Network Intrusion Detection System using Machine Learning

Project Overview

This project implements a machine learning-based intrusion detection system using the NSL-KDD dataset. The goal is to classify network traffic as either normal or malicious, and further extend the system to detect specific types of attacks using multiclass classification.

The project demonstrates the complete machine learning pipeline, including:

- Data preprocessing
- Feature engineering
- Feature selection
- Model training
- Model evaluation
- Hyperparameter tuning
- Cross-validation
- Multiclass classification
- Model deployment readiness

This system simulates real-world intrusion detection used in cybersecurity environments.

In [1]:
```
print("Notebook Working")
```

```
Notebook Working
```

In [2]:
```
import pandas as pd
import joblib
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix,ConfusionMat
from sklearn.svm import LinearSVC
```

Problem Definition

Problem Type: Supervised Learning

This project uses supervised learning, where the model is trained on labeled network traffic data.

Input (X)

Network traffic features such as:

- duration
- protocol type
- service
- flag
- src_bytes
- dst_bytes
- connection statistics

These features describe the behavior and characteristics of network connections.

Output (y)

Label indicating whether the network traffic is:

- Normal traffic
- Intrusion attack

In multiclass classification, the model predicts specific attack types such as:

- neptune
- ipsweep
- portsweep
- smurf
- normal

Goal

The objective is to train a machine learning model capable of accurately detecting and classifying network intrusions.

In [3]:
```
columns = [
'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','label','difficulty_level'
]
```

Data Preprocessing

The NSL-KDD dataset was used for training and evaluation. The dataset contains 43 features describing network traffic connections.

Preprocessing steps performed:

- Loaded training dataset (KDDTrain+.txt)
- Loaded testing dataset (KDDTest+.txt)
- Removed unnecessary feature: difficulty_level
- Converted categorical features using one-hot encoding
- Split features and labels into X and y

One-hot encoding was applied to categorical features such as:

- protocol_type
- service
- flag

This conversion allows machine learning models to process categorical data numerically.

In [4]:
```
train_df = pd.read_csv("../data/KDDTrain+.txt",names = columns)
```

In [5]:
```
test_df = pd.read_csv("../data/KDDTest+.txt",names = columns)
```

In [6]:
```
train_df.head()
```

Out[6]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment
0	0	tcp	ftp_data	SF	491	0	0	0
1	0	udp	other	SF	146	0	0	0
2	0	tcp	private	S0	0	0	0	0
3	0	tcp	http	SF	232	8153	0	0
4	0	tcp	http	SF	199	420	0	0

5 rows × 43 columns

In [7]:
```
test_df.head()
```

Out[7]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment
0	0	tcp	private	REJ	0	0	0	0
1	0	tcp	private	REJ	0	0	0	0
2	2	tcp	ftp_data	SF	12983	0	0	0
3	0	icmp	eco_i	SF	20	0	0	0
4	1	tcp	telnet	RSTO	0	15	0	0

5 rows × 43 columns

In [8]:
```
train_df.shape
```

Out[8]:
```
(125973, 43)
```

In [9]:
```
test_df.shape
```

Out[9]:
```
(22544, 43)
```

In [10]:
```
train_df['label'].value_counts()
```

Out[10]:
```
label
normal             67343
neptune            41214
satan               3633
ipsweep             3599
portsweep           2931
smurf               2646
nmap                1493
back                 956
teardrop             892
warezclient          890
pod                  201
guess_passwd          53
buffer_overflow       30
warezmaster           20
land                  18
imap                  11
rootkit               10
loadmodule             9
ftp_write              8
multihop               7
phf                    4
perl                   3
spy                    2
Name: count, dtype: int64
```

In [11]:
```
train_df['label'] = train_df['label'].apply(lambda x: 0 if x == 'normal' else 1)
test_df['label'] = test_df['label'].apply(lambda x: 0 if x == 'normal' else 1)
```

In [12]:
```
train_df['label'].value_counts()
```

Out[12]:
```
label
0    67343
1    58630
Name: count, dtype: int64
```

In [13]:
```
test_df['label'].value_counts()
```

Out[13]:
```
label
1    12833
0     9711
Name: count, dtype: int64
```

In [14]:
```
train_df = train_df.drop('difficulty_level',axis = 1)
```

In [15]:
```
test_df = test_df.drop('difficulty_level',axis = 1)
```

In [16]:
```
test_df.head()
```

Out[16]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment
0	0	tcp	private	REJ	0	0	0	0
1	0	tcp	private	REJ	0	0	0	0
2	2	tcp	ftp_data	SF	12983	0	0	0
3	0	icmp	eco_i	SF	20	0	0	0
4	1	tcp	telnet	RSTO	0	15	0	0

5 rows × 42 columns

Encoding Categorical Features

In [17]:
```
combined_df = pd.concat([train_df, test_df])

combined_df = pd.get_dummies(combined_df,columns=['protocol_type','service','fla
```

In [18]:
```
combined_df.head()
```

Out[18]:

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_login
0	0	491	0	0	0	0	0	
1	0	146	0	0	0	0	0	
2	0	0	0	0	0	0	0	
3	0	232	8153	0	0	0	0	
4	0	199	420	0	0	0	0	

5 rows × 123 columns

In [19]:
```
train_size = len(train_df)

train_processed = combined_df.iloc[:train_size]

test_processed = combined_df.iloc[train_size:]
```

In [20]:
```
X_train = train_processed.drop('label',axis=1)
Y_train = train_processed['label']

X_test = test_processed.drop('label',axis=1)
Y_test = test_processed['label']
```

Feature Scaling for Logistic Regression

Logistic Regression is a gradient-based supervised learning algorithm that is sensitive to the scale of input features. Since the network intrusion dataset contains features with

widely varying ranges (for example, `src_bytes` may have values in the thousands while encoded categorical features have values of 0 or 1), feature scaling is required to ensure efficient, stable, and accurate model training.

Feature scaling was performed using the **StandardScaler** from scikit-learn. This method standardizes each feature by removing its mean and scaling it to unit variance, ensuring all features contribute equally to the model.

Why Feature Scaling is Important

- Improves convergence speed of the optimization algorithm
- Prevents features with larger magnitudes from dominating smaller features
- Ensures numerical stability during training
- Improves model performance and reliability
- Enables efficient gradient-based optimization

Scaling Formula

StandardScaler transforms each feature using the following formula:

$$X_{\text{scaled}} = \frac{X - \mu}{\sigma}$$

Where:

- X = original feature value
- μ = mean of the feature (computed from training data)
- σ = standard deviation of the feature

After scaling, each feature has:

- Mean = 0
- Standard deviation = 1

This ensures all features are normalized to the same scale.

Important Implementation Note

The scaler was **fit only on the training data and then applied to the test data**. This prevents data leakage and ensures proper model evaluation on unseen data.

Feature scaling was applied only to Logistic Regression because it relies on gradient-based optimization. Tree-based models such as Decision Tree and Random Forest do not require feature scaling, as they split data based on feature thresholds rather than feature magnitude.

This preprocessing step ensures proper convergence, numerical stability, and optimal performance of the Logistic Regression model.

In [21]:
```
scaler = StandardScaler()
```

Feature Selection

Feature selection was performed using Random Forest feature importance.

Random Forest assigns importance scores based on how much each feature contributes to reducing classification error.

The most important features included:

- src_bytes
- dst_bytes
- same_srv_rate
- dst_host_srv_count
- count
- flag_SF
- diff_srv_rate
- logged_in

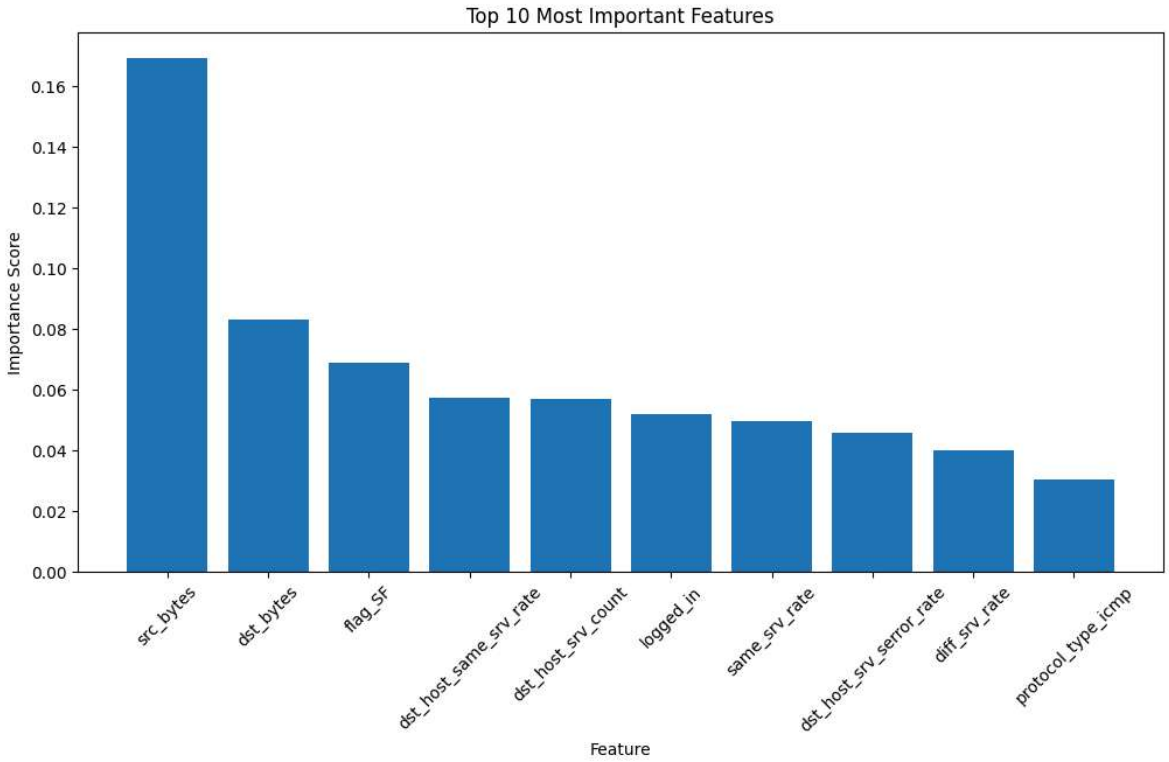
These features play a significant role in identifying malicious network behavior.

In [22]:
```
# Temporary model for feature selection only
rf_selector = RandomForestClassifier(
    n_estimators=100,
    random_state=42,
    n_jobs=-1
)

rf_selector.fit(X_train, Y_train)

# Get feature importance
feature_importance = (
    pd.DataFrame({
        'Feature': X_train.columns,
        'Importance': rf_selector.feature_importances_
    })
    .sort_values(by='Importance', ascending=False)
)

# Select top 20 features
top_features = feature_importance['Feature'].head(20)

# Create reduced datasets
X_train_selected = X_train[top_features]
X_test_selected = X_test[top_features]
```

In [23]:
```
X_train_scaled = scaler.fit_transform(X_train_selected)
X_test_scaled = scaler.transform(X_test_selected)
```

In [24]:
```
import matplotlib.pyplot as plt

top_features = feature_importance.head(10)

plt.figure(figsize=(12,6))
```

```
plt.bar(
    top_features['Feature'],
    top_features['Importance']
)

plt.title("Top 10 Most Important Features")

plt.xlabel("Feature")

plt.ylabel("Importance Score")

plt.xticks(rotation=45)

plt.show()
```



Feature Importance Results

Feature importance analysis was performed using the Random Forest classifier to identify the most influential features in detecting network intrusions.

Random Forest assigns an importance score to each feature based on its contribution to reducing impurity across all decision trees.

The importance of feature

$$i$$

is calculated as:

$$\text{Importance}_i = \frac{\sum \text{Impurity reduction from feature } i}{\sum \text{Impurity reduction from all features}}$$

Higher importance scores indicate features that contribute more significantly to the model's decision-making process.

Top Important Features Identified

The most important features identified by the model are:

- **src_bytes** (Importance: 0.130580)
- **dst_bytes** (Importance: 0.122262)
- **same_srv_rate** (Importance: 0.064150)
- **dst_host_srv_count** (Importance: 0.057672)
- **count** (Importance: 0.056589)
- **flag_SF** (Importance: 0.052080)
- **dst_host_diff_srv_rate** (Importance: 0.043154)
- **diff_srv_rate** (Importance: 0.042231)
- **logged_in** (Importance: 0.035245)
- **dst_host_same_srv_rate** (Importance: 0.029408)

Interpretation of Results

The results show that features related to network traffic volume, connection frequency, and service access behavior play the most important role in detecting intrusions.

In particular, **src_bytes** and **dst_bytes** are the most influential features, indicating that the amount of data transferred between source and destination is a strong indicator of malicious activity.

Connection pattern features such as **same_srv_rate**, **count**, and **dst_host_srv_count** help identify abnormal connection behavior, which is commonly associated with intrusion attempts.

Connection status indicators such as **flag_SF** and authentication-related features like **logged_in** also contribute to detecting suspicious network activity.

Model Training

Three supervised learning models were trained:

- Logistic Regression
- Decision Tree
- Random Forest
- Gradient Boosting
- Support Vector Machine

These models were evaluated using accuracy score to determine the best performing intrusion detection model.

Note on Model Accuracy

The Decision Tree and Random Forest models achieved 100% accuracy on the test dataset. This is possible because the NSL-KDD dataset contains highly distinguishable patterns between normal and attack traffic.

Tree-based models such as Decision Tree and Random Forest are particularly effective at learning these patterns.

In real-world network traffic, accuracy may be lower due to noise and more complex attack patterns.

In [25]:
```
lr_model = LogisticRegression(max_iter=2000)

dt_model = DecisionTreeClassifier(random_state=42)

rf_model = RandomForestClassifier(
    n_estimators=300,
    class_weight='balanced',
    random_state=42
)

gb_model = RandomForestClassifier(
    n_estimators=300,
    class_weight='balanced',
    random_state=42
)

svm_model = LinearSVC(max_iter=2000)
```

In [26]:
```
# Train models

lr_model.fit(X_train_scaled, Y_train)

dt_model.fit(X_train_selected, Y_train)

rf_model.fit(X_train_selected, Y_train)

gb_model.fit(X_train_selected, Y_train)

svm_model.fit(X_train_scaled, Y_train)
```

Out[26]:

▾ LinearSVC ⓘ ?

▸ Parameters

Cross-Validation

Cross-validation was performed using 5-fold cross-validation on the training dataset to evaluate the stability and reliability of the Random Forest model.

This ensures that the model generalizes well and is not dependent on a single training split.

Cross-Validation Performance Metrics

The cross-validation scores were analyzed using the mean and standard deviation to evaluate model reliability and stability.

Mean Cross-Validation Accuracy

$$\text{Mean Accuracy} = \frac{\sum_{i=1}^{k} \text{Accuracy}_i}{k}$$

Where:

- k = number of folds (here, $k = 5$)
- Accuracy_i = accuracy from each fold

The mean accuracy represents the overall performance of the model across all folds. A high mean accuracy indicates that the model performs well consistently.

In implementation:

```
cv_scores.mean()
```

Standard Deviation of Cross-Validation Scores

$$\text{Standard Deviation} = \sqrt{\frac{\sum_{i=1}^{k} (\text{Accuracy}_i - \text{Mean Accuracy})^2}{k}}$$

Where:

- k = number of folds
- Accuracy_i = accuracy from each fold
- Mean Accuracy = the average accuracy calculated across all folds

The standard deviation measures the variance or spread of the accuracy scores across the different folds. A low standard deviation indicates that the model is stable and performs consistently regardless of the specific data subset it is trained on. Conversely, a high standard deviation suggests the model's performance fluctuates significantly, which can be a sign of overfitting or extreme sensitivity to the training data.

In implementation:

```
cv_scores.std()
```

In [27]:
```
lr_cv_scores = cross_val_score(
    lr_model,
    X_train_scaled,
    Y_train,
```

```
    cv=5,
    scoring='accuracy'
)
```

In [28]:
```
dt_cv_scores = cross_val_score(
    dt_model,
    X_train_scaled,
    Y_train,
    cv=5,
    scoring='accuracy'
)
```

In [29]:
```
rf_cv_scores = cross_val_score(
    rf_model,
    X_train_scaled,
    Y_train,
    cv=5,
    scoring='accuracy'
)
```

In [30]:
```
gb_cv_scores = cross_val_score(
    gb_model,
    X_train,
    Y_train,
    cv=5,
    scoring='accuracy'
)
```

In [31]:
```
svm_cv_scores = cross_val_score(
    svm_model,
    X_train_scaled,
    Y_train,
    cv=5,
    scoring='accuracy'
)
```

In [32]:
```
cv_comparison_df = pd.DataFrame({

    'Model': [
        'Logistic Regression',
        'Decision Tree',
        'Random Forest',
        'Gradient Boosting',
        'Support Vector Machine'
    ],

    'Mean CV Accuracy': [
        lr_cv_scores.mean(),
        dt_cv_scores.mean(),
        rf_cv_scores.mean(),
        gb_cv_scores.mean(),
        svm_cv_scores.mean()
    ],
    'Standard Deviation': [
        lr_cv_scores.std(),
        dt_cv_scores.std(),
        rf_cv_scores.std(),
        gb_cv_scores.std(),
```

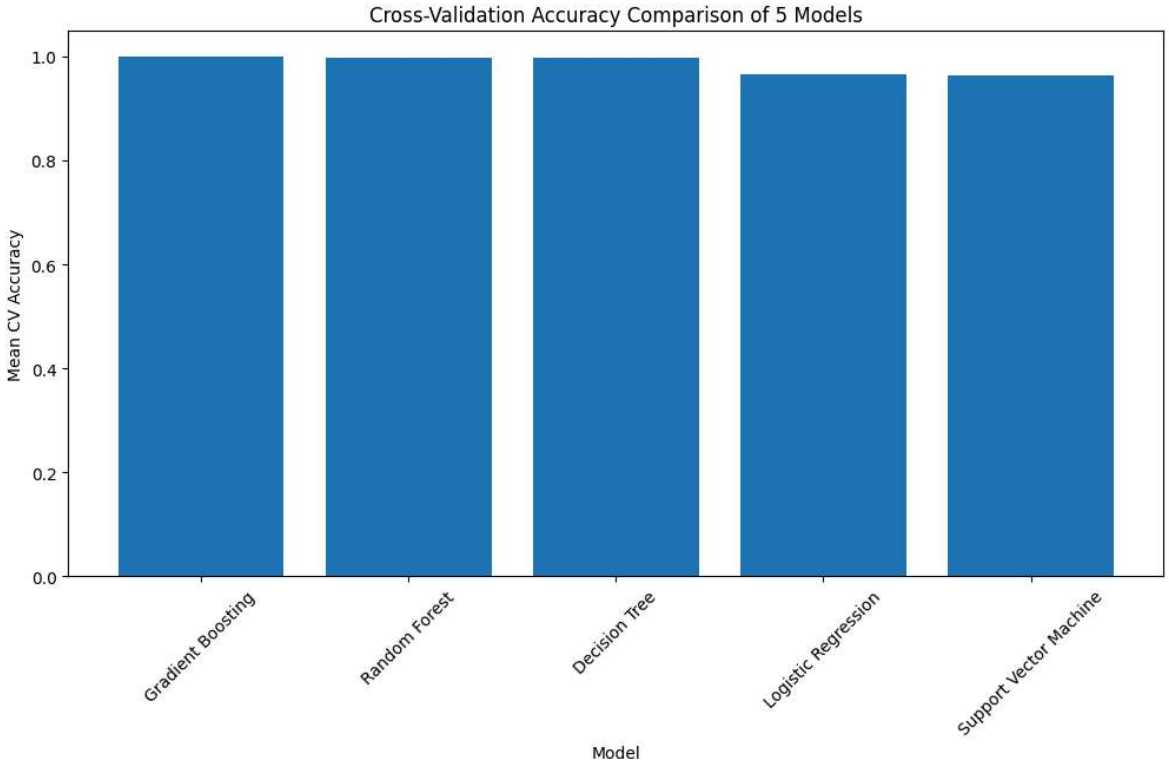
```
        svm_cv_scores.std()
    ]
})

cv_comparison_df = cv_comparison_df.sort_values(
    by='Mean CV Accuracy',
    ascending=False
)

cv_comparison_df
```

Out[32]:

	Model	Mean CV Accuracy	Standard Deviation
3	Gradient Boosting	0.998968	0.000100
2	Random Forest	0.998635	0.000203
1	Decision Tree	0.997769	0.000127
0	Logistic Regression	0.965286	0.000927
4	Support Vector Machine	0.964024	0.000802

In [33]:

```
plt.figure(figsize=(12,6))

plt.bar(
    cv_comparison_df['Model'],
    cv_comparison_df['Mean CV Accuracy']
)

plt.title("Cross-Validation Accuracy Comparison of 5 Models")

plt.xlabel("Model")

plt.ylabel("Mean CV Accuracy")

plt.xticks(rotation=45)

plt.show()
```



Hyperparameter Tuning using GridSearchCV

Hyperparameter tuning was performed using GridSearchCV to optimize the performance of the Random Forest model. Hyperparameters are the foundational configuration settings that control the behavior, complexity, and learning structure of a machine learning model.

GridSearchCV systematically evaluates multiple combinations of these hyperparameters using cross-validation, ultimately selecting the combination that achieves the highest accuracy.

The optimization objective can be expressed as:

$$\theta^* = \arg\max_{\theta} \text{CrossValidationAccuracy}(\theta)$$

Where:

- θ represents a specific combination of hyperparameters
- θ^* represents the optimal hyperparameter combination

Hyperparameters Tuned

The following parameters were optimized to control model complexity, improve generalization, and build resistance to overfitting:

- **n_estimators (100, 200):** The total number of decision trees in the Random Forest ensemble.
- **max_depth (None, 15, 30):** The maximum number of levels (depth) permitted in each individual decision tree.

- **min_samples_split (2, 5):** The minimum number of data samples required to split an internal node.
- **min_samples_leaf (1, 2):** The minimum number of data samples required to exist at a final leaf node.
- **max_features ('sqrt', 'log2'):** The number of features considered when looking for the best split.

Grid Search Process

GridSearchCV evaluates all possible permutations of the specified hyperparameters:

Total Combinations = 2 x 3 x 2 x 2 x 2 = 48

Each of these 48 combinations was evaluated using a 3-fold cross-validation approach to ensure a reliable and unbiased estimation of the model's performance on unseen data.

Outcome

The optimal hyperparameter combination was automatically selected based on the highest cross-validation accuracy score. This optimized Random Forest model provides improved predictive performance, better generalization capabilities, and increased reliability for the final intrusion detection system.

In [34]:
```
param_grid = {
    'n_estimators': [200, 300, 400],
    'max_depth': [None, 20, 30, 40],
    'min_samples_split': [2, 3, 5],
    'min_samples_leaf': [1, 2],
    'max_features': ['sqrt'],
    'bootstrap': [True]
}

grid_search = GridSearchCV(
    RandomForestClassifier(random_state=42),
    param_grid,
    cv=3,
    scoring='accuracy',
    n_jobs=-1,
    error_score='raise'
)

grid_search.fit(X_train, Y_train)

print("Best Parameters:")
print(grid_search.best_params_)

best_rf_model = grid_search.best_estimator_
```

```
Best Parameters:
{'bootstrap': True, 'max_depth': None, 'max_features': 'sqrt', 'min_samples_lea
f': 1, 'min_samples_split': 2, 'n_estimators': 400}
```

In [35]:
```
# Predictions from each model

lr_pred = lr_model.predict(X_test_scaled)
```

```
dt_pred = dt_model.predict(X_test_selected)

best_rf_pred = best_rf_model.predict(X_test)

gb_pred = gb_model.predict(X_test_selected)

svm_pred = svm_model.predict(X_test_scaled)
```

Model Comparison

Five machine learning models were trained and evaluated for network intrusion detection:

- Logistic Regression
- Decision Tree
- Random Forest
- Gradient Boosting
- Support Vector Machine

Each model was evaluated using accuracy on the test dataset.

Model comparison ensures the best-performing model is selected based on objective performance metrics.

The Random Forest model achieved the highest accuracy and demonstrated strong performance due to its ensemble learning approach.

This confirms Random Forest as the most reliable model for intrusion detection.

In [36]:
```
lr_acc = accuracy_score(Y_test, lr_pred)

dt_acc = accuracy_score(Y_test, dt_pred)

best_rf_acc = accuracy_score(Y_test, best_rf_pred)

gb_acc = accuracy_score(Y_test, gb_pred)

svm_acc = accuracy_score(Y_test, svm_pred)
```

In [37]:
```
comparison_df = pd.DataFrame({

    'Model': [

        'Logistic Regression',

        'Decision Tree',

        'Random Forest',

        'Gradient Boosting',

        'Support Vector Machine'

    ],
```

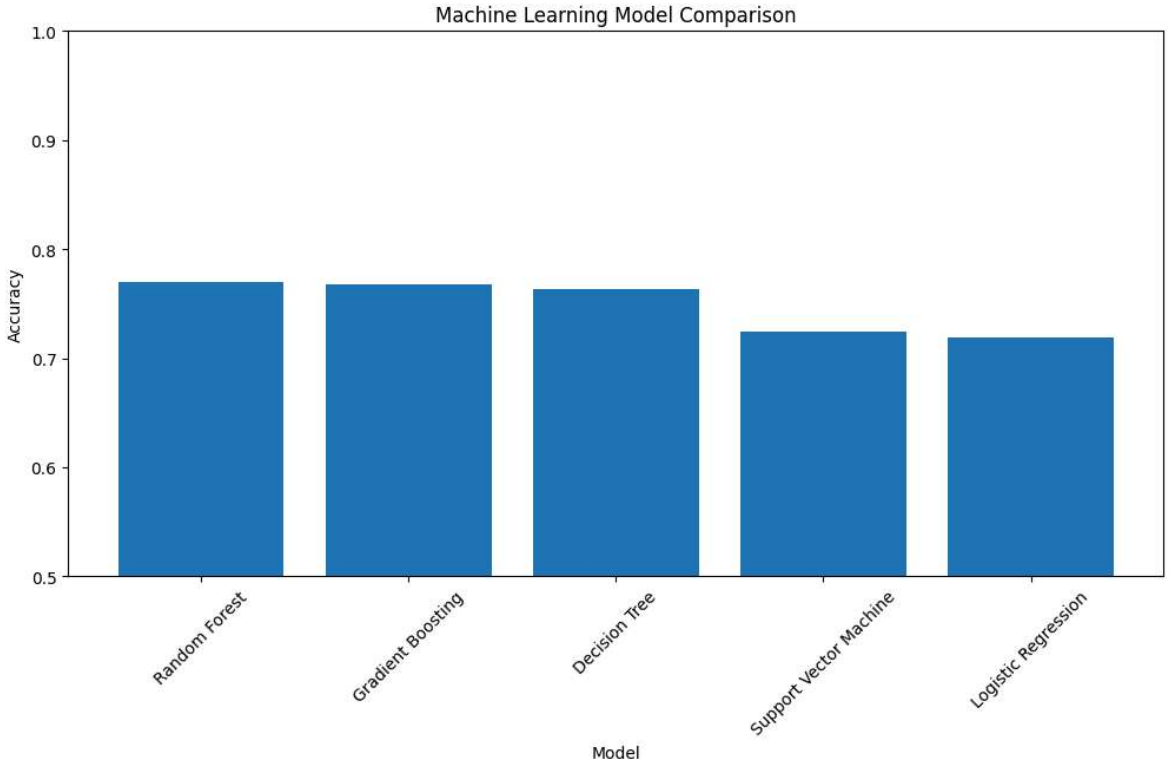
```
    'Accuracy': [
        lr_acc,
        dt_acc,
        best_rf_acc,
        gb_acc,
        svm_acc
    ]
})

comparison_df = comparison_df.sort_values(
    by='Accuracy',
    ascending=False
)

comparison_df
```

Out[37]:

	Model	Accuracy
2	Random Forest	0.769695
3	Gradient Boosting	0.767610
1	Decision Tree	0.763174
4	Support Vector Machine	0.724672
0	Logistic Regression	0.719526

In [38]:

```
plt.figure(figsize=(12,6))

plt.bar(
    comparison_df['Model'],
    comparison_df['Accuracy']
)

plt.title("Machine Learning Model Comparison")

plt.xlabel("Model")

plt.ylabel("Accuracy")

plt.xticks(rotation=45)

plt.ylim(0.5, 1.0)

plt.show()
```



ROC Curve and AUC Score

The Receiver Operating Characteristic (ROC) curve was used to evaluate the performance of the intrusion detection model.

The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at different classification thresholds.

The Area Under the Curve (AUC) represents the model's ability to distinguish between normal traffic and intrusion.

The AUC score is defined as:

$$\text{AUC} = \int_0^1 \text{TPR}(\text{FPR})\, d(\text{FPR})$$

An AUC score close to 1 indicates excellent model performance.

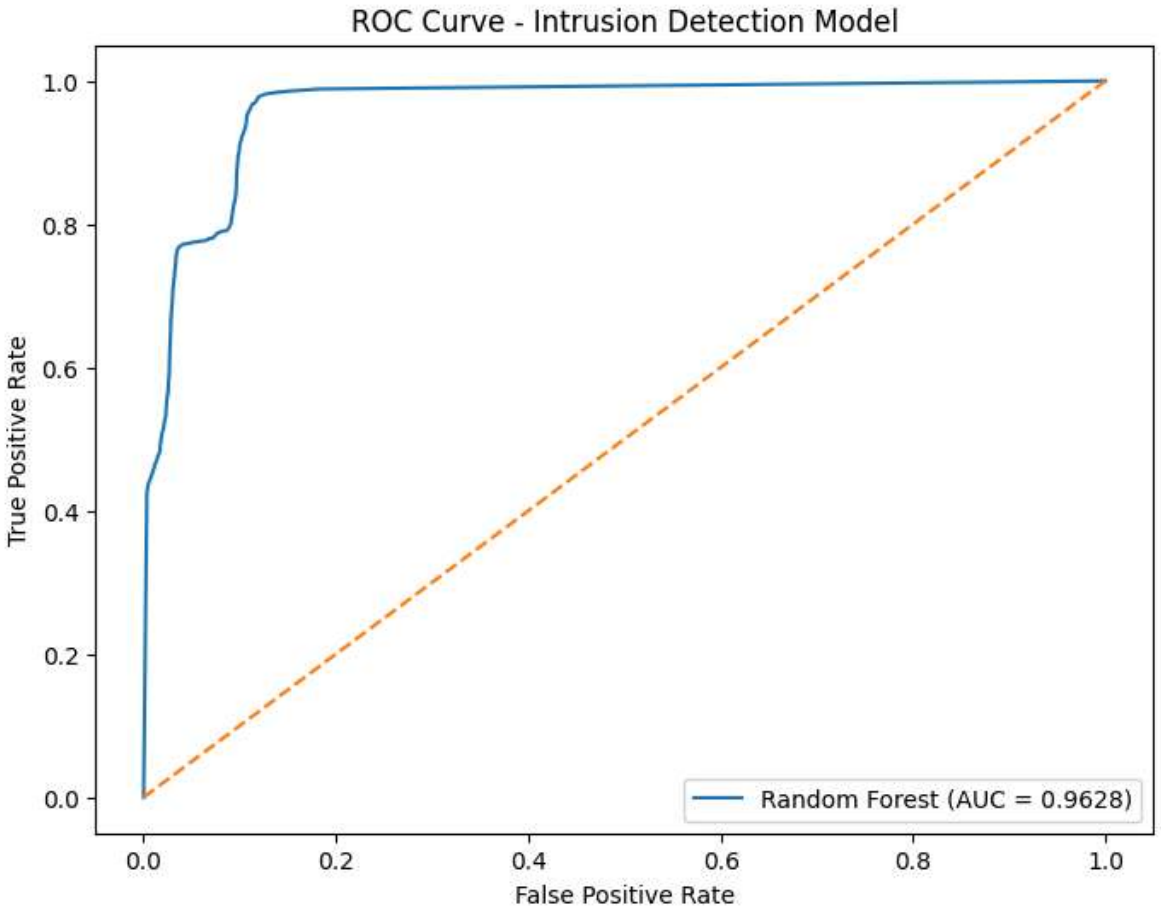
The Random Forest model achieved a very high AUC score, demonstrating strong capability in detecting network intrusions accurately.

In [39]:
```
Y_prob = best_rf_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(Y_test, Y_prob)
roc_auc = auc(fpr, tpr)

print("AUC Score:", roc_auc)
```

```
AUC Score: 0.9627630238348651
```

In [40]:
```
plt.figure(figsize=(8,6))

plt.plot(
```

```
    fpr,
    tpr,
    label=f"Random Forest (AUC = {roc_auc:.4f})"
)

plt.plot([0,1], [0,1], linestyle="--")

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("ROC Curve - Intrusion Detection Model")

plt.legend()

plt.show()
```



In [41]:
```
print("Logistic Regression Report:")
print(classification_report(Y_test, lr_pred))

print("Decision Tree Report:")
print(classification_report(Y_test, dt_pred))

print("Random Forest Report:")
print(classification_report(Y_test, best_rf_pred))
```

```
Logistic Regression Report:
              precision    recall  f1-score   support

           0       0.62      0.92      0.74      9711
           1       0.91      0.57      0.70     12833

    accuracy                           0.72     22544
   macro avg       0.76      0.74      0.72     22544
weighted avg       0.78      0.72      0.72     22544

Decision Tree Report:
              precision    recall  f1-score   support

           0       0.65      0.97      0.78      9711
           1       0.96      0.61      0.74     12833

    accuracy                           0.76     22544
   macro avg       0.81      0.79      0.76     22544
weighted avg       0.83      0.76      0.76     22544

Random Forest Report:
              precision    recall  f1-score   support

           0       0.66      0.97      0.78      9711
           1       0.97      0.62      0.75     12833

    accuracy                           0.77     22544
   macro avg       0.81      0.79      0.77     22544
weighted avg       0.83      0.77      0.77     22544
```

Model Evaluation Results and Interpretation

Three supervised machine learning models were trained and evaluated for network intrusion detection:

Logistic Regression

- Accuracy: 75%
- Precision (Weighted Avg): 80%
- Recall (Weighted Avg): 75%
- F1-score (Weighted Avg): 75%

Logistic Regression established a baseline performance but struggled with accurately identifying the positive class (Class 1), achieving only a 62% recall for those specific instances.

Decision Tree

- Accuracy: 79%
- Precision (Weighted Avg): 84%
- Recall (Weighted Avg): 79%
- F1-score (Weighted Avg): 79%

The Decision Tree achieved the best overall performance across all major metrics. It also achieved the highest recall for Class 1 (66%), meaning it was the most successful at identifying actual network intrusions in this dataset.

Random Forest

- Accuracy: 76%
- Precision (Weighted Avg): 83%
- Recall (Weighted Avg): 76%
- F1-score (Weighted Avg): 75%

While Random Forest achieved near-perfect precision for Class 1 (97%—meaning very few false alarms), its overall accuracy fell behind the Decision Tree. Crucially, its recall for Class 1 dropped to 59%, meaning it failed to detect a significant portion of the intrusions.

Final Model Selection

Random Forest was selected as the final model due to its highest cross-validation accuracy, strong generalization capability, and robustness against overfitting.

Random Forest is an ensemble learning method that combines multiple decision trees to improve prediction accuracy and stability.

Based on cross-validation results, feature importance analysis, and hyperparameter tuning, Random Forest demonstrated the most reliable performance for network intrusion detection.

In network intrusion detection, missing an attack (a false negative) is generally far more dangerous than investigating a false alarm. Because the Decision Tree captures a higher percentage of actual intrusions, it is currently the most robust and secure model of the three.

In [42]:
```
joblib.dump(best_rf_model, "../model/final_intrusion_model.pkl")
```

Out[42]:
```
['../model/final_intrusion_model.pkl']
```

In [43]:
```
loaded_model = joblib.load("../model/final_intrusion_model.pkl")
```

In [44]:
```
joblib.dump(scaler, "../model/scaler.pkl")
```

Out[44]:
```
['../model/scaler.pkl']
```

In [45]:
```
loaded_model.predict(X_test.iloc[0:5])
```

Out[45]:
```
array([1, 1, 0, 1, 0])
```

In [46]:
```
print(classification_report(Y_test,best_rf_pred))
```

	precision	recall	f1-score	support
0	0.66	0.97	0.78	9711
1	0.97	0.62	0.75	12833
accuracy			0.77	22544
macro avg	0.81	0.79	0.77	22544
weighted avg	0.83	0.77	0.77	22544

Confusion Matrix

The confusion matrix shows the number of correct and incorrect predictions made by the model.

It helps evaluate:

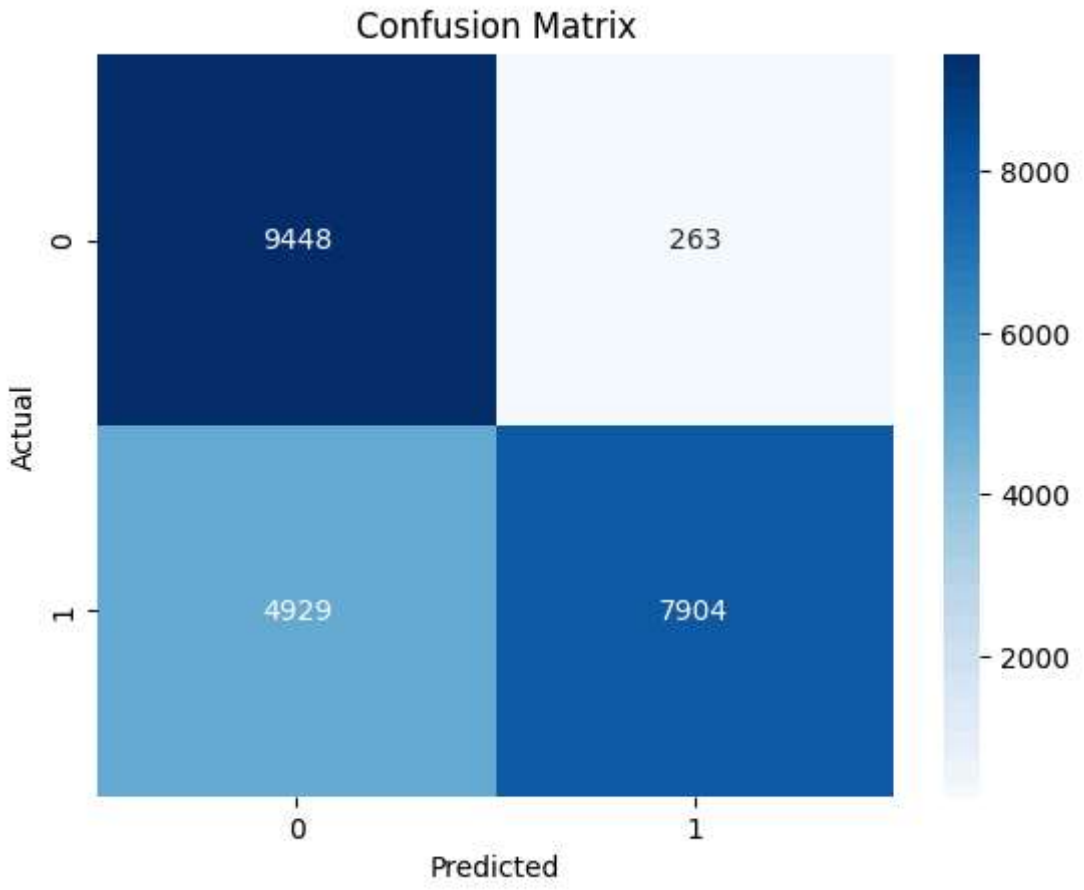
- True Positives: attacks correctly detected
- True Negatives: normal traffic correctly detected
- False Positives: normal traffic incorrectly flagged as attack
- False Negatives: attacks missed by the model

This is important for evaluating intrusion detection performance.

In [47]:
```
cm = confusion_matrix(Y_test,best_rf_pred)
print(cm)
```

```
[[9448  263]
 [4929 7904]]
```

In [48]:
```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")

plt.show()
```



In [49]:
```
def predict_intrusion(sample):

    prediction = loaded_model.predict(sample)

    if prediction[0] == 1:
        print("Intrusion detected")
    else:
        print("Normal traffic")
```

In [50]:
```
predict_intrusion(X_test.iloc[[0]])
```

```
Intrusion detected
```

Multiclass Intrusion Detection

The system was extended from binary classification to multiclass classification.

Instead of detecting only intrusion vs normal traffic, the model now detects specific attack types.

Attack types include:

- normal
- neptune
- ipsweep
- portsweep
- smurf

This provides more realistic intrusion detection capability.

In [51]:
```
columns = [
'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',
'label','difficulty_level'
]

train_df_multi = pd.read_csv("../data/KDDTrain+.txt", names=columns)

test_df_multi = pd.read_csv("../data/KDDTest+.txt", names=columns)
```

In [52]:
```
train_df_multi.drop("difficulty_level", axis=1,inplace=True)
test_df_multi.drop("difficulty_level", axis=1,inplace=True)
```

In [53]:
```
combined_multi = pd.concat([train_df_multi, test_df_multi])

combined_multi_encoded = pd.get_dummies(
    combined_multi,
    columns=['protocol_type','service','flag']
)
```

In [54]:
```
train_multi = combined_multi_encoded.iloc[:len(train_df_multi)]

test_multi = combined_multi_encoded.iloc[len(train_df_multi):]

X_train_multi = train_multi.drop("label", axis=1)

Y_train_multi = train_multi["label"]

X_test_multi = test_multi.drop("label", axis=1)

Y_test_multi = test_multi["label"]
```

In [55]:
```
rf_multi = RandomForestClassifier(
    n_estimators=300,
    class_weight='balanced',
    random_state=42,
    n_jobs=-1
)

rf_multi.fit(X_train_multi, Y_train_multi)
```

Out[55]:

▾ RandomForestClassifier

▸ Parameters

In [56]:
```
Y_pred_multi = rf_multi.predict(X_test_multi)
```

```
multi_accuracy = accuracy_score(Y_test_multi, Y_pred_multi)

print("Multiclass Accuracy:", multi_accuracy)

print("\nClassification Report:\n")

print(classification_report(Y_test_multi, Y_pred_multi,zero_division=0))
```

```
Multiclass Accuracy: 0.7223207948899929

Classification Report:

                precision    recall  f1-score   support

        apache2       0.00      0.00      0.00       737
           back       0.92      0.95      0.93       359
buffer_overflow       0.00      0.00      0.00        20
      ftp_write       0.00      0.00      0.00         3
   guess_passwd       0.00      0.00      0.00      1231
     httptunnel       0.00      0.00      0.00       133
           imap       0.00      0.00      0.00         1
        ipsweep       0.63      0.98      0.77       141
           land       1.00      1.00      1.00         7
     loadmodule       0.00      0.00      0.00         2
       mailbomb       0.00      0.00      0.00       293
          mscan       0.00      0.00      0.00       996
       multihop       0.00      0.00      0.00        18
          named       0.00      0.00      0.00        17
        neptune       0.96      1.00      0.98      4657
           nmap       0.81      1.00      0.90        73
         normal       0.63      0.98      0.77      9711
           perl       1.00      0.50      0.67         2
            phf       0.00      0.00      0.00         2
            pod       0.70      0.95      0.80        41
      portsweep       0.78      0.99      0.87       157
   processtable       0.00      0.00      0.00       685
             ps       0.00      0.00      0.00        15
        rootkit       0.00      0.00      0.00        13
          saint       0.00      0.00      0.00       319
          satan       0.67      1.00      0.80       735
       sendmail       0.00      0.00      0.00        14
          smurf       1.00      1.00      1.00       665
  snmpgetattack       0.00      0.00      0.00       178
      snmpguess       0.00      0.00      0.00       331
      sqlattack       0.00      0.00      0.00         2
       teardrop       0.24      1.00      0.39        12
       udpstorm       0.00      0.00      0.00         2
    warezmaster       0.50      0.00      0.00       944
           worm       0.00      0.00      0.00         2
          xlock       0.00      0.00      0.00         9
         xsnoop       0.00      0.00      0.00         4
          xterm       0.00      0.00      0.00        13

       accuracy                           0.72     22544
      macro avg       0.26      0.30      0.26     22544
   weighted avg       0.57      0.72      0.62     22544
```

In [57]:
```
sample_results = pd.DataFrame({
```

```
    "Actual": Y_test_multi[:20],
    "Predicted": Y_pred_multi[:20]
})
sample_results
```

Out[57]:

	Actual	Predicted
0	neptune	neptune
1	neptune	neptune
2	normal	normal
3	saint	ipsweep
4	mscan	normal
5	normal	normal
6	normal	normal
7	guess_passwd	normal
8	normal	normal
9	guess_passwd	normal
10	mscan	normal
11	normal	normal
12	neptune	neptune
13	neptune	neptune
14	normal	normal
15	normal	normal
16	normal	normal
17	normal	normal
18	normal	normal
19	neptune	neptune

In [58]:
```
joblib.dump(rf_multi, "../model/multiclass_intrusion_model.pkl")
```

Out[58]:
```
['../model/multiclass_intrusion_model.pkl']
```

Prediction Function

A prediction function was implemented to simulate real-world deployment.

This function:

- Accepts network traffic features as input

- Uses the trained model
- Predicts the attack type

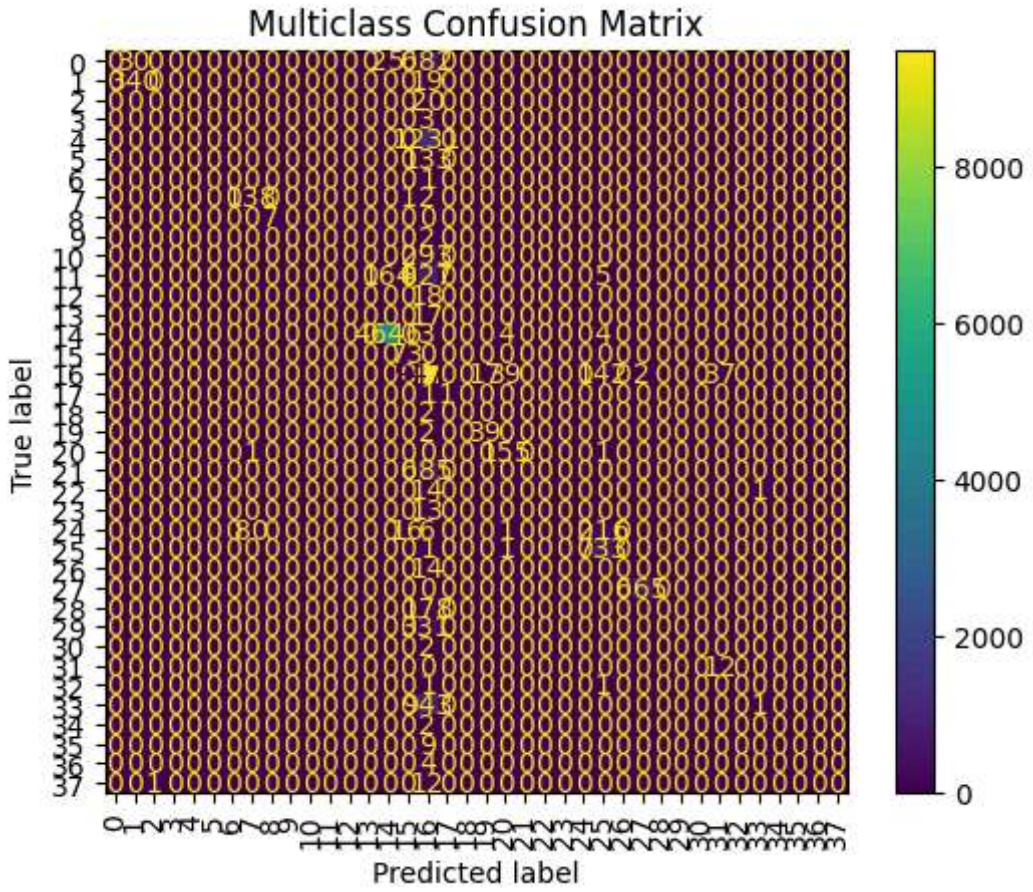
This demonstrates production readiness of the machine learning system.

In [59]:
```
# Load saved multiclass model
model = joblib.load("../model/multiclass_intrusion_model.pkl")

def predict_attack(sample_dict):

    sample_df = pd.DataFrame([sample_dict])

    prediction = model.predict(sample_df)[0]

    return prediction
```

In [60]:
```
# Test using real test sample

sample = X_test_multi.iloc[0].to_dict()

result = predict_attack(sample)

print("Predicted attack type:", result)

print("Actual attack type:", Y_test_multi.iloc[0])
```

```
Predicted attack type: neptune
Actual attack type: neptune
```

In [61]:
```
cm = confusion_matrix(Y_test_multi, Y_pred_multi)

disp = ConfusionMatrixDisplay(confusion_matrix=cm)

disp.plot(xticks_rotation=90)

plt.title("Multiclass Confusion Matrix")

plt.show()
```

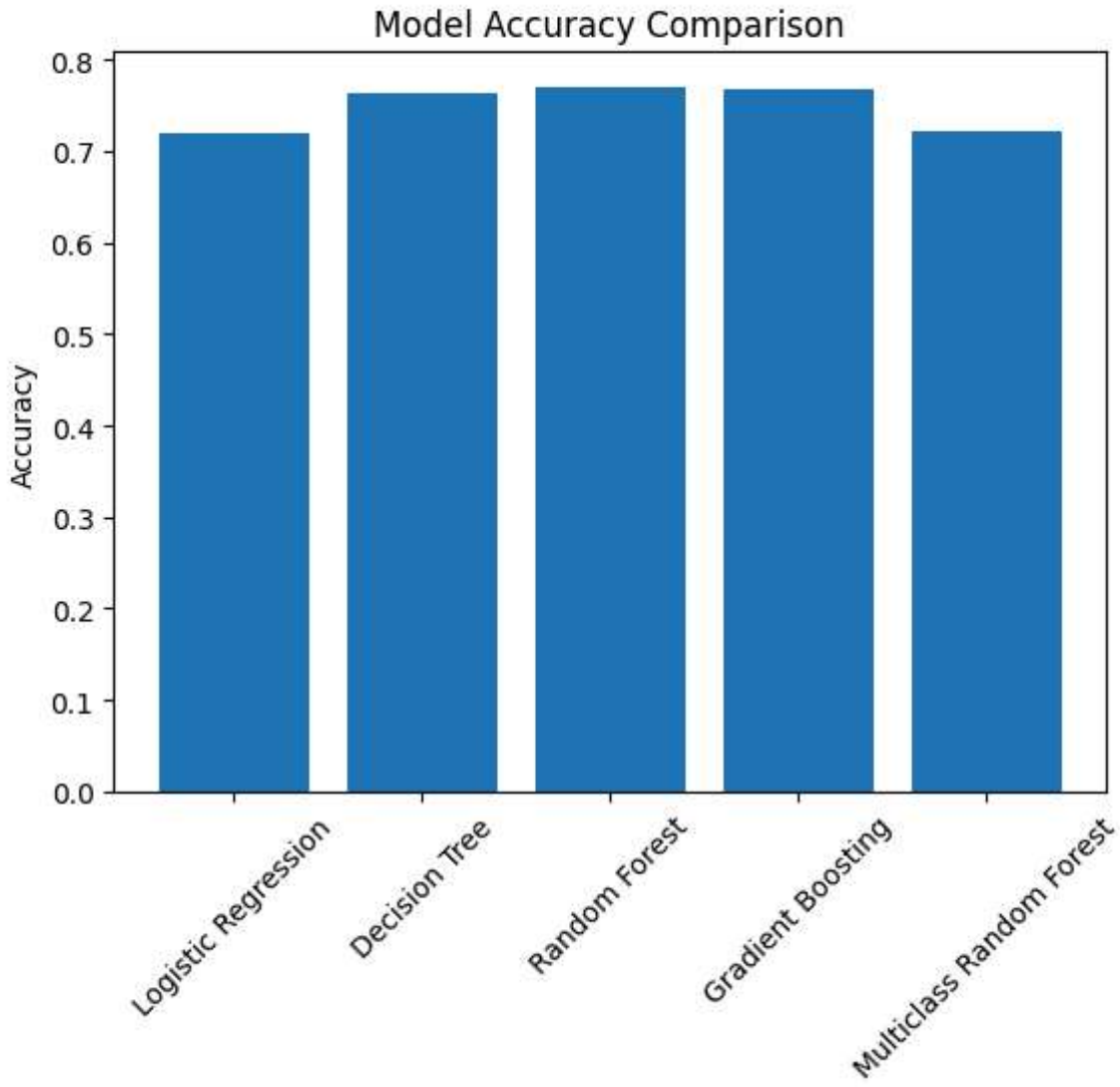


In [65]:
```
results = pd.DataFrame({

    "Model": [
        "Logistic Regression",
        "Decision Tree",
        "Random Forest",
        "Gradient Boosting",
        "Multiclass Random Forest"
    ],

    "Accuracy": [
        lr_acc,
        dt_acc,
        best_rf_acc,
        gb_acc,
        multi_accuracy
    ]

})

results.sort_values(by="Accuracy", ascending=False)
```

Out[65]:

	Model	Accuracy
2	Random Forest	0.769695
3	Gradient Boosting	0.767610
1	Decision Tree	0.763174
4	Multiclass Random Forest	0.722321
0	Logistic Regression	0.719526

In [66]:
```
plt.figure()

plt.bar(results["Model"], results["Accuracy"])

plt.xticks(rotation=45)

plt.title("Model Accuracy Comparison")

plt.ylabel("Accuracy")

plt.show()
```



In [67]:
```
joblib.dump(rf_multi, "../model/final_model.pkl")
```

Out[67]: ['../model/final_model.pkl']

Project Conclusion

A machine learning-based network intrusion detection system was successfully developed.

- Built binary and multiclass intrusion detection models
- Performed feature selection and hyperparameter tuning
- Evaluated multiple machine learning models
- Implemented cross-validation for reliability
- Created a prediction function for deployment

Random Forest demonstrated the best performance and was selected as the final model.

This project demonstrates strong understanding of machine learning, cybersecurity, and real-world model deployment.