

Individual AI/ML/DL Models Implementation Below is a complete Python script covering preprocessing, feature engineering, model training, testing, and saving outputs. Run on a standard server (CPU OK) — GPU speeds up the autoencoder training.

Save as train_models.py. It uses: pandas, numpy, scikit-learn, imblearn, xgboost, tensorflow (keras), matplotlib, seaborn.

```
# train_models_fixed.py
"""
Fixed, runnable version of your training script (XGBoost + Autoencoder
+ Stacked meta-classifier).
Put dataset CSV(s) in ./data/ and set DATASET to "UNSW" or "CICIDS".

Dependencies:
pip install pandas numpy scikit-learn imbalanced-learn xgboost
tensorflow joblib matplotlib seaborn
"""

import os
import numpy as np
import pandas as pd
import joblib
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (accuracy_score, precision_score,
                             recall_score,
                             f1_score, roc_auc_score,
                             confusion_matrix, classification_report,
                             roc_curve, auc)
from imblearn.over_sampling import SMOTE
import xgboost as xgb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.linear_model import LogisticRegression

# ----- Config -----
DATA_PATH = "data"      # folder with datasets
DATASET = "UNSW"        # "UNSW" or "CICIDS"
RANDOM_STATE = 42
TEST_SIZE = 0.2

# Define column names for UNSW-NB15_1.csv, assuming 49 columns based
# on common dataset structure
# This list includes 47 generic feature names plus 'attack_cat' and
# 'label' as the last two.
```

```

UNSW_NB15_COL_NAMES = [
    'srcip', 'sport', 'dstip', 'dsport', 'proto', 'state', 'dur',
    'sbytes',
    'dbytes', 'sttl', 'dttl', 'sloss', 'dloss', 'service', 'sload',
    'dload',
    'spkts', 'dpkts', 'swin', 'dwin', 'tcprrt', 'synack', 'ackdat',
    'smean',
    'dmean', 'trans_depth', 'response_body_len', 'ct_srv_src',
    'ct_ftp_cmd',
    'ct_src_ltm', 'ct_srv_dst', 'ct_dst_ltm', 'ct_src_dport_ltm',
    'ct_dst_sport_ltm',
    'ct_dst_src_ltm', 'is_ftp_login', 'ct_flw_http_mthd',
    'ct_src_ltm_2',
    'ct_srv_dst_2', 'is_sm_ips_ports', 'conn_id', 'ts', 'uid',
    'id.orig_h',
    'id.orig_p', 'id.resp_h', 'id.resp_p', 'attack_cat', 'label' #
    Last two are labels
]

# ----- Loading helpers -----
def load_unsw(path):
    if not os.path.exists(path):
        raise FileNotFoundError(f"UNSW CSV not found at {path}")
    # Load without header and assign column names explicitly for UNSW-
    NB15_1.csv
    df = pd.read_csv(path, header=None, names=UNSW_NB15_COL_NAMES,
        low_memory=False)
    return df

def load_cicids(path):
    if not os.path.exists(path):
        raise FileNotFoundError(f"CICIDS CSV not found at {path}")
    df = pd.read_csv(path, low_memory=False)
    return df

# ----- Preprocessing helpers -----
def basic_preprocess(df, label_col="label"):
    """
    - Consolidate label to binary: normal/benign -> 0, others -> 1
    - Keep numeric columns + label
    - Drop numeric columns that contain NaNs
    """
    df = df.copy()
    # Attempt to normalize label column name if it's not present
    if label_col not in df.columns:
        possible = [c for c in df.columns if c.lower() in ("label",
            "attack_cat", "class", "result")]
        if possible:
            label_col = possible[0]
        else:

```

```

        raise ValueError("Label column not found. Provide a
label_col present in the dataframe.")

    # Normalize label values to binary
    def map_label(x):
        s = str(x).strip().lower()
        if s in ("normal", "benign", "normal traffic",
"normal_traffic", "benign_traffic", "benignpacket", "0", "none"):
            return 0
        # sometimes CSVs have 'BENIGN' uppercase etc
        if "normal" in s or "benign" in s:
            return 0
        return 1

    df[label_col] = df[label_col].apply(map_label).astype(int)

    # Keep numeric features only (float/int)
    numeric = df.select_dtypes(include=[np.number]).copy()

    # Ensure label is present in numeric (if label was non-numeric
earlier, add it)
    if label_col not in numeric.columns:
        numeric[label_col] = df[label_col].values

    # Drop columns with any NaN in numeric (safer for modeling; you
can change policy if you prefer)
    cols_before = numeric.shape[1]
    numeric = numeric.dropna(axis=1)
    cols_after = numeric.shape[1]
    dropped = cols_before - cols_after
    if dropped > 0:
        print(f"[preprocess] Dropped {dropped} numeric columns due to
NaNs.")

    return numeric, label_col

def split_xy(df, label_col="label"):
    X = df.drop(columns=[label_col])
    y = df[label_col].astype(int)
    return X, y

# ----- Load dataset -----
# ===== Load dataset =====
if DATASET == "UNSW":
    csv_path = os.path.join(DATA_PATH, "UNSW-NB15_1.csv") # <--
correct filename
    df = load_unsw(csv_path)

    # set label column
    label_col = "label" # or "attack_cat" if you want multi-class

```

```

elif DATASET == "CICIDS":
    csv_path = os.path.join(DATA_PATH, "CICIDS2017.csv")
    df = load_cicids(csv_path)
    label_col = "Label" if "Label" in df.columns else ("label" if
"label" in df.columns else None)

else:
    raise ValueError("Set DATASET variable to UNSW or CICIDS")

# ----- Preprocess -----
df_proc, label_col = basic_preprocess(df, label_col=label_col)
X, y = split_xy(df_proc, label_col=label_col)
print(f"Features: {X.shape}, Label distribution:
{y.value_counts().to_dict()}")

# Train/test split (stratified)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=TEST_SIZE,
                                                    stratify=y,
random_state=RANDOM_STATE)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Save scaler for later use
joblib.dump(scaler, "scaler.joblib")

# ----- Handle imbalance with SMOTE on train
-----
sm = SMOTE(random_state=RANDOM_STATE)
# SMOTE expects 2D array, 1D labels
X_train_res, y_train_res = sm.fit_resample(X_train_scaled,
y_train.values)
print("After SMOTE class counts:",
np.bincount(y_train_res.astype(int)))

# ----- Model A: XGBoost -----
xgb_clf = xgb.XGBClassifier(
    n_estimators=200,
    max_depth=6,
    learning_rate=0.1,
    n_jobs=-1,
    use_label_encoder=False,
    eval_metric='logloss',
    random_state=RANDOM_STATE

```

```

)
xgb_clf.fit(X_train_res, y_train_res)
joblib.dump(xgb_clf, "xgb_model.joblib")

# Predictions
y_pred_xgb = xgb_clf.predict(X_test_scaled)
y_proba_xgb = xgb_clf.predict_proba(X_test_scaled)[: , 1]

# ----- Model B: Autoencoder (unsupervised anomaly
detection) -----
n_features = X_train_scaled.shape[1]
encoding_dim = max(8, n_features // 4)

autoencoder = Sequential([
    Dense(encoding_dim, activation='relu', input_shape=(n_features,)),
    Dense(max(4, encoding_dim // 2), activation='relu'),
    Dense(encoding_dim, activation='relu'),
    Dense(n_features, activation='linear')
])
autoencoder.compile(optimizer='adam', loss='mse')

# Train autoencoder only on normal samples (y_train == 0)
mask_train_normal = (y_train.values == 0)
if mask_train_normal.sum() < 10:
    raise ValueError("Too few normal samples to train the autoencoder.
Check dataset and label mapping.")

X_train_norm = X_train_scaled[mask_train_normal]
es = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
history = autoencoder.fit(X_train_norm, X_train_norm,
epochs=100, batch_size=256,
validation_split=0.1,
callbacks=[es], verbose=1)

autoencoder.save("autoencoder.h5")

# Calculate reconstruction error as anomaly score for test set
X_test_pred = autoencoder.predict(X_test_scaled)
mse = np.mean(np.square(X_test_scaled - X_test_pred), axis=1)

# choose threshold (e.g., mean + 3*std of train normal errors)
train_norm_pred = autoencoder.predict(X_train_norm)
train_mse = np.mean(np.square(X_train_norm - train_norm_pred), axis=1)
threshold = np.mean(train_mse) + 3 * np.std(train_mse)
print(f"[autoencoder] threshold = {threshold:.6f}")

y_pred_ae = (mse > threshold).astype(int)
ae_scores = mse # continuous anomaly score

```

```

# ----- Hybrid model (stacking): Logistic Regression
# combining XGB probability + AE score -----
skf = StratifiedKFold(n_splits=5, shuffle=True,
random_state=RANDOM_STATE)
oof_xgb = np.zeros(len(X_train_scaled), dtype=float)
oof_ae = np.zeros(len(X_train_scaled), dtype=float)

# Generate out-of-fold predictions for stacking training set
for train_idx, val_idx in skf.split(X_train_scaled, y_train):
    X_tr, X_val = X_train_scaled[train_idx], X_train_scaled[val_idx]
    y_tr, y_val = y_train.values[train_idx], y_train.values[val_idx]

    # Resample training fold to handle imbalance
    X_tr_res, y_tr_res = sm.fit_resample(X_tr, y_tr)

    # Fit a temporary XGB on the fold
    clf = xgb.XGBClassifier(
        n_estimators=100, max_depth=6, learning_rate=0.1,
        n_jobs=-1, use_label_encoder=False, eval_metric='logloss',
        random_state=RANDOM_STATE
    )
    clf.fit(X_tr_res, y_tr_res)

    # XGB OOF probability for val
    oof_xgb[val_idx] = clf.predict_proba(X_val)[: , 1]

    # AE recon error for the validation fold
    X_val_pred = autoencoder.predict(X_val)
    oof_ae[val_idx] = np.mean(np.square(X_val - X_val_pred), axis=1)

# Prepare stacking training set and test set features
stack_X_train = np.vstack([oof_xgb, oof_ae]).T
stack_y_train = y_train.values

X_test_xgb_proba = y_proba_xgb # from earlier trained full xgb_clf on
# whole train
stack_X_test = np.vstack([X_test_xgb_proba, ae_scores]).T

meta_clf = LogisticRegression(max_iter=1000)
meta_clf.fit(stack_X_train, stack_y_train)
y_pred_stack = meta_clf.predict(stack_X_test)
y_proba_stack = meta_clf.predict_proba(stack_X_test)[: , 1]

# ----- Evaluation helper -----
def evaluate(y_true, y_pred, y_proba=None, name="Model"):
    print(f"\n--- {name} ---")
    print("Accuracy:", accuracy_score(y_true, y_pred))
    print("Precision:", precision_score(y_true, y_pred,
zero_division=0))
    print("Recall:", recall_score(y_true, y_pred, zero_division=0))

```

```

print("F1:", f1_score(y_true, y_pred, zero_division=0))
if y_proba is not None:
    try:
        print("AUC:", roc_auc_score(y_true, y_proba))
    except Exception as e:
        print("AUC could not be computed:", e)
print("Confusion matrix:\n", confusion_matrix(y_true, y_pred))
print(classification_report(y_true, y_pred, digits=4,
zero_division=0))

# Evaluate all
evaluate(y_test.values, y_pred_xgb, y_proba_xgb, "XGBoost")
evaluate(y_test.values, y_pred_ae, ae_scores, "Autoencoder
(threshold)")
evaluate(y_test.values, y_pred_stack, y_proba_stack, "Hybrid
(Stacked)")

# Save meta model
joblib.dump(meta_clf, "meta_logreg.joblib")

# ----- Plot ROC curves -----
plt.figure(figsize=(8, 6))
fpr, tpr, _ = roc_curve(y_test.values, y_proba_xgb)
plt.plot(fpr, tpr, label=f'XGB (AUC={auc(fpr, tpr):.3f})')
# For AE use the anomaly scores as "probabilities" (higher means more
likely positive)
fpr, tpr, _ = roc_curve(y_test.values, ae_scores)
plt.plot(fpr, tpr, label=f'Autoencoder (AUC={auc(fpr, tpr):.3f})')
fpr, tpr, _ = roc_curve(y_test.values, y_proba_stack)
plt.plot(fpr, tpr, label=f'Hybrid (AUC={auc(fpr, tpr):.3f})')
plt.plot([0, 1], [0, 1], '--', color='gray')
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves")
plt.grid(True)
plt.savefig("roc_curves.png", dpi=200)
plt.close()

print("\nAll done. Models and scaler saved (xgb_model.joblib,
autoencoder.h5, meta_logreg.joblib, scaler.joblib).")

Features: (700001, 40), Label distribution: {0: 677786, 1: 22215}
After SMOTE class counts: [542228 542228]

/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199:
UserWarning: [18:59:31] WARNING: /workspace/src/learner.cc:790:
Parameters: { "use_label_encoder" } are not used.

bst.update(dtrain, iteration=i, fobj=obj)

```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py
:93: UserWarning: Do not pass an `input_shape`/`input_dim` argument to
a layer. When using Sequential models, prefer using an `Input(shape)`
object as the first layer in the model instead.
```

```
    super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

Epoch 1/100

1907/1907 ————— 8s 3ms/step - loss: 0.5422 - val_loss: 0.3471

Epoch 2/100

1907/1907 ————— 6s 3ms/step - loss: 0.3227 - val_loss: 0.3145

Epoch 3/100

1907/1907 ————— 5s 3ms/step - loss: 0.3241 - val_loss: 0.2977

Epoch 4/100

1907/1907 ————— 6s 3ms/step - loss: 0.3131 - val_loss: 0.2864

Epoch 5/100

1907/1907 ————— 5s 3ms/step - loss: 0.2930 - val_loss: 0.2801

Epoch 6/100

1907/1907 ————— 6s 3ms/step - loss: 0.2837 - val_loss: 0.2777

Epoch 7/100

1907/1907 ————— 6s 3ms/step - loss: 0.2741 - val_loss: 0.2761

Epoch 8/100

1907/1907 ————— 6s 3ms/step - loss: 0.3064 - val_loss: 0.2765

Epoch 9/100

1907/1907 ————— 7s 3ms/step - loss: 0.2711 - val_loss: 0.2751

Epoch 10/100

1907/1907 ————— 7s 3ms/step - loss: 0.2972 - val_loss: 0.2747

Epoch 11/100

1907/1907 ————— 9s 5ms/step - loss: 0.2804 - val_loss: 0.2742

Epoch 12/100

1907/1907 ————— 6s 3ms/step - loss: 0.2689 - val_loss: 0.2750

Epoch 13/100

1907/1907 ————— 7s 3ms/step - loss: 0.2853 - val_loss: 0.2740

Epoch 14/100

1907/1907 ————— 5s 3ms/step - loss: 0.2888 - val_loss: 0.2727

Epoch 15/100

```
1907/1907 _____ 6s 3ms/step - loss: 0.2713 - val_loss: 0.2727
Epoch 16/100
1907/1907 _____ 5s 3ms/step - loss: 0.2635 - val_loss: 0.2727
Epoch 17/100
1907/1907 _____ 6s 3ms/step - loss: 0.2754 - val_loss: 0.2725
Epoch 18/100
1907/1907 _____ 10s 3ms/step - loss: 0.2996 - val_loss: 0.2741
Epoch 19/100
1907/1907 _____ 7s 4ms/step - loss: 0.2612 - val_loss: 0.2724
Epoch 20/100
1907/1907 _____ 6s 3ms/step - loss: 0.2803 - val_loss: 0.2726
Epoch 21/100
1907/1907 _____ 7s 3ms/step - loss: 0.2743 - val_loss: 0.2737
Epoch 22/100
1907/1907 _____ 5s 3ms/step - loss: 0.2814 - val_loss: 0.2723
Epoch 23/100
1907/1907 _____ 6s 3ms/step - loss: 0.2869 - val_loss: 0.2726
Epoch 24/100
1907/1907 _____ 5s 2ms/step - loss: 0.2692 - val_loss: 0.2730
Epoch 25/100
1907/1907 _____ 5s 3ms/step - loss: 0.2757 - val_loss: 0.2719
Epoch 26/100
1907/1907 _____ 6s 3ms/step - loss: 0.2800 - val_loss: 0.2722
Epoch 27/100
1907/1907 _____ 5s 3ms/step - loss: 0.2719 - val_loss: 0.2714
Epoch 28/100
1907/1907 _____ 6s 3ms/step - loss: 0.2859 - val_loss: 0.2719
Epoch 29/100
1907/1907 _____ 6s 3ms/step - loss: 0.2797 - val_loss: 0.2713
Epoch 30/100
1907/1907 _____ 7s 4ms/step - loss: 0.2845 - val_loss: 0.2734
Epoch 31/100
1907/1907 _____ 5s 3ms/step - loss: 0.2856 - val_loss:
```

```
0.2711
Epoch 32/100
1907/1907 _____ 6s 3ms/step - loss: 0.2776 - val_loss:
0.2713
Epoch 33/100
1907/1907 _____ 7s 3ms/step - loss: 0.2941 - val_loss:
0.2723
Epoch 34/100
1907/1907 _____ 5s 3ms/step - loss: 0.2777 - val_loss:
0.2711
Epoch 35/100
1907/1907 _____ 7s 3ms/step - loss: 0.2726 - val_loss:
0.2718
Epoch 36/100
1907/1907 _____ 5s 3ms/step - loss: 0.2746 - val_loss:
0.2718
Epoch 37/100
1907/1907 _____ 6s 3ms/step - loss: 0.2832 - val_loss:
0.2717
Epoch 38/100
1907/1907 _____ 6s 3ms/step - loss: 0.2721 - val_loss:
0.2707
Epoch 39/100
1907/1907 _____ 7s 4ms/step - loss: 0.2617 - val_loss:
0.2724
Epoch 40/100
1907/1907 _____ 5s 3ms/step - loss: 0.2847 - val_loss:
0.2717
Epoch 41/100
1907/1907 _____ 6s 3ms/step - loss: 0.2874 - val_loss:
0.2712
Epoch 42/100
1907/1907 _____ 5s 3ms/step - loss: 0.2838 - val_loss:
0.2711
Epoch 43/100
1907/1907 _____ 5s 3ms/step - loss: 0.2907 - val_loss:
0.2705
Epoch 44/100
1907/1907 _____ 7s 3ms/step - loss: 0.2676 - val_loss:
0.2713
Epoch 45/100
1907/1907 _____ 5s 3ms/step - loss: 0.2716 - val_loss:
0.2712
Epoch 46/100
1907/1907 _____ 6s 3ms/step - loss: 0.2594 - val_loss:
0.2708
Epoch 47/100
1907/1907 _____ 6s 3ms/step - loss: 0.2848 - val_loss:
0.2716
```

Epoch 48/100
1907/1907 ————— 8s 4ms/step - loss: 0.2643 - val_loss:
0.2713

WARNING:absl:You are saving your model as an HDF5 file via
`model.save()` or `keras.saving.save_model(model)`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.

4376/4376 ————— 11s 2ms/step
16945/16945 ————— 25s 1ms/step
[autoencoder] threshold = 29.200563

/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199:
UserWarning: [19:05:56] WARNING: /workspace/src/learner.cc:790:
Parameters: { "use_label_encoder" } are not used.

bst.update(dtrain, iteration=i, fobj=obj)

3500/3500 ————— 6s 2ms/step

/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199:
UserWarning: [19:06:24] WARNING: /workspace/src/learner.cc:790:
Parameters: { "use_label_encoder" } are not used.

bst.update(dtrain, iteration=i, fobj=obj)

3500/3500 ————— 5s 1ms/step

/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199:
UserWarning: [19:06:49] WARNING: /workspace/src/learner.cc:790:
Parameters: { "use_label_encoder" } are not used.

bst.update(dtrain, iteration=i, fobj=obj)

3500/3500 ————— 5s 1ms/step

/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199:
UserWarning: [19:07:15] WARNING: /workspace/src/learner.cc:790:
Parameters: { "use_label_encoder" } are not used.

bst.update(dtrain, iteration=i, fobj=obj)

3500/3500 ————— 4s 1ms/step

/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199:
UserWarning: [19:07:40] WARNING: /workspace/src/learner.cc:790:
Parameters: { "use_label_encoder" } are not used.

bst.update(dtrain, iteration=i, fobj=obj)

3500/3500 ————— 4s 1ms/step

--- XGBoost ---

Accuracy: 0.9977214448468226

Precision: 0.937791932059448

Recall: 0.9941480981318929

F1: 0.9651480388943516

AUC: 0.9998948801900652

Confusion matrix:

[[135265 293]

[26 4417]]

	precision	recall	f1-score	support
0	0.9998	0.9978	0.9988	135558
1	0.9378	0.9941	0.9651	4443

accuracy			0.9977	140001
macro avg	0.9688	0.9960	0.9820	140001
weighted avg	0.9978	0.9977	0.9978	140001

--- Autoencoder (threshold) ---

Accuracy: 0.9682930836208313

Precision: 0.5294117647058824

Recall: 0.008102633355840648

F1: 0.015960984260696077

AUC: 0.9823479279285221

Confusion matrix:

[[135526 32]

[4407 36]]

	precision	recall	f1-score	support
0	0.9685	0.9998	0.9839	135558
1	0.5294	0.0081	0.0160	4443

accuracy			0.9683	140001
macro avg	0.7490	0.5039	0.4999	140001
weighted avg	0.9546	0.9683	0.9532	140001

--- Hybrid (Stacked) ---

Accuracy: 0.9981571560203142

Precision: 0.9659318637274549

Recall: 0.9763673193787981

F1: 0.9711215580926796

AUC: 0.999894855284879

Confusion matrix:

[[135405 153]

[105 4338]]

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.9992	0.9989	0.9990	135558
1	0.9659	0.9764	0.9711	4443
accuracy			0.9982	140001
macro avg	0.9826	0.9876	0.9851	140001
weighted avg	0.9982	0.9982	0.9982	140001
All done. Models and scaler saved (xgb_model.joblib, autoencoder.h5, meta_logreg.joblib, scaler.joblib).				