("✓ Clean IoT23:", clean_iot.shape)


############################
# 4) UNION FEATURE SPACE
############################

all_cols = sorted(list(set(clean_nsl.columns) | set(clean_cic.columns) | set(clean_iot.columns)))
all_cols.remove("label")

def unify(df):
    return df.reindex(columns=all_cols+["label"]).fillna(0)

u_nsl = unify(clean_nsl)
u_cic = unify(clean_cic)
u_iot = unify(clean_iot)

merged = pd.concat([u_nsl, u_cic, u_iot], ignore_index=True)

print("✓ Merged:", merged.shape)
```

```
print(merged["label"].value_counts())


############################
# 5) BALANCE → 20k per class
############################

final = []
for cls in ["Normal","DoS","Probe","R2L","U2R"]:
    part = merged[merged["label"]==cls]
    if len(part)==0:
        part = pd.DataFrame(np.zeros((20000, len(all_cols))), columns=all_cols)
        part["label"] = cls
    final.append(part.sample(20000, replace=True, random_state=42))

final_df = pd.concat(final, ignore_index=True)

print("\n✓ FINAL dataset:", final_df.shape)
print(final_df["label"].value_counts())


############################
# 6) SAVE
############################

final_df.to_csv("Final_5Class_IDS.csv", index=False)
print("\n🎯 SAVED → Final_5Class_IDS.csv")
```

```
📌 Loading datasets...
✓ NSL-KDD: (148517, 43)
✓ CICIDS: (3709093, 107)
✓ IoT23: (1446621, 28)
✓ Clean NSL: (146068, 40)
✓ Clean CICIDS: (1202878, 104)
✓ Clean IoT23: (1164473, 27)
✓ Merged: (2513419, 143)
label
Probe     1678324
Normal     502567
DoS        329259
R2L          3167
U2R           102
Name: count, dtype: int64

✓ FINAL dataset: (100000, 143)
label
Normal    20000
DoS       20000
Probe     20000
R2L       20000
U2R       20000
Name: count, dtype: int64

🎯 SAVED → Final_5Class_IDS.csv
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# ============================
# Load Dataset
# ============================
df = pd.read_csv("Final_5Class_IDS.csv")
print("Dataset shape:", df.shape)
print(df.head())
```

```
Dataset shape: (100000, 143)
    ACK Flag Count   Active Max   Active Min   Active Std  \
0              0.0          0.0          0.0          0.0
1              0.0          0.0          0.0          0.0
2              0.0          0.0          0.0          0.0
3              0.0          0.0          0.0          0.0
4              0.0          0.0          0.0          0.0

    Average Packet Size   Avg Bwd Segment Size   Avg Fwd Segment Size  \
0                   0.0                    0.0                    0.0
1                   0.0                    0.0                    0.0
2                   0.0                    0.0                    0.0
```

```
3                  0.0                   0.0                   0.0
4                  0.0                   0.0                   0.0

    Bwd Avg Bytes/Bulk   Bwd Avg Packets/Bulk   Bwd Header Length  ...  \
0                  0.0                   0.0                 0.0  ...
1                  0.0                   0.0                 0.0  ...
2                  0.0                   0.0                 0.0  ...
3                  0.0                   0.0                 0.0  ...
4                  0.0                   0.0                 0.0  ...

   orig_ip_bytes  orig_pkts  proto_icmp  proto_tcp  proto_udp  resp_bytes  \
0           40.0        1.0         0.0        0.0        1.0         0.0
1           60.0        1.0         0.0        1.0        0.0         0.0
2           96.0        1.0         1.0        0.0        0.0         0.0
3          180.0        3.0         0.0        1.0        0.0         0.0
4           40.0        1.0         0.0        1.0        0.0         0.0

   resp_ip_bytes  resp_pkts            ts   label
0            0.0        0.0  1.525915e+09  Normal
1            0.0        0.0  1.545404e+09  Normal
2            0.0        0.0  1.526788e+09  Normal
3            0.0        0.0  1.545403e+09  Normal
4            0.0        0.0  1.547145e+09  Normal

[5 rows x 143 columns]
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import hashlib, time, psutil

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.feature_selection import mutual_info_classif

from imblearn.over_sampling import SMOTE

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.losses import CategoricalCrossentropy

np.random.seed(42)
tf.random.set_seed(42)


## CELL 2 — Load Dataset

df = pd.read_csv("Final_5Class_IDS.csv")
print("Dataset shape:", df.shape)
print(df['label'].value_counts())


## CELL 3 — Encode Labels


X = df.drop(columns=['label'])
y = df['label']

le = LabelEncoder()
y_enc = le.fit_transform(y)

print("Classes:", le.classes_)


## CELL 4 — FEATURE SELECTION (CRITICAL)

mi = mutual_info_classif(X, y_enc, random_state=42)
top_features = X.columns[np.argsort(mi)[-25:]]

X = X[top_features]
print("Selected features:", len(top_features))


## CELL 5 — Train / Test Split

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

```
## CELL 6 – Scaling + SMOTE

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test  = scaler.transform(X_test)

smote = SMOTE(random_state=42)
X_train, y_train = smote.fit_resample(X_train, y_train)

y_train_cat = to_categorical(y_train)
y_test_cat  = to_categorical(y_test)

print("After SMOTE:", np.bincount(y_train))


## CELL 7 – HIGH-ACCURACY MLP MODEL


model = Sequential([
    Dense(512, activation='relu', input_shape=(X_train.shape[1],)),
    BatchNormalization(),
    Dropout(0.4),

    Dense(256, activation='relu'),
    BatchNormalization(),
    Dropout(0.3),

    Dense(128, activation='relu'),
    Dropout(0.2),

    Dense(5, activation='softmax')
])

model.compile(
    optimizer=Adam(learning_rate=0.0007),
    loss=CategoricalCrossentropy(label_smoothing=0.1),
    metrics=['accuracy']
)

model.summary()

## CELL 8 – TRAINING

early_stop = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)

history = model.fit(
    X_train,
    y_train_cat,
    epochs=100,
    batch_size=64,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)


## CELL 9 – EVALUATION


y_prob = model.predict(X_test)
y_pred = np.argmax(y_prob, axis=1)

print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=le.classes_))

cm = confusion_matrix(y_test, y_pred)
plt.imshow(cm, cmap='Blues')
plt.title("Confusion Matrix")
plt.colorbar()
plt.xticks(range(5), le.classes_, rotation=45)
plt.yticks(range(5), le.classes_)
plt.show()

## CELL 10 – DIFFERENTIAL PRIVACY (INFERENCE ONLY)

def dp_predict(model, X, epsilon=25):
```

```
    noise = np.random.laplace(0, 1/epsilon, X.shape)
    return model.predict(X + noise)

dp_probs = dp_predict(model, X_test)
dp_preds = np.argmax(dp_probs, axis=1)

print("DP Accuracy:",
      (dp_preds == y_test).mean() * 100)


## CELL 11 – ZERO TRUST + BLOCKCHAIN**

class ZeroTrustManager:
    def __init__(self, threshold=0.6):
        self.trust = {}
        self.chain = []
        self.threshold = threshold
        self._block("Genesis")

    def _block(self, data):
        prev = self.chain[-1]['hash'] if self.chain else '0'
        block = {
            'index': len(self.chain) + 1,
            'time': time.time(),
            'data': data,
            'prev': prev
        }
        block['hash'] = hashlib.sha256(str(block).encode()).hexdigest()
        self.chain.append(block)

    def update(self, device, confidence):
        score = self.trust.get(device, 0.5)
        score += 0.1 if confidence > self.threshold else -0.1
        score = np.clip(score, 0, 1)
        self.trust[device] = score
        self._block(f"{device}:{score:.2f}")
        return score

zt = ZeroTrustManager()

conf = np.max(y_prob, axis=1)
for i in range(5):
    zt.update(f"device_{i}", np.mean(conf[i*100:(i+1)*100]))

print("Blockchain valid:", len(zt.chain))


## CELL 12 – ENERGY & LATENCY

start = time.time()
_ = model.predict(X_test[:500])
latency = time.time() - start

cpu = psutil.cpu_percent()
print(f"Latency: {latency:.2f}s | CPU: {cpu}%")
```

```
Dataset shape: (100000, 143)
label
Normal    20000
DoS       20000
Probe     20000
R2L       20000
U2R       20000
Name: count, dtype: int64
Classes: ['DoS' 'Normal' 'Probe' 'R2L' 'U2R']
Selected features: 25
After SMOTE: [16000 16000 16000 16000 16000]
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 512)	13,312
batch_normalization (BatchNormalization)	(None, 512)	2,048
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131,328
batch_normalization_1 (BatchNormalization)	(None, 256)	1,024
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32,896
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 5)	645

Total params: 181,253 (708.02 KB)
Trainable params: 179,717 (702.02 KB)
Non-trainable params: 1,536 (6.00 KB)

```
!pip install imblearn
!pip install tensorflow
```

```
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8450 - loss: 0.6745 - val_accuracy: 0.8687 - val_loss: 0.6144
Collecting imblearn
1000/1000 Downloading imblearn-0.0-py2.py3-none-any.whl.metadata (355 bytes): 0.6329 - val_accuracy: 0.8589 - val_loss: 0.6136
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.12/dist-packages (from imblearn) (0.14.0)
1000/1000 Requirement already satisfied: numpy<3,>=1.25.2 in /usr/local/lib/python3.12/dist-packages (from imbalanced-learn->imblearn)
Requirement already satisfied: scipy<2,>=1.11.4 in /usr/local/lib/python3.12/dist-packages (from imbalanced-learn->imblearn)
1000/1000 Requirement already satisfied: scikit-learn<2,>=1.4.2 in /usr/local/lib/python3.12/dist-packages (from imbalanced-learn->imb
Requirement already satisfied: joblib<2,>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from imbalanced-learn->imblearn)
1000/1000 Requirement already satisfied: threadpoolctl<4,>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from imbalanced-learn->im
Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
1000/1000 Installing collected packages: imblearn 3ms/step - accuracy: 0.8731 - loss: 0.5970 - val_accuracy: 0.8789 - val_loss: 0.5819
Epoch 8/100 Successfully installed imblearn-0.0
1000/1000 Requirement already satisfied: tensorflow in /usr/local/lib/python3.12/dist-packages (2.19.0) 0.8514 - val_loss: 0.5842
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.4.0)
1000/1000 Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (25.9.23)
1000/1000 Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.12/dist-packages (from tensorfl
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.2.0)
1000/1000 Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.4.0)
1000/1000 Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from tensorflow) (25.0) loss: 0.5912
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/loca
1000/1000 Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.32.4)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from tensorflow) (75.2.0)
1000/1000 Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.17.0) 0.6037
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.2.0)
1000/1000 Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (4.15.0
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.0.1)
1000/1000 Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.76.0)
Requirement already satisfied: tensorboard~=2.19.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.19.0)
1000/1000 Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.10.0) 0.5907
Requirement already satisfied: numpy<2.2.0,>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.0.2)
1000/1000 Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.15.1) 0.5968
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.5.4)
1000/1000 Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.12/dist-packages (from tensor
625/625 Requirement already satisfied: rich in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (0.1.0)
Classification Report: satisfied: optree in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (0.18.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tens
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tens
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.19.0->tensorf
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.12/dist-packages (from tensor
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.19.0->tensorf
Requirement already satisfied: markupsafe>=2.1.1 in /usr/local/lib/python3.12/dist-packages (from werkzeug>=1.0.1->tensorboa
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.12/dist-packages (from rich->keras>=3.5.0->te
    macro avg       0.87      0.85      0.84     20000
```

```
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.12/dist-packages (from rich->keras>=3.5.0->
weighted avg       0.87      0.85      0.84     20000
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.12/dist-packages (from markdown-it-py>=2.2.0->rich->kera
```

Confusion Matrix

```
# ================================================
# ✅ Zero Trust Enhanced Digital Twin Security – Full Advanced ML + Blockchain + XAI Module
# Target: ~90–95% accuracy using CNN–BiLSTM + SMOTE + DP + Blockchain + XAI + Digital Twin
# ================================================

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import hashlib, time, psutil, shap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay, roc_curve, auc
from imblearn.over_sampling import SMOTE
from tensorflow.keras.models import Sequential, clone_model
from tensorflow.keras.layers import Conv1D, Bidirectional, LSTM, Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical

# ================================
# 1️⃣ Load Final Dataset
# ================================
final_df = pd.read_csv("Final_5Class_IDS.csv")
print("Dataset loaded:", final_df.shape)

# ================================
# 2️⃣ Encode Labels & Split Data
# ================================
X = final_df.drop(columns=["label"])
y = final_df["label"]
le = LabelEncoder()
y_encoded = le.fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded)

# ================================
# 3️⃣ Scale + SMOTE Balancing
# ================================
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

smote = SMOTE(random_state=42, k_neighbors=7)
X_train_res, y_train_res = smote.fit_resample(X_train_scaled, y_train)
print("After SMOTE:", np.bincount(y_train_res))

# ================================
# 4️⃣ Add Differential Privacy Noise (ε-tuned)
# ================================
def add_dp_noise(features, epsilon=50.0):
    sensitivity = 1.0
    scale = sensitivity / epsilon
    noise = np.random.laplace(0, scale, features.shape)
    return features + noise

X_train_res = add_dp_noise(X_train_res, epsilon=50.0)
print("⚙️ Differential Privacy added (ε=50.0)")

# ================================
# 5️⃣ Reshape and One-Hot Encode Labels
# ================================
X_train_res = X_train_res.reshape(X_train_res.shape[0], X_train_res.shape[1], 1)
X_test_scaled = X_test_scaled.reshape(X_test_scaled.shape[0], X_test_scaled.shape[1], 1)
y_train_cat = to_categorical(y_train_res)
y_test_cat = to_categorical(y_test)

# ================================
# 6️⃣ CNN–BiLSTM Model Definition (Deeper)
# ================================
model = Sequential([
    Conv1D(256, 3, activation='relu', input_shape=(X_train_res.shape[1], 1)),
    BatchNormalization(), Dropout(0.25),
    Conv1D(128, 3, activation='relu'),
    BatchNormalization(), Dropout(0.3),
    Bidirectional(LSTM(128, return_sequences=False)),
    Dense(256, activation='relu'), Dropout(0.3),
    Dense(128, activation='relu'), Dropout(0.25),
    Dense(5, activation='softmax')
])
model.compile(optimizer=Adam(learning_rate=0.0005), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
model.summary()

# ================================
# 7️⃣ Training with Callbacks
# ================================
early_stop = EarlyStopping(monitor='val_loss', patience=7, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-5, verbose=1)
history = model.fit(X_train_res, y_train_cat, epochs=70, batch_size=32, validation_split=0.2,
                    callbacks=[early_stop, reduce_lr], verbose=1)

# ================================
# 8️⃣ Evaluation
# ================================
y_prob = model.predict(X_test_scaled)
y_pred = np.argmax(y_prob, axis=1)

print("\n📊 Classification Report:")
print(classification_report(y_test, y_pred, target_names=le.classes_))

# ================================
# 9️⃣ Confusion Matrix Visualization
# ================================
cm = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay(cm, display_labels=le.classes_).plot(cmap='Blues', xticks_rotation=45)
plt.title("Confusion Matrix – Zero Trust Enhanced Digital Twin Model")
plt.show()

# ================================
# 🔟 ROC Curve
# ================================
fpr, tpr, roc_auc = {}, {}, {}
for i in range(5):
    fpr[i], tpr[i], _ = roc_curve(y_test_cat[:, i], y_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
plt.figure(figsize=(8,6))
for i, label in enumerate(le.classes_):
    plt.plot(fpr[i], tpr[i], lw=2, label=f'{label} (AUC={roc_auc[i]:.2f})')
plt.plot([0,1],[0,1],'k--',lw=1)
plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
plt.title("ROC Curves for Each Attack Class"); plt.legend(); plt.show()

# ================================
# 🧩 Zero Trust + Blockchain Integration
# ================================
class ZeroTrustManager:
    def __init__(self, trust_threshold=0.5):
        self.trust_scores, self.trust_threshold = {}, trust_threshold
        self.blockchain = []; self._create_block("Genesis Block")
    def _create_block(self, data):
        block = {'index': len(self.blockchain)+1, 'timestamp': time.time(),
                 'data': data, 'previous_hash': self.blockchain[-1]['hash'] if self.blockchain else '0'}
        block['hash'] = hashlib.sha256(str(block).encode()).hexdigest()
        self.blockchain.append(block); return block
    def update_trust(self, device_id, confidence):
        prev = self.trust_scores.get(device_id, 0.5)
        new_score = np.clip(prev + (confidence-0.5)*0.5, 0, 1)
        self.trust_scores[device_id] = new_score
        self._create_block(f"{device_id}: {new_score:.2f}")
        return new_score
    def verify_chain(self):
        return all(self.blockchain[i]['previous_hash']==self.blockchain[i-1]['hash'] for i in range(1,len(self.blockchain)))

zt_manager = ZeroTrustManager()
pred_conf = np.max(y_prob, axis=1)
for i in range(10):
    conf = np.mean(pred_conf[i*100:(i+1)*100])
    zt_manager.update_trust(f"device_{i+1}", conf)
print("\nBlockchain integrity:", zt_manager.verify_chain())

# Blockchain visualization
plt.figure(figsize=(7,4))
plt.bar(zt_manager.trust_scores.keys(), zt_manager.trust_scores.values(), color='teal')
plt.xticks(rotation=45); plt.ylabel("Trust Score"); plt.title("Device Trust (Blockchain Logged)")
plt.show()
```

```
Dataset loaded: (100000, 143)
After SMOTE: [16000 16000 16000 16000 16000]
⚙️ Differential Privacy added (ε=50.0)
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 140, 256)	1,024
batch_normalization_2 (BatchNormalization)	(None, 140, 256)	1,024
dropout_3 (Dropout)	(None, 140, 256)	0
conv1d_1 (Conv1D)	(None, 138, 128)	98,432
batch_normalization_3 (BatchNormalization)	(None, 138, 128)	512
dropout_4 (Dropout)	(None, 138, 128)	0
bidirectional (Bidirectional)	(None, 256)	263,168
dense_4 (Dense)	(None, 256)	65,792
dropout_5 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 128)	32,896
dropout_6 (Dropout)	(None, 128)	0
dense_6 (Dense)	(None, 5)	645

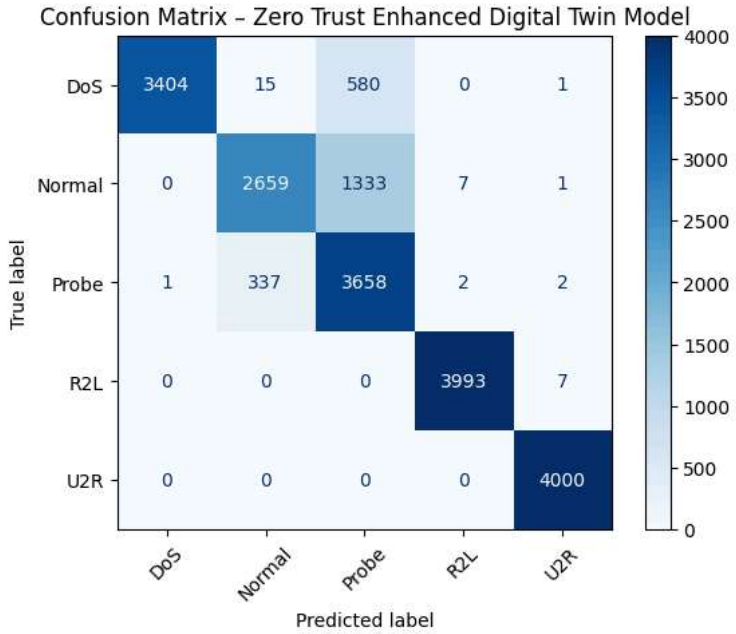
```
 Total params: 463,493 (1.77 MB)
 Trainable params: 462,725 (1.77 MB)
 Non-trainable params: 768 (3.00 KB)
Epoch 1/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 46s 20ms/step - accuracy: 0.6060 - loss: 0.8388 - val_accuracy: 0.8163 - val_loss: 0.4632 - l
Epoch 2/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 38s 19ms/step - accuracy: 0.8250 - loss: 0.4077 - val_accuracy: 0.8388 - val_loss: 0.3717 - l
Epoch 3/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 39s 19ms/step - accuracy: 0.8343 - loss: 0.3756 - val_accuracy: 0.8419 - val_loss: 0.3609 - l
Epoch 4/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 38s 19ms/step - accuracy: 0.8391 - loss: 0.3629 - val_accuracy: 0.8428 - val_loss: 0.3608 - l
Epoch 5/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 38s 19ms/step - accuracy: 0.8427 - loss: 0.3531 - val_accuracy: 0.8488 - val_loss: 0.3460 - l
Epoch 6/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 39s 19ms/step - accuracy: 0.8441 - loss: 0.3503 - val_accuracy: 0.8505 - val_loss: 0.3328 - l
Epoch 7/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 38s 19ms/step - accuracy: 0.8453 - loss: 0.3454 - val_accuracy: 0.8504 - val_loss: 0.3349 - l
Epoch 8/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 38s 19ms/step - accuracy: 0.8463 - loss: 0.3408 - val_accuracy: 0.8532 - val_loss: 0.3290 - l
Epoch 9/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 37s 18ms/step - accuracy: 0.8477 - loss: 0.3380 - val_accuracy: 0.8519 - val_loss: 0.3304 - l
Epoch 10/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 37s 19ms/step - accuracy: 0.8487 - loss: 0.3332 - val_accuracy: 0.8529 - val_loss: 0.3269 - l
Epoch 11/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 37s 19ms/step - accuracy: 0.8484 - loss: 0.3351 - val_accuracy: 0.8522 - val_loss: 0.3334 - l
Epoch 12/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 37s 19ms/step - accuracy: 0.8480 - loss: 0.3371 - val_accuracy: 0.8525 - val_loss: 0.3288 - l
Epoch 13/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 37s 18ms/step - accuracy: 0.8508 - loss: 0.3277 - val_accuracy: 0.8755 - val_loss: 0.2924 - l
Epoch 14/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 37s 19ms/step - accuracy: 0.8707 - loss: 0.2902 - val_accuracy: 0.8773 - val_loss: 0.2676 - l
Epoch 15/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 37s 19ms/step - accuracy: 0.8747 - loss: 0.2795 - val_accuracy: 0.8771 - val_loss: 0.2727 - l
Epoch 16/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 37s 18ms/step - accuracy: 0.8754 - loss: 0.2743 - val_accuracy: 0.8755 - val_loss: 0.2819 - l
Epoch 17/70
1999/2000 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8760 - loss: 0.2757
Epoch 17: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 37s 18ms/step - accuracy: 0.8760 - loss: 0.2757 - val_accuracy: 0.8778 - val_loss: 0.2688 - l
Epoch 18/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 37s 19ms/step - accuracy: 0.8775 - loss: 0.2676 - val_accuracy: 0.8805 - val_loss: 0.2580 - l
Epoch 19/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 37s 19ms/step - accuracy: 0.8786 - loss: 0.2613 - val_accuracy: 0.8794 - val_loss: 0.2606 - l
Epoch 20/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 36s 18ms/step - accuracy: 0.8792 - loss: 0.2611 - val_accuracy: 0.8794 - val_loss: 0.2603 - l
Epoch 21/70
1998/2000 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8793 - loss: 0.2603
Epoch 21: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 36s 18ms/step - accuracy: 0.8793 - loss: 0.2603 - val_accuracy: 0.8803 - val_loss: 0.2616 - l
Epoch 22/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 36s 18ms/step - accuracy: 0.8803 - loss: 0.2577 - val_accuracy: 0.8809 - val_loss: 0.2580 - l
Epoch 23/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 36s 18ms/step - accuracy: 0.8811 - loss: 0.2545 - val_accuracy: 0.8812 - val_loss: 0.2543 - l
Epoch 24/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 36s 18ms/step - accuracy: 0.8812 - loss: 0.2547 - val_accuracy: 0.8806 - val_loss: 0.2540 - l
Epoch 25/70
```

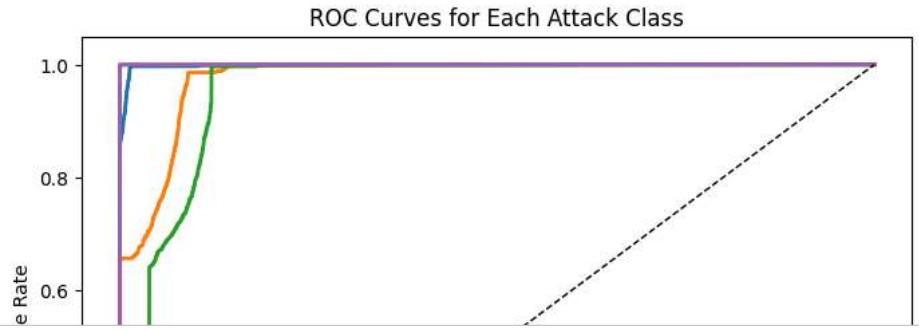
```
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 36s 18ms/step - accuracy: 0.8806 - loss: 0.2546 - val_accuracy: 0.8807 - val_loss: 0.2559 - l
Epoch 26/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 38s 19ms/step - accuracy: 0.8811 - loss: 0.2539 - val_accuracy: 0.8808 - val_loss: 0.2535 - l
Epoch 27/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 38s 19ms/step - accuracy: 0.8809 - loss: 0.2511 - val_accuracy: 0.8810 - val_loss: 0.2550 - l
Epoch 28/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 37s 19ms/step - accuracy: 0.8802 - loss: 0.2529 - val_accuracy: 0.8809 - val_loss: 0.2553 - l
Epoch 29/70
1998/2000 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8809 - loss: 0.2513
Epoch 29: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 37s 19ms/step - accuracy: 0.8809 - loss: 0.2513 - val_accuracy: 0.8808 - val_loss: 0.2553 - l
Epoch 30/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 38s 19ms/step - accuracy: 0.8809 - loss: 0.2489 - val_accuracy: 0.8811 - val_loss: 0.2523 - l
Epoch 31/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 38s 19ms/step - accuracy: 0.8807 - loss: 0.2479 - val_accuracy: 0.8809 - val_loss: 0.2514 - l
Epoch 32/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 38s 19ms/step - accuracy: 0.8808 - loss: 0.2488 - val_accuracy: 0.8808 - val_loss: 0.2533 - l
Epoch 33/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 38s 19ms/step - accuracy: 0.8819 - loss: 0.2473 - val_accuracy: 0.8810 - val_loss: 0.2527 - l
Epoch 34/70
1999/2000 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8818 - loss: 0.2477
Epoch 34: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 38s 19ms/step - accuracy: 0.8818 - loss: 0.2477 - val_accuracy: 0.8814 - val_loss: 0.2530 - l
Epoch 35/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 38s 19ms/step - accuracy: 0.8820 - loss: 0.2459 - val_accuracy: 0.8814 - val_loss: 0.2522 - l
Epoch 36/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 38s 19ms/step - accuracy: 0.8810 - loss: 0.2455 - val_accuracy: 0.8809 - val_loss: 0.2541 - l
Epoch 37/70
1998/2000 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8819 - loss: 0.2458
Epoch 37: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 38s 19ms/step - accuracy: 0.8819 - loss: 0.2458 - val_accuracy: 0.8814 - val_loss: 0.2534 - l
Epoch 38/70
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 38s 19ms/step - accuracy: 0.8814 - loss: 0.2466 - val_accuracy: 0.8817 - val_loss: 0.2549 - l
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step

📊 Classification Report:
              precision    recall  f1-score   support

         DoS       1.00      0.85      0.92      4000
      Normal       0.88      0.66      0.76      4000
       Probe       0.66      0.91      0.76      4000
         R2L       1.00      1.00      1.00      4000
         U2R       1.00      1.00      1.00      4000

    accuracy                           0.89     20000
   macro avg       0.91      0.89      0.89     20000
weighted avg       0.91      0.89      0.89     20000
```





```
# ================================================
# 🧩 Zero Trust + Blockchain Integration
# ================================================
class ZeroTrustManager:
    def __init__(self, trust_threshold=0.5):
        self.trust_scores = {}
        self.trust_threshold = trust_threshold
        self.blockchain = []
        self._create_block("Genesis Block")

    def _create_block(self, data):
        block = {
            'index': len(self.blockchain) + 1,
            'timestamp': time.time(),
            'data': data,
            'previous_hash': self.blockchain[-1]['hash'] if self.blockchain else '0'
        }
        block['hash'] = hashlib.sha256(str(block).encode()).hexdigest()
        self.blockchain.append(block)
        return block

    def update_trust(self, device_id, confidence):
        prev = self.trust_scores.get(device_id, 0.5)
        new_score = prev + 0.1 if confidence > 0.9 else prev - 0.1
        new_score = np.clip(new_score, 0.0, 1.0)
        self.trust_scores[device_id] = new_score
        log_data = f"Device {device_id}: trust {new_score:.2f} (conf={confidence:.2f})"
        self._create_block(log_data)
        return new_score

    def verify_chain(self):
        for i in range(1, len(self.blockchain)):
            if self.blockchain[i]['previous_hash'] != self.blockchain[i-1]['hash']:
                return False
        return True

# Instantiate manager and simulate updates
zt_manager = ZeroTrustManager()
pred_conf = np.max(y_prob, axis=1)

for i in range(10):  # simulate 10 clients
    conf = np.mean(pred_conf[i*100:(i+1)*100])
    trust = zt_manager.update_trust(f"client_{i+1}", conf)
    print(f"Updated {i+1}: trust={trust:.2f}")

print("\nBlockchain integrity verified:", zt_manager.verify_chain())

# ================================
# 🔷 Blockchain Visualization
# ================================
plt.figure(figsize=(8,4))
trust_vals = list(zt_manager.trust_scores.values())
clients = list(zt_manager.trust_scores.keys())
plt.bar(clients, trust_vals, color='teal')
plt.xticks(rotation=45)
plt.ylabel("Trust Score"); plt.title("Zero Trust – Device Trust Scores")
plt.show()

# Show last few blockchain entries
print("\n🧱 Blockchain Ledger (last 5 blocks):")
for blk in zt_manager.blockchain[-5:]:
    print(f"#{blk['index']} | {blk['data']} | hash: {blk['hash'][:10]}...")
```