The **NSL-KDD** dataset is a benchmark dataset used for evaluating **intrusion detection systems (IDS)**. It is an improved version of the **KDD'99** dataset, created to address issues like redundancy and class imbalance. Each record represents a network connection and is labeled as either **normal** or an **attack** (e.g., DoS, Probe, R2L, U2R), with **41 features** describing the connection. The dataset is widely used in cybersecurity and machine learning research for developing and testing intrusion detection algorithms.

Features

---

1. Basic Features of Individual TCP Connections

1. `duration` – length (in seconds) of the connection
2. `protocol_type` – type of protocol (e.g., tcp, udp, icmp)
3. `service` – network service on the destination (e.g., http, telnet)
4. `flag` – status flag of the connection
5. `src_bytes` – number of data bytes from source to destination
6. `dst_bytes` – number of data bytes from destination to source
7. `land` – 1 if connection is from/to the same host/port; 0 otherwise
8. `wrong_fragment` – number of wrong fragments
9. `urgent` – number of urgent packets

---

2. Content Features within a Connection Suggested by Domain Knowledge

10. `hot` – number of "hot" indicators
11. `num_failed_logins` – number of failed login attempts
12. `logged_in` – 1 if successfully logged in; 0 otherwise
13. `num_compromised` – number of compromised conditions
14. `root_shell` – 1 if root shell is obtained; 0 otherwise
15. `su_attempted` – 1 if `su root` command attempted; 0 otherwise
16. `num_root` – number of “root” accesses
17. `num_file_creations` – number of file creation operations
18. `num_shells` – number of shell prompts
19. `num_access_files` – number of operations on access control files
20. `num_outbound_cmds` – number of outbound commands (always 0 in KDD)
21. `is_host_login` – 1 if login belongs to the host list; 0 otherwise
22. `is_guest_login` – 1 if login is a guest login; 0 otherwise

---

3. Traffic Features (Same Host)

23. `count` – number of connections to the same host in the past 2 seconds
24. `srv_count` – number of connections to the same service in past 2 seconds
25. `serror_rate` – % of connections with “SYN” errors
26. `srv_serror_rate` – % of same service connections with “SYN” errors
27. `rerror_rate` – % of connections with “REJ” errors
28. `srv_rerror_rate` – % of same service connections with “REJ” errors
29. `same_srv_rate` – % of connections to the same service
30. `diff_srv_rate` – % of connections to different services
31. `srv_diff_host_rate` – % of same service connections to different hosts

---

4. Traffic Features (Same Service)

32. `dst_host_count` – number of connections to same host
33. `dst_host_srv_count` – number of connections to same service
34. `dst_host_same_srv_rate` – % of connections to same service
35. `dst_host_diff_srv_rate` – % of connections to different services
36. `dst_host_same_src_port_rate` – % of connections from same source port
37. `dst_host_srv_diff_host_rate` – % of same service connections to different hosts
38. `dst_host_serror_rate` – % of connections with SYN errors
39. `dst_host_srv_serror_rate` – % of same service connections with SYN errors
40. `dst_host_rerror_rate` – % of connections with REJ errors

41. `dst_host_srv_rerror_rate` – % of same service connections with REJ errors

IMPORT LIBRARIES

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

READ DATASET

In [3]:
```
df_0 = pd.read_csv(r"KDDTrain+.txt")
df= df_0.copy()
df.head()
```

Out[3]:

	0	tcp	ftp_data	SF	491	0.1	0.2	0.3	0.4	0.5	...	0.17	0.03	0.17.1	0.00.6	0.00.7	0.00.8	0.05	0.00.9	normal	20
0	0	udp	other	SF	146	0	0	0	0	0	...	0.00	0.60	0.88	0.00	0.00	0.00	0.0	0.00	normal	15
1	0	tcp	private	S0	0	0	0	0	0	0	...	0.10	0.05	0.00	0.00	1.00	1.00	0.0	0.00	neptune	19
2	0	tcp	http	SF	232	8153	0	0	0	0	...	1.00	0.00	0.03	0.04	0.03	0.01	0.0	0.01	normal	21
3	0	tcp	http	SF	199	420	0	0	0	0	...	1.00	0.00	0.00	0.00	0.00	0.00	0.0	0.00	normal	21
4	0	tcp	private	REJ	0	0	0	0	0	0	...	0.07	0.07	0.00	0.00	0.00	0.00	1.0	1.00	neptune	21

5 rows × 43 columns

In [6]:
```
columns = (['duration','protocol_type','service','flag','src_bytes','dst_bytes','land','wrong_fragment','urgent

df.columns = columns
```

Source : https://www.kaggle.com/code/timgoodfellow/nsl-kdd-explorations

In [7]:
```
df.head(5)
```

Out[7]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_same_srv_rate	ds
0	0	udp	other	SF	146	0	0	0	0	0	...	0.00	
1	0	tcp	private	S0	0	0	0	0	0	0	...	0.10	
2	0	tcp	http	SF	232	8153	0	0	0	0	...	1.00	
3	0	tcp	http	SF	199	420	0	0	0	0	...	1.00	
4	0	tcp	private	REJ	0	0	0	0	0	0	...	0.07	

5 rows × 43 columns

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 125972 entries, 0 to 125971
Data columns (total 43 columns):
 #   Column                       Non-Null Count   Dtype  
---  ------                       --------------   -----  
 0   duration                     125972 non-null  int64  
 1   protocol_type                125972 non-null  object 
 2   service                      125972 non-null  object 
 3   flag                         125972 non-null  object 
 4   src_bytes                    125972 non-null  int64  
 5   dst_bytes                    125972 non-null  int64  
 6   land                         125972 non-null  int64  
 7   wrong_fragment               125972 non-null  int64  
 8   urgent                       125972 non-null  int64  
 9   hot                          125972 non-null  int64  
 10  num_failed_logins            125972 non-null  int64  
 11  logged_in                    125972 non-null  int64  
 12  num_compromised              125972 non-null  int64  
 13  root_shell                   125972 non-null  int64  
 14  su_attempted                 125972 non-null  int64  
 15  num_root                     125972 non-null  int64  
 16  num_file_creations           125972 non-null  int64  
 17  num_shells                   125972 non-null  int64  
 18  num_access_files             125972 non-null  int64  
 19  num_outbound_cmds            125972 non-null  int64  
 20  is_host_login                125972 non-null  int64  
 21  is_guest_login               125972 non-null  int64  
 22  count                        125972 non-null  int64  
 23  srv_count                    125972 non-null  int64  
 24  serror_rate                  125972 non-null  float64
 25  srv_serror_rate              125972 non-null  float64
 26  rerror_rate                  125972 non-null  float64
 27  srv_rerror_rate              125972 non-null  float64
 28  same_srv_rate                125972 non-null  float64
 29  diff_srv_rate                125972 non-null  float64
 30  srv_diff_host_rate           125972 non-null  float64
 31  dst_host_count               125972 non-null  int64  
 32  dst_host_srv_count           125972 non-null  int64  
 33  dst_host_same_srv_rate       125972 non-null  float64
 34  dst_host_diff_srv_rate       125972 non-null  float64
 35  dst_host_same_src_port_rate  125972 non-null  float64
 36  dst_host_srv_diff_host_rate  125972 non-null  float64
 37  dst_host_serror_rate         125972 non-null  float64
 38  dst_host_srv_serror_rate     125972 non-null  float64
 39  dst_host_rerror_rate         125972 non-null  float64
 40  dst_host_srv_rerror_rate     125972 non-null  float64
 41  attack                       125972 non-null  object 
 42  level                        125972 non-null  int64  
dtypes: float64(15), int64(24), object(4)
memory usage: 41.3+ MB
```

We have different types of dtypes, we need encoding, doesn't seem like we have null values but we will check

DATA CLEANING

NULL VALUES

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
duration                        0
protocol_type                   0
service                         0
flag                            0
src_bytes                       0
dst_bytes                       0
land                            0
wrong_fragment                  0
urgent                          0
hot                             0
num_failed_logins               0
logged_in                       0
num_compromised                 0
root_shell                      0
su_attempted                    0
num_root                        0
num_file_creations              0
num_shells                      0
num_access_files                0
num_outbound_cmds               0
is_host_login                   0
is_guest_login                  0
count                           0
srv_count                       0
serror_rate                     0
srv_serror_rate                 0
rerror_rate                     0
srv_rerror_rate                 0
same_srv_rate                   0
diff_srv_rate                   0
srv_diff_host_rate              0
dst_host_count                  0
dst_host_srv_count              0
dst_host_same_srv_rate          0
dst_host_diff_srv_rate          0
dst_host_same_src_port_rate     0
dst_host_srv_diff_host_rate     0
dst_host_serror_rate            0
dst_host_srv_serror_rate        0
dst_host_rerror_rate            0
dst_host_srv_rerror_rate        0
attack                          0
level                           0
dtype: int64
```

Dataset doesn't contain any null value

DUPLICATES

In [10]:
```
df.duplicated().sum()
```

Out[10]:
```
0
```

Dataset doesn't contain any duplicated row

OUTLIERS

In [11]:
```
df.shape
```

Out[11]:
```
(125972, 43)
```

In [ ]:
```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

rows, cols = 8, 5
fig = make_subplots(rows=rows, cols=cols, subplot_titles=df.columns)

i = 0
for row in range(1, rows + 1):
    for col in range(1, cols + 1):
        if i < len(df.columns):
            col_name = df.columns[i]
            fig.add_trace(
                go.Box(y=df[col_name], name=col_name, boxpoints='outliers'),
                row=row, col=col
            )
            i += 1

fig.update_layout(height=2000, width=1200, title_text="Boxplots of Features")
fig.show()
```

We chose not to remove outliers from this dataset because it contains ID-like or identifier features, which might appear as outliers but are actually meaningful. Moreover, SMOTE handles class imbalance effectively, and neural networks are generally robust to mild outliers, especially after normalization.

Make It binary

In [13]:
```
df['attack'] = ['normal' if i == 'normal' else 'attack' for i in df['attack']]
```

In [14]:
```
df['attack'].unique()
```

Out[14]:
```
array(['normal', 'attack'], dtype=object)
```

PREPROCESSING

ENCODING

In [15]:
```
cat_features = df.select_dtypes(include='object').columns
cat_features
```

Out[15]:
```
Index(['protocol_type', 'service', 'flag', 'attack'], dtype='object')
```

In [16]:
```
from sklearn import preprocessing
le=preprocessing.LabelEncoder()
column=['protocol_type', 'service', 'flag', 'attack']
for x in column:
    df[x]=le.fit_transform(df[x])
```

SPLIT THE DATA

In [17]:
```
from sklearn.model_selection import train_test_split

X = df.drop(["attack"], axis=1)
y = df["attack"]

X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2,random_state=43)
```

In [18]:
```
train_index = X_train.columns
train_index
```

Out[18]:
```
Index(['duration', 'protocol_type', 'service', 'flag', 'src_bytes',
       'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot',
       'num_failed_logins', 'logged_in', 'num_compromised', 'root_shell',
       'su_attempted', 'num_root', 'num_file_creations', 'num_shells',
       'num_access_files', 'num_outbound_cmds', 'is_host_login',
       'is_guest_login', 'count', 'srv_count', 'serror_rate',
       'srv_serror_rate', 'rerror_rate', 'srv_rerror_rate', 'same_srv_rate',
       'diff_srv_rate', 'srv_diff_host_rate', 'dst_host_count',
       'dst_host_srv_count', 'dst_host_same_srv_rate',
       'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',
       'dst_host_srv_diff_host_rate', 'dst_host_serror_rate',
       'dst_host_srv_serror_rate', 'dst_host_rerror_rate',
       'dst_host_srv_rerror_rate', 'level'],
      dtype='object')
```

Feature Engineering

Mutual Information

$$\left(\frac{p(x,y)}{p(x)p(y)}\right) p(x,y) \log \sum_{y\in Y}\sum_{x\in X} = I(X;Y)$$

In [19]:
```
from sklearn.feature_selection import mutual_info_classif
mutual_info = mutual_info_classif(X_train, y_train)
mutual_info = pd.Series(mutual_info)
mutual_info.index = train_index
mutual_info.sort_values(ascending=False)
```

Out[19]:
```
src_bytes                       0.565920
service                         0.468649
dst_bytes                       0.439254
flag                            0.368508
same_srv_rate                   0.363403
diff_srv_rate                   0.358261
dst_host_srv_count              0.335522
dst_host_same_srv_rate          0.312897
logged_in                       0.289576
dst_host_serror_rate            0.287460
dst_host_diff_srv_rate          0.284164
dst_host_srv_serror_rate        0.279930
serror_rate                     0.275758
srv_serror_rate                 0.268365
count                           0.263614
dst_host_srv_diff_host_rate     0.188440
level                           0.151983
dst_host_count                  0.141203
dst_host_same_src_port_rate     0.130408
srv_diff_host_rate              0.099664
srv_count                       0.065486
dst_host_srv_rerror_rate        0.064493
protocol_type                   0.051065
rerror_rate                     0.038822
srv_rerror_rate                 0.036605
dst_host_rerror_rate            0.036102
duration                        0.024509
wrong_fragment                  0.006457
hot                             0.006418
num_compromised                 0.003573
is_host_login                   0.002536
su_attempted                    0.001665
num_failed_logins               0.001390
num_root                        0.001094
num_outbound_cmds               0.001039
num_shells                      0.000373
is_guest_login                  0.000186
num_access_files                0.000000
num_file_creations              0.000000
root_shell                      0.000000
urgent                          0.000000
land                            0.000000
dtype: float64
```

In [20]:
```
import plotly.express as px

sorted_mutual_info = mutual_info.sort_values(ascending=False)

fig = px.bar(
    x=sorted_mutual_info.index,
    y=sorted_mutual_info.values,
    labels={'x': 'Features', 'y': 'Mutual Information'},
    title='Mutual Information Scores',
)

fig.update_layout(
    xaxis_tickangle=90,
    width=1100,
    height=700,
)
fig.show()
```

Feature Selection

In [21]:
```
from sklearn.feature_selection import SelectKBest
Select_features = SelectKBest(mutual_info_classif, k=15)
Select_features.fit(X_train, y_train)
train_index[Select_features.get_support()]
```

Out[21]:
```
Index(['service', 'flag', 'src_bytes', 'dst_bytes', 'logged_in', 'count',
       'serror_rate', 'srv_serror_rate', 'same_srv_rate', 'diff_srv_rate',
       'dst_host_srv_count', 'dst_host_same_srv_rate',
       'dst_host_diff_srv_rate', 'dst_host_serror_rate',
       'dst_host_srv_serror_rate'],
      dtype='object')
```

In [22]:
```
columns=['service', 'flag', 'src_bytes', 'dst_bytes', 'logged_in', 'count',
       'serror_rate', 'srv_serror_rate', 'same_srv_rate', 'diff_srv_rate',
       'dst_host_srv_count', 'dst_host_same_srv_rate',
       'dst_host_diff_srv_rate', 'dst_host_serror_rate',
       'dst_host_srv_serror_rate']
```

```
X_train=X_train[columns]
X_test=X_test[columns]
```

RESAMPLING (SMOTE) AND SCALING

In [23]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, LeakyReLU
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
from imblearn.over_sampling import SMOTE

#Handle Imbalance with SMOTE
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

#Scale Resampled and Test Data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_resampled)
X_test_scaled = scaler.transform(X_test)
```

In [24]:
```
class_counts_before = y_train.value_counts().sort_index()
class_counts_after = pd.Series(y_train_resampled).value_counts().sort_index()

print("Class Distribution Before SMOTE:")
print(class_counts_before)
print("\nClass Distribution After SMOTE:")
print(class_counts_after)

fig = make_subplots(rows=1, cols=2, subplot_titles=("Before SMOTE", "After SMOTE"))

fig.add_trace(
    go.Bar(
        x=class_counts_before.index.astype(str),
        y=class_counts_before.values,
        name='Before SMOTE',
        marker_color='blue'
    ),
    row=1, col=1
)

fig.add_trace(
    go.Bar(
        x=class_counts_after.index.astype(str),
        y=class_counts_after.values,
        name='After SMOTE',
        marker_color='green'
    ),
    row=1, col=2
)

fig.update_layout(
    height=400,
    width=900,
    showlegend=False,
    title_text="Class Distribution Before and After SMOTE"
)

fig.update_xaxes(title_text="Class", row=1, col=1)
fig.update_xaxes(title_text="Class", row=1, col=2)
fig.update_yaxes(title_text="Count", row=1, col=1)
fig.update_yaxes(title_text="Count", row=1, col=2)

fig.show()
```

```
Class Distribution Before SMOTE:
attack
0    46813
1    53964
Name: count, dtype: int64

Class Distribution After SMOTE:
attack
0    53964
1    53964
Name: count, dtype: int64
```

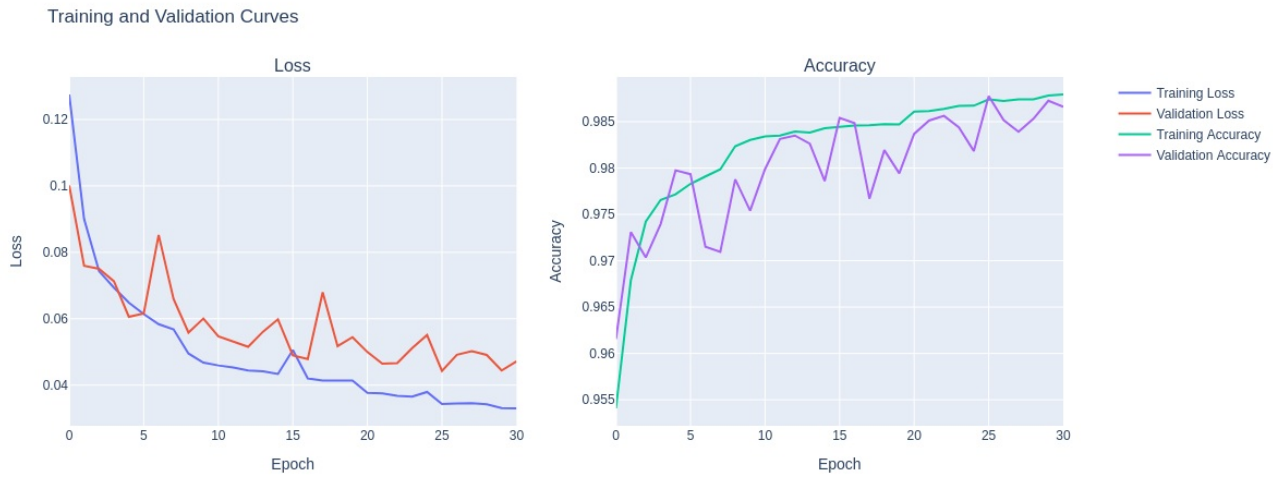
MODEL BUILD

In [ ]:

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
import numpy as np
import time
from sklearn.metrics import confusion_matrix, classification_report
from art.estimators.classification import PyTorchClassifier
from art.attacks.evasion import FastGradientMethod
import pandas as pd

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

class NeuralNetwork(nn.Module):
    def __init__(self, input_dim):
        super(NeuralNetwork, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.BatchNorm1d(128),
            nn.LeakyReLU(negative_slope=0.1),
            nn.Dropout(0.4)
        )
        self.layer2 = nn.Sequential(
            nn.Linear(128, 64),
            nn.BatchNorm1d(64),
            nn.LeakyReLU(negative_slope=0.1),
            nn.Dropout(0.3)
        )
        self.layer3 = nn.Sequential(
            nn.Linear(64, 32),
            nn.BatchNorm1d(32),
            nn.LeakyReLU(negative_slope=0.1),
            nn.Dropout(0.2)
        )
        self.output = nn.Linear(32, 2)

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.output(x)
        return x

class EarlyStopping:
    def __init__(self, patience=5, restore_best_weights=True):
        self.patience = patience
        self.restore_best_weights = restore_best_weights
        self.best_loss = float('inf')
        self.best_weights = None
        self.counter = 0
        self.early_stop = False

    def __call__(self, val_loss, model):
        if val_loss < self.best_loss:
            self.best_loss = val_loss
            self.counter = 0
            if self.restore_best_weights:
                self.best_weights = {k: v.cpu().clone() for k, v in model.state_dict().items()}
        else:
            self.counter += 1
            if self.counter >= self.patience:
                self.early_stop = True
                if self.restore_best_weights:
                    model.load_state_dict(self.best_weights)

class ReduceLROnPlateau:
    def __init__(self, factor=0.5, patience=3):
        self.factor = factor
        self.patience = patience
        self.best_loss = float('inf')
        self.counter = 0

    def __call__(self, val_loss, optimizer):
        if val_loss >= self.best_loss:
            self.counter += 1
            if self.counter >= self.patience:
                for param_group in optimizer.param_groups:
                    param_group['lr'] *= self.factor
                self.counter = 0
        else:
            self.best_loss = val_loss
            self.counter = 0
```

```python
def to_numpy_array(data):
    if isinstance(data, (pd.Series, pd.DataFrame)):
        return data.to_numpy()
    return data

X_train_scaled = to_numpy_array(X_train_scaled)
y_train_resampled = to_numpy_array(y_train_resampled)
X_test_scaled = to_numpy_array(X_test_scaled)
y_test = to_numpy_array(y_test)

y_train_resampled = y_train_resampled.astype(np.int64)
y_test = y_test.astype(np.int64)

X_train_tensor = torch.tensor(X_train_scaled, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train_resampled, dtype=torch.long)
X_test_tensor = torch.tensor(X_test_scaled, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.long)

val_size = int(0.2 * len(X_train_tensor))
X_val_tensor = X_train_tensor[-val_size:]
y_val_tensor = y_train_tensor[-val_size:]
X_train_tensor = X_train_tensor[:-val_size]
y_train_tensor = y_train_tensor[:-val_size]

train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
val_dataset = TensorDataset(X_val_tensor, y_val_tensor)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)

model = NeuralNetwork(input_dim=X_train_scaled.shape[1]).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

classifier = PyTorchClassifier(
    model=model,
    loss=criterion,
    optimizer=optimizer,
    input_shape=(X_train_scaled.shape[1],),
    nb_classes=2,
    clip_values=(0, 1)
)

attack = FastGradientMethod(estimator=classifier, eps=0.01)

def adversarial_training(model, train_loader, val_loader, epochs, callbacks, device):
    history = {'loss': [], 'accuracy': [], 'val_loss': [], 'val_accuracy': []}

    early_stopping = callbacks[0]
    reduce_lr = callbacks[1]

    for epoch in range(epochs):
        print(f'Epoch {epoch+1}/{epochs}')
        start_time = time.time()

        model.train()
        running_loss = 0.0
        correct = 0
        total = 0

        for X_batch, y_batch in train_loader:
            X_batch, y_batch = X_batch.to(device), y_batch.to(device)

            X_batch_adv = attack.generate(x=X_batch.cpu().numpy())
            X_batch_adv = torch.tensor(X_batch_adv, dtype=torch.float32).to(device)

            X_batch_combined = torch.cat([X_batch, X_batch_adv], dim=0)
            y_batch_combined = torch.cat([y_batch, y_batch], dim=0)

            optimizer.zero_grad()

            outputs = model(X_batch_combined)
            loss = criterion(outputs, y_batch_combined)

            loss.backward()
            optimizer.step()

            running_loss += loss.item() * X_batch_combined.size(0)
            _, preds = torch.max(outputs, 1)
            correct += (preds == y_batch_combined).sum().item()
            total += y_batch_combined.size(0)

        epoch_loss = running_loss / total
```

```
        epoch_acc = correct / total

        # Validation
        model.eval()
        val_loss = 0.0
        val_correct = 0
        val_total = 0
        with torch.no_grad():
            for X_val, y_val in val_loader:
                X_val, y_val = X_val.to(device), y_val.to(device)
                outputs = model(X_val)
                loss = criterion(outputs, y_val)
                val_loss += loss.item() * X_val.size(0)
                _, preds = torch.max(outputs, 1)
                val_correct += (preds == y_val).sum().item()
                val_total += y_val.size(0)

        val_loss = val_loss / val_total
        val_acc = val_correct / val_total

        history['loss'].append(epoch_loss)
        history['accuracy'].append(epoch_acc)
        history['val_loss'].append(val_loss)
        history['val_accuracy'].append(val_acc)

        print(f"loss: {epoch_loss:.4f} - accuracy: {epoch_acc:.4f} - "
              f"val_loss: {val_loss:.4f} - val_accuracy: {val_acc:.4f}")
        print(f"Epoch time: {time.time() - start_time:.2f} seconds")

        early_stopping(val_loss, model)
        reduce_lr(val_loss, optimizer)
        if early_stopping.early_stop:
            print("Early stopping triggered")
            break

    return history

early_stop = EarlyStopping(patience=5, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(factor=0.5, patience=3)

history = adversarial_training(model, train_loader, val_loader, epochs=50, callbacks=[early_stop, reduce_lr], d

model.eval()
X_test_tensor = X_test_tensor.to(device)
with torch.no_grad():
    outputs = model(X_test_tensor)
    _, y_pred_classes = torch.max(outputs, 1)
y_pred_classes = y_pred_classes.cpu().numpy()

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred_classes))

print("\nClassification Report:")
print(classification_report(y_test, y_pred_classes))
```



Model Save

Sniff Live Network Packets & GUI

In [ ]:

```
import os
import tkinter as tk
from tkinter import ttk
from tkinter.scrolledtext import ScrolledText
from threading import Thread
from queue import Queue
from scapy.all import sniff, IP, TCP
import numpy as np
import logging
import torch
import torch.nn as nn
from sklearn.preprocessing import MinMaxScaler
from queue import Queue

packet_queue = Queue()

# Setup logging
log_dir = r'D:\me\College\3rd year 2nd term\Ai sec issues\ids_project'
os.makedirs(log_dir, exist_ok=True)

for handler in logging.root.handlers[:]:
    logging.root.removeHandler(handler)

logging.basicConfig(
    filename=os.path.join(log_dir, 'packet_logs.txt'),
    level=logging.DEBUG,
    format='%(asctime)s - %(levelname)s - %(message)s'
)
logging.info("Logger initialized")

FEATURES = [
    'service', 'flag', 'src_bytes', 'dst_bytes', 'logged_in', 'count',
    'serror_rate', 'srv_serror_rate', 'same_srv_rate', 'diff_srv_rate',
    'dst_host_srv_count', 'dst_host_same_srv_rate',
    'dst_host_diff_srv_rate', 'dst_host_serror_rate',
    'dst_host_srv_serror_rate'
]

SERVICE_MAP = {'http': 0, 'ftp': 1, 'telnet': 2, 'other': 3}
FLAG_MAP = {'SF': 0, 'S0': 1, 'REJ': 2, 'RSTR': 3}
WINDOW_SIZE = 100

packet_window = []
dst_host_counts = {}
dst_host_srv_counts = {}

class NeuralNetwork(nn.Module):
    def __init__(self, input_dim):
        super(NeuralNetwork, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.BatchNorm1d(128),
            nn.LeakyReLU(negative_slope=0.1),
            nn.Dropout(0.4)
        )
        self.layer2 = nn.Sequential(
            nn.Linear(128, 64),
            nn.BatchNorm1d(64),
            nn.LeakyReLU(negative_slope=0.1),
            nn.Dropout(0.3)
        )
        self.layer3 = nn.Sequential(
            nn.Linear(64, 32),
            nn.BatchNorm1d(32),
            nn.LeakyReLU(negative_slope=0.1),
            nn.Dropout(0.2)
        )
        self.output = nn.Linear(32, 2)

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.output(x)
        return x

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
```

```python
try:
    input_dim = len(FEATURES)
    model = NeuralNetwork(input_dim=input_dim)
    model.load_state_dict(
    torch.load(r"D:\me\College\3rd year 2nd term\Ai sec issues\ids_project\adversarial_model.pth",
               map_location=device,
               weights_only=True)
)
    model.to(device)
    model.eval()
    logging.info("PyTorch model loaded successfully from 'adversarial_model.pth'")
except FileNotFoundError:
    logging.error("Error: 'adversarial_model.pth' not found in the working directory")
    raise
except Exception as e:
    logging.error(f"Failed to load PyTorch model: {e}")
    raise


scaler = MinMaxScaler()

def extract_features(packet):

    features = {f: 0 for f in FEATURES}

    if not packet.haslayer(IP):
        return None

    ip_layer = packet[IP]
    src_ip = ip_layer.src
    dst_ip = ip_layer.dst
    src_port = packet[TCP].sport if packet.haslayer(TCP) else 0
    dst_port = packet[TCP].dport if packet.haslayer(TCP) else 0

    # Basic features
    features['src_bytes'] = len(packet) if packet.haslayer('Raw') else 0
    features['dst_bytes'] = 0
    features['service'] = SERVICE_MAP.get('http' if dst_port == 80 else 'other', 3)
    features['flag'] = FLAG_MAP.get('SF', 0)
    features['logged_in'] = 1 if dst_port in [21, 22, 23] else 0


    packet_window.append((packet, dst_ip, dst_port))
    if len(packet_window) > WINDOW_SIZE:
        packet_window.pop(0)

    recent_dst_packets = [p for p in packet_window if p[1] == dst_ip]
    features['count'] = len(recent_dst_packets)


    serror_count = sum(1 for (pkt, _, _) in packet_window if pkt.haslayer(TCP) and pkt[TCP].flags & 0x02 and no
    features['serror_rate'] = serror_count / len(packet_window) if packet_window else 0


    same_service_packets = [pkt for (pkt, d_ip, d_port) in packet_window if d_port == dst_port]
    same_service_serror = sum(1 for pkt in same_service_packets if pkt.haslayer(TCP) and pkt[TCP].flags & 0x02 
    features['srv_serror_rate'] = same_service_serror / len(same_service_packets) if same_service_packets else 


    same_service_count = sum(1 for (_, d_ip, d_port) in packet_window if d_port == dst_port)
    features['same_srv_rate'] = same_service_count / len(packet_window) if packet_window else 0
    features['diff_srv_rate'] = (len(packet_window) - same_service_count) / len(packet_window) if packet_window


    dst_host_counts[dst_ip] = dst_host_counts.get(dst_ip, 0) + 1
    key = (dst_ip, dst_port)
    dst_host_srv_counts[key] = dst_host_srv_counts.get(key, 0) + 1
    features['dst_host_srv_count'] = dst_host_srv_counts[key]

    features['dst_host_same_srv_rate'] = features['dst_host_srv_count'] / dst_host_counts[dst_ip] if dst_host_c
    features['dst_host_diff_srv_rate'] = (dst_host_counts[dst_ip] - features['dst_host_srv_count']) / dst_host_


    dst_packets = [pkt for (pkt, d_ip, _) in packet_window if d_ip == dst_ip]
    dst_serror = sum(1 for pkt in dst_packets if pkt.haslayer(TCP) and pkt[TCP].flags & 0x02 and not pkt[TCP].f
    features['dst_host_serror_rate'] = dst_serror / len(dst_packets) if dst_packets else 0


    dst_srv_packets = [pkt for (pkt, d_ip, d_port) in packet_window if d_ip == dst_ip and d_port == dst_port]
    dst_srv_serror = sum(1 for pkt in dst_srv_packets if pkt.haslayer(TCP) and pkt[TCP].flags & 0x02 and not pk
    features['dst_host_srv_serror_rate'] = dst_srv_serror / len(dst_srv_packets) if dst_srv_packets else 0

    return list(features.values())
```

```
def analyze_packet(packet):
    try:
        features = extract_features(packet)
        if features is None:
            return


        feature_array = np.array([features])
        feature_array = scaler.fit_transform(feature_array)  # Normalize to [0,1]
        feature_tensor = torch.tensor(feature_array, dtype=torch.float32).to(device)


        with torch.no_grad():
            outputs = model(feature_tensor)
            probabilities = torch.softmax(outputs, dim=1)
            prediction = probabilities[0][1].item()  # Probability of "Attack" (class 1)
            label = 'Attack' if prediction > 0.5 else 'Normal'

        src = packet[IP].src if IP in packet else "Unknown"
        dst = packet[IP].dst if IP in packet else "Unknown"
        proto = packet[IP].proto if IP in packet else "Unknown"
        log_msg = f"Src: {src}, Dst: {dst}, Protocol: {proto}, Prediction: {label})"
        logging.info(log_msg)

        packet_queue.put(log_msg)
    except Exception as e:
        logging.error(f"Error analyzing packet: {e}")
        packet_queue.put(f"Error analyzing packet: {e}")

def update_gui():
    """Update the GUI with new packet analysis results."""
    while not packet_queue.empty():
        log_msg = packet_queue.get()
        output_text.insert(tk.END, log_msg + "\n")
        output_text.see(tk.END)
    window.after(100, update_gui)

def start_sniffing():
    """Start packet sniffing in a separate thread."""
    global sniffing
    sniffing = True
    try:
        sniff(prn=analyze_packet, store=False, stop_filter=lambda x: not sniffing)
    except Exception as e:
        logging.error(f"Sniffing error: {e}")
        packet_queue.put(f"Sniffing error: {e}")

def on_start():
    """Handle start button click."""
    global sniffer_thread
    start_button.config(state='disabled')
    stop_button.config(state='normal')
    status_label.config(text="Status: Sniffing Active", foreground="#2ecc71")
    sniffer_thread = Thread(target=start_sniffing, daemon=True)
    sniffer_thread.start()

def on_stop():
    """Handle stop button click."""
    global sniffing
    sniffing = False
    start_button.config(state='normal')
    stop_button.config(state='disabled')
    packet_queue.put("Sniffing stopped.")
    status_label.config(text="Status: Sniffing Stopped", foreground="#e74c3c")

# GUI Setup
window = tk.Tk()
window.title("Real-time Packet Analyzer")
window.geometry("900x600")
window.configure(bg="#2c3e50")
window.resizable(True, True)

style = ttk.Style()
style.theme_use('clam')
style.configure("TButton", font=("Helvetica", 12, "bold"), padding=10)
style.map("TButton",
          background=[('active', '#3498db'), ('disabled', '#95a5a6')],
          foreground=[('active', '#ffffff'), ('disabled', '#d9d9d9')])

main_frame = tk.Frame(window, bg="#2c3e50", padx=20, pady=20)
main_frame.pack(fill="both", expand=True)
```

```
title_label = tk.Label(
    main_frame,
    text="Real-time Packet Analyzer",
    font=("Helvetica", 24, "bold"),
    fg="#ecf0f1",
    bg="#2c3e50"
)
title_label.pack(pady=10)

status_label = tk.Label(
    main_frame,
    text="Status: Idle",
    font=("Helvetica", 12),
    fg="#f1c40f",
    bg="#2c3e50"
)
status_label.pack(pady=5)

button_frame = tk.Frame(main_frame, bg="#2c3e50")
button_frame.pack(pady=10)

start_button = ttk.Button(
    button_frame,
    text="Start Sniffing",
    command=on_start,
    style="TButton",
    width=15
)
start_button.pack(side=tk.LEFT, padx=5)

stop_button = ttk.Button(
    button_frame,
    text="Stop Sniffing",
    command=on_stop,
    state="disabled",
    style="TButton",
    width=15
)
stop_button.pack(side=tk.LEFT, padx=5)

output_frame = tk.Frame(main_frame, bg="#34495e", bd=2, relief="flat")
output_frame.pack(fill="both", expand=True, padx=10, pady=10)
output_text = ScrolledText(
    output_frame,
    height=20,
    width=80,
    font=("Consolas", 10),
    bg="#ecf0f1",
    fg="#2c3e50",
    insertbackground="#3498db",
    relief="flat",
    wrap=tk.WORD
)
output_text.pack(padx=10, pady=10, fill="both", expand=True)

window.after(100, update_gui)

def on_closing():
    """Handle window closing."""
    global sniffing
    sniffing = False
    window.destroy()

window.protocol("WM_DELETE_WINDOW", on_closing)
window.mainloop()
```

```
Using device: cpu
```