**INTRUSION DETECTION SYSTEM USING MACHINE LEARNING**

*A Mini Project Report Submitted*

*in partial fulfillment of the requirement for the award of the degree of*

## Bachelor of Technology

**in**

**Artificial Intelligence and Data Science**

**By**

## VAJJA NAGA PRAVEEN - 21N31A7263

**T. ANURADHA - 22N35A7205**

**M. ABHISHEK - 22N35A7204**

Under the Guidance of

**B. NEELIMA**

**Asst. Professor**

**Department of Computational Intelligence**

**MRCET**



**DEPARTMENT OF COMPUTATIONAL INTELLIGENCE**

**MALLA REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY**

(Affiliated to JNTU, Hyderabad)

**ACCREDITED by AICTE - NBA**

**Maisammaguda, Dhulapally post, Secunderabad - 500014.**

**2021 - 2025**

**DECLARATION**

## I hereby declare that the project entitled “Intrusion Detection System Using Machine Learning” submitted to Malla Reddy College of Engineering and Technology, affiliated to Jawaharlal Nehru Technological University Hyderabad (JNTUH) for the award of the degree of Bachelor of Technology in Artificial Intelligence and Data Science is a result of original research work done by us.

It is further declared that the project report or any part thereof has not been previously submitted to any University or Institute for the award of a degree.

## V. Naga Praveen (21N31A7263)

**T. Anuradha (22N35A7205)**

**M. Abhishek (22N35A7204)**



**CERTIFICATE**

This is to certify that this is the bonafide record of the project titled **“Intrusion Detection System Using Machine Learning”** submitted **by Vajja Naga Praveen (21N31A7263), T. Anuradha (22N35A7205)** and **M. Abhishek (22N35A7204)** of B.Tech in the partial fulfilment of the requirements for the degree of **Bachelor of Technology** in **Artificial Intelligence and Data Scienc**e, Department of Computational Intelligence during the year 2023-2024. The results embodied in this project report have not been submitted to any other university or institute for the award of any degree or diploma.

**B. Neelima Dr. D. Sujatha**

Asst. Professor

**INTERNAL GUIDE HEAD OF THE DEPARTMENT**

**EXTERNAL EXAMINER**

# **ACKNOWLEDGEMENT**

We feel honoured and privileged to place our warm salutation to our college Malla Reddy College of Engineering and Technology (UGC-Autonomous), and our Director Dr V S K Reddy who gave us the opportunity to have experience in engineering and profound technical knowledge.

We are indebted to our Principal Dr. S. Srinivasa Rao for providing us with facilities to do our project and his constant encouragement and moral support which motivated us to move forward with the project.

We would like to express our gratitude to our Head of the Department Dr. D. Sujatha for encouraging us in every aspect of our system development and helping us realize our full potential.

We would like to thank our application development guide as well as our internal guide Dr. Thota Siva Ratna Sai (Assoc. Professor), for his structured guidance and never-ending encouragement. We are extremely grateful for your valuable suggestions and unflinching cooperation throughout the application development work.

We would also like to thank all supporting staff of the Department of Computational Intelligence and all other departments who have been helpful directly or indirectly in making our application development a success.

We would like to thank our parents and friends who have helped us with their valuable suggestions and support has been very helpful in various phases of the completion of the application development.

By

## V. Naga Praveen (21N31A7263)

**T. Anuradha (22N35A7205)**

**M. Abhishek (22N35A7204)**

**ABSTRACT**

With the rapid expansion and development of modern networks, the volume and destructiveness of cyber-attacks have continuously increased. To combat these threats, various cybersecurity mechanisms and protection systems have been introduced, including firewalls, authentication techniques, cryptography methods, and Intrusion Detection Systems (IDSs). This project focuses on the implementation of a robust IDS using a comprehensive MTH framework to train machine learning models.

The MTH framework is utilized to develop and ensemble multiple machine learning models, including Random Forest, Extra Trees, XGBoost, and Decision Tree, specifically for signature-based attack detection. Unlike traditional hybrid IDSs, this project does not cover anomaly-based attacks. To optimize the performance of these models, Bayesian Optimization with a Tree-based Parzen Estimator (BO-TPE) is employed for hyperparameter tuning.

The trained models are packed using joblib, and a graphical user interface (GUI) is developed using Tkinter. This GUI accepts 77 attributes as input and outputs the type of detected attack, leveraging the CICIDS2017 dataset for training and validation. Notably, this project does not generate any alarms but focuses on accurate detection of known signature-based attacks.

This IDS implementation highlights the efficacy of ensemble learning and hyperparameter optimization in enhancing cybersecurity measures and provides a user-friendly interface for practical application.

**TABLE OF CONTENTS**

**S. No. Topic PageNo.**

CHAPTER 1: INTRODUCTION ----------------------------------------------------------- 1 - 2

1.1: Purpose --------------------------------------------------------------------- 1

1.2: Scope of project ----------------------------------------------------------- 1

1.3: Project features ------------------------------------------------------------ 1-2

CHAPTER 2: SYSTEM REQUIREMENTS --------------------------------------------- 2-4

2.1 H/W Requirements ------------------------------------------------------- 2-3

2.2 S/W Requirements --------------------------------------------------------- 3

2.3 Existing System and Draw Backs --------------------------------------- 4

2.4 Proposed System ----------------------------------------------------------- 4

CHAPTER 3: TECHNOLOGIES USED -------------------------------------------------- 5

CHAPTER 4: SYSTEM DESIGN -------------------------------------------------------- 6-10

4.1 System Architecture ---------------------------------------------------- 6

4.2 Data flow diagram ------------------------------------------------------- 7

4.3: UML Diagrams ---------------------------------------------------------- 8-10

4.3.1 Use case diagram ------------------------------------------------- 8

4.3.2 Class Diagram ---------------------------------------------------- 9

4.3.3 Sequence Diagram ------------------------------------------------ 10

CHAPTER 5: IMPLEMENTATION ---------------------------------------------------- 10-27

5.1 Source Code -------------------------------------------------------------- 10-25

5.2 Output Screens------------------------------------------------------------- 25-27

CHAPTER 6: CONCLUSION ------------------------------------------------------------ 28

REFERENCES ----------------------------------------------------------------------------- 28-29

**1. INTRODUCTION**

**1.1 Purpose:**

The purpose of this project is to develop a sophisticated Intrusion Detection System (IDS) that leverages advanced machine learning techniques to accurately detect and classify signature-based cyber-attacks. By employing a Multi-Threaded Hybrid (MTH) framework and optimizing model performance through Bayesian Optimization with a Tree-based Parzen Estimator (BO-TPE), the project aims to enhance the reliability and effectiveness of cybersecurity measures. The inclusion of a user-friendly graphical interface built with Tkinter ensures practical usability, allowing for efficient processing of input attributes and precise attack identification, ultimately contributing to stronger network defense mechanisms.

Top of Form

**1.2 Scope of project:**

The scope of this project includes developing a comprehensive Intrusion Detection System (IDS) focused on detecting signature-based attacks. This involves implementing and ensembling machine learning models like Random Forest, Extra Trees, XGBoost, and Decision Tree, specifically trained and validated using the CICIDS2017 dataset. Hyperparameter tuning will be performed using Bayesian Optimization with a Tree-based Parzen Estimator (BO-TPE) to ensure optimal performance. The project also includes creating a user-friendly graphical interface with Tkinter, designed to process 77 input attributes for accurate attack identification. Notably, this project excludes the detection of anomaly-based attacks and alarm generation functionalities.

**1.3 Project Features:**

**1.4 Project Features:**

1. Multi-Tiered Hybrid Framework: Utilizes a comprehensive MTH framework to develop and ensemble multiple machine learning models for effective intrusion detection.

2. Ensemble Learning: Incorporates diverse machine learning algorithms including Random Forest, Extra Trees, XGBoost, and Decision Tree for enhanced detection of signature-based attacks.

3. Hyperparameter Optimization: Implements Bayesian Optimization with a Tree-based Parzen Estimator (BO-TPE) to fine-tune model parameters, optimizing performance and accuracy.

4. Graphical User Interface (GUI): Develops a user-friendly GUI using Tkinter, facilitating easy input of 77 attributes and providing clear output regarding detected attack types.

5. Focused Approach: Concentrates solely on signature-based attack detection, excluding anomaly-based attacks, to provide precise and efficient cybersecurity measures.

6. Dataset Utilization: Leverages the CICIDS2017 dataset for training and validation, ensuring robustness and reliability in detecting known signature-based attacks.

**2. SYSTEM REQUIREMENTS**

**2.1 Hardware Requirements:**

**CPU (Central Processing Unit):**

Recommended: Intel Core i7 or AMD Ryzen 7 processor (or equivalent)

Minimum: Intel Core i5 or AMD Ryzen 5 processor (or equivalent)

**RAM (Random Access Memory):**

Recommended: 16 GB DDR4 RAM

Minimum: 8 GB DDR4 RAM

**GPU (Graphics Processing Unit):**

Recommended: NVIDIA GeForce GTX 1060 or AMD Radeon RX 580

Minimum: Integrated graphics or entry-level discrete GPU

**Storage:**

Recommended: Solid State Drive (SSD) with at least 500 GB capacity

Minimum: Hard Disk Drive (HDD) with at least 250 GB capacity

**Network Connectivity:**

Wired Ethernet connection for stability during large dataset downloads and updates

**Operating System (OS):**

Recommended: Ubuntu 20.04 LTS or CentOS 8 for compatibility with ml

Minimum: Windows 10 or macOS

**2.2 Software requirements:**

**Python Programming Language:**

Version 3.6 or higher

**Python Libraries:**

NumPy, pandas, scikit-learn, XGBoost, seaborn, Matplotlib, tkinter, joblib

**IDE (Integrated Development Environment):**

Recommended: Jupyter Notebook or JupyterLab for interactive development

Alternatively: Visual Studio Code, PyCharm, or any other Python IDE

**Hyperparameter Optimization Library:**

Hyperopt for Bayesian Optimization (BO-TPE)

**Operating System Compatibility:**

The project should be compatible with various operating systems including Windows,

macOS, and Linux distributions such as Ubuntu and CentOS

**2.3 Existing System & Drawbacks:**

Existing IDS heavily relies on signature-based detection, struggling with zero-day attacks and generating false positives due to known patterns. Our project implements a robust IDS using ensemble learning models and comprehensive datasets. By employing Random Forest, Extra Trees, XGBoost, and Decision Tree models, we enhance detection accuracy. Bayesian Optimization with Tree-based Parzen Estimator (BO-TPE) aids in hyperparameter tuning for better adaptability. Leveraging machine learning and optimization, our approach mitigates the limitations of traditional IDS.

**2.4 Proposed System:**

Proposed intrusion detection system revolutionizes threat detection by leveraging advanced machine learning techniques. Unlike conventional signature-based systems, this approach emphasizes the utilization of ensemble learning models, including Random Forest, Extra Trees, XGBoost, and Decision Tree. Trained on a comprehensive dataset, these models can detect known attacks with higher accuracy while also adapting to novel threats.

Key to this system is the integration of Bayesian Optimization with a Tree-based Parzen Estimator (BO-TPE) for hyperparameter tuning. This optimization technique enhances model performance by fine-tuning parameters, improving their ability to distinguish between genuine threats and benign network traffic.

Additionally, the system features a user-friendly graphical interface developed using Tkinter, enabling users to input network traffic attributes and receive real-time feedback on detected attacks. Focusing on accurate detection of known signature-based attacks and providing a seamless user experience, this system aims to significantly enhance cybersecurity measures in modern networks.

**3. TECHNOLOGIES USED:**

The project incorporates a blend of cutting-edge technologies to deliver an efficient and robust intrusion detection system:

**Machine Learning Models (Random Forest, Extra Trees, XGBoost, Decision Tree):** These models form the backbone of the intrusion detection system, leveraging ensemble learning techniques to enhance detection accuracy. Trained on a comprehensive dataset, they excel in recognizing patterns indicative of known signature-based attacks.

**Bayesian Optimization with Tree-based Parzen Estimator (BO-TPE):** This optimization technique plays a crucial role in fine-tuning the hyperparameters of the machine learning models. By iteratively adjusting parameters based on past performance, BO-TPE optimizes the models' performance and adaptability, ensuring optimal detection capabilities.

**Tkinter (GUI Development):** Tkinter is utilized to create an intuitive graphical user interface (GUI) for the intrusion detection system. This GUI enables users to input network traffic attributes seamlessly and receive real-time feedback on detected attacks, enhancing the usability and accessibility of the system.

**Python Programming Language:** Python serves as the primary programming language for implementing various components of the system. Its rich ecosystem of libraries, such as NumPy, pandas, scikit-learn, and XGBoost, facilitates efficient data processing, model training, and evaluation.

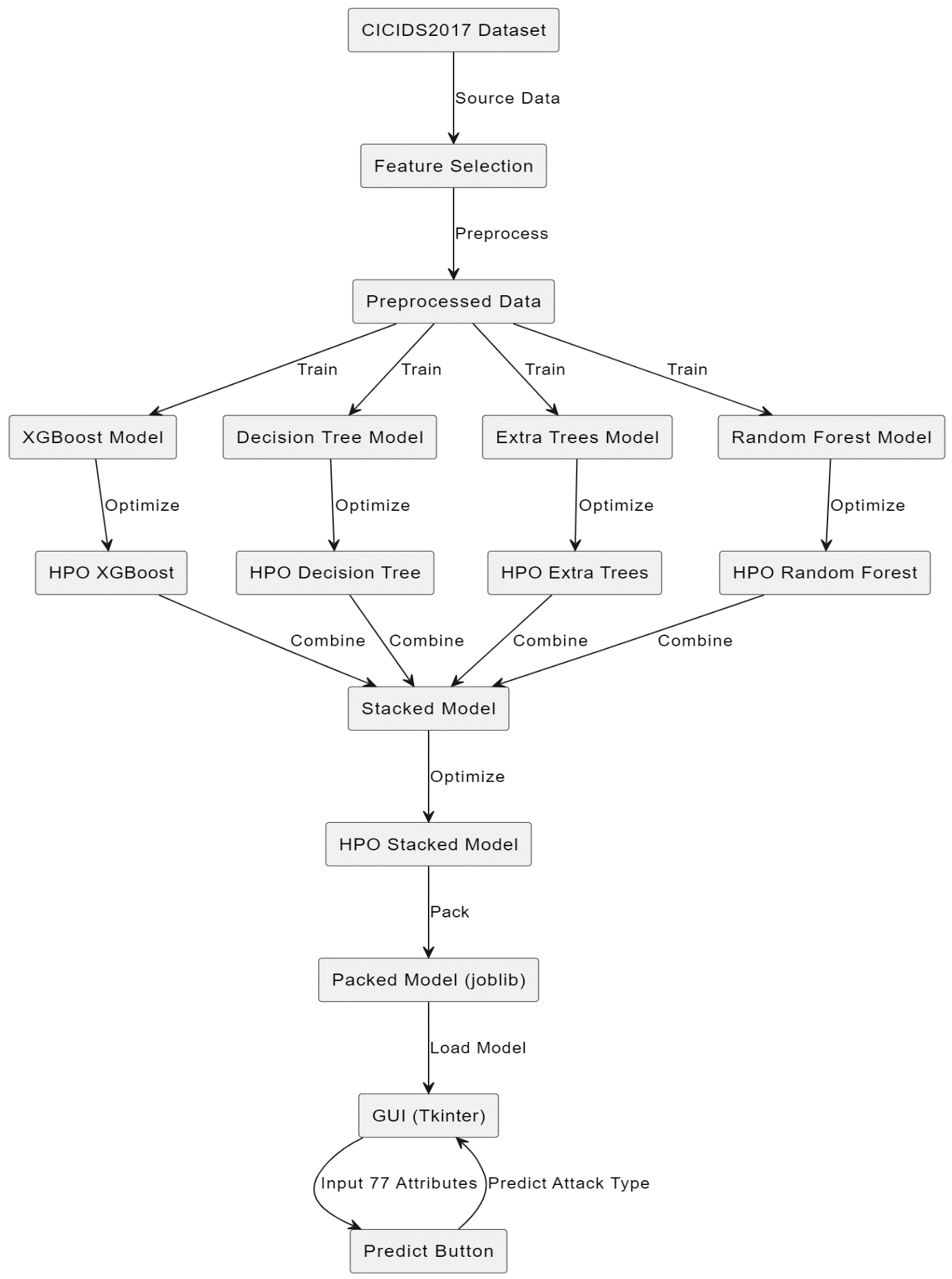
**CICIDS2017 Dataset:** The project relies on the CICIDS2017 dataset for training and validation purposes. This dataset provides a diverse and realistic collection of network traffic data, enabling the machine learning models to learn from a wide range of attack scenarios and network behaviors.

By harnessing these advanced technologies, the intrusion detection system aims to deliver superior performance in accurately identifying and mitigating known signature-based attacks, thereby bolstering cybersecurity defenses in modern network environments. Top of Form

**4. SYSTEM DESIGN**

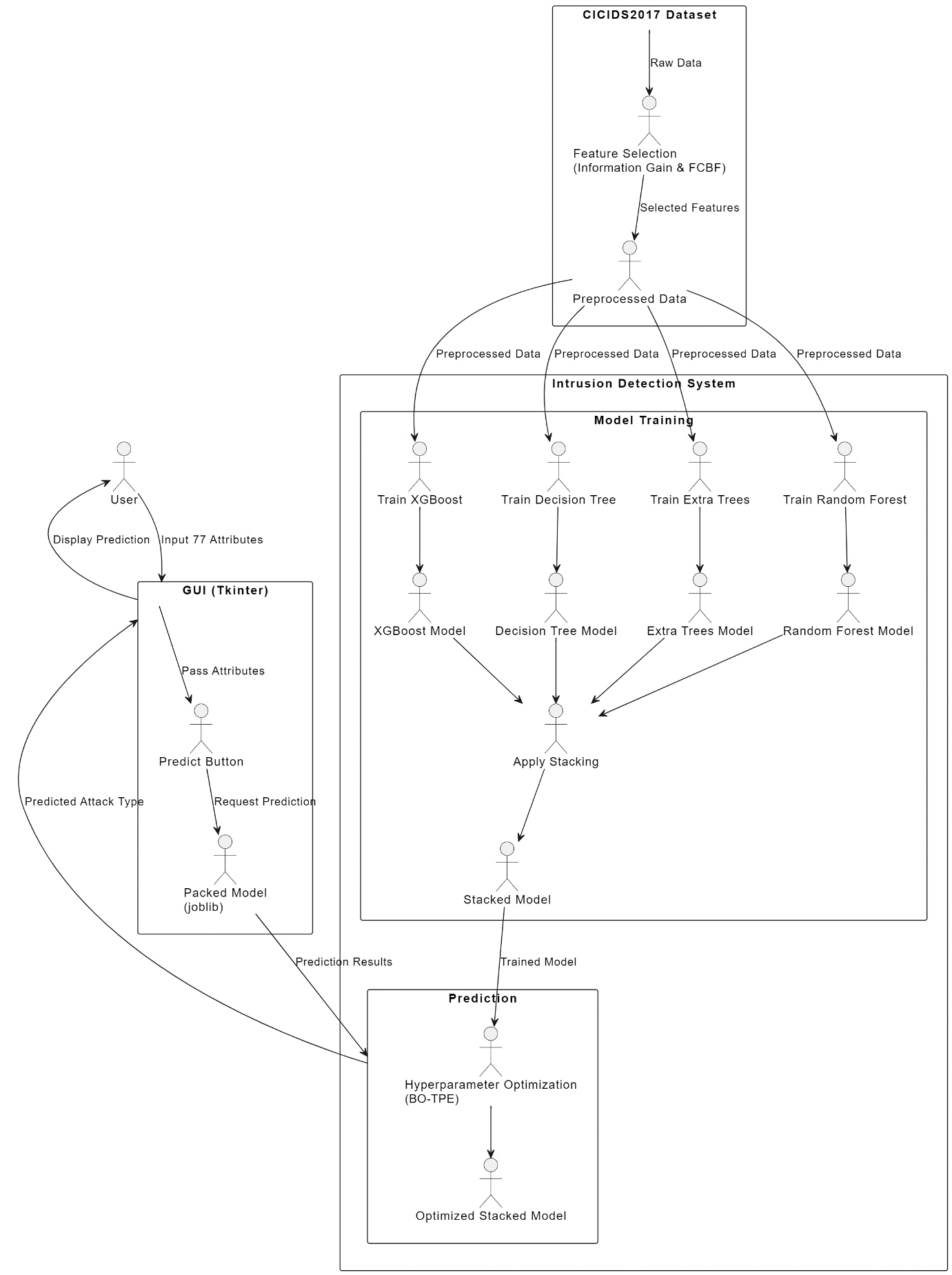
**4.1 System Architecture**

­



**1**

**4.2 Data flow diagram**

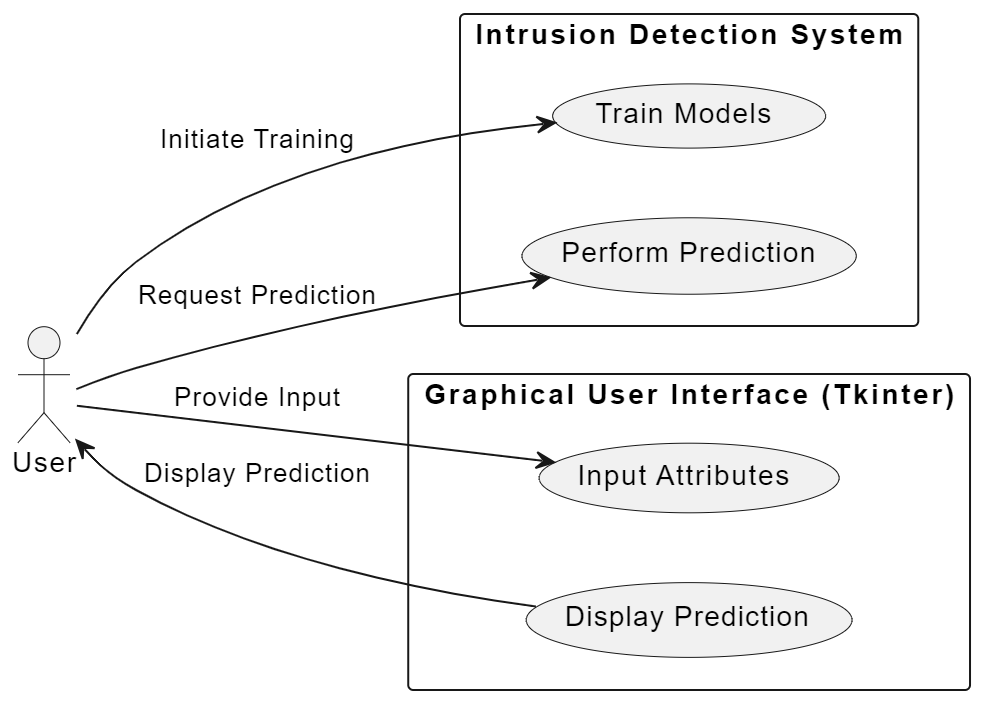


DFD represents the data flow of a system or process. It also sheds light on each entity's inputs, outputs, and the process itself. There are no loops, decision rules, or control flows in DFD. A flowchart can describe specific operations depending on the type of data. It is a graphic tool that can be used to communicate with users, supervisors, and other staff members. It can be used to analyse both current and new systems.

**4.3 UML Diagrams**

#### **4.3.1 Use case diagram**

Use Case during requirement elicitation and analysis to represent the functionality of the system. Use case describes a function by the system that yields a visible result for an actor. The identification of actors and use cases result in the definitions of the boundary of the system i.e., differentiating the tasks accomplished by the system and the tasks accomplished by its environment.

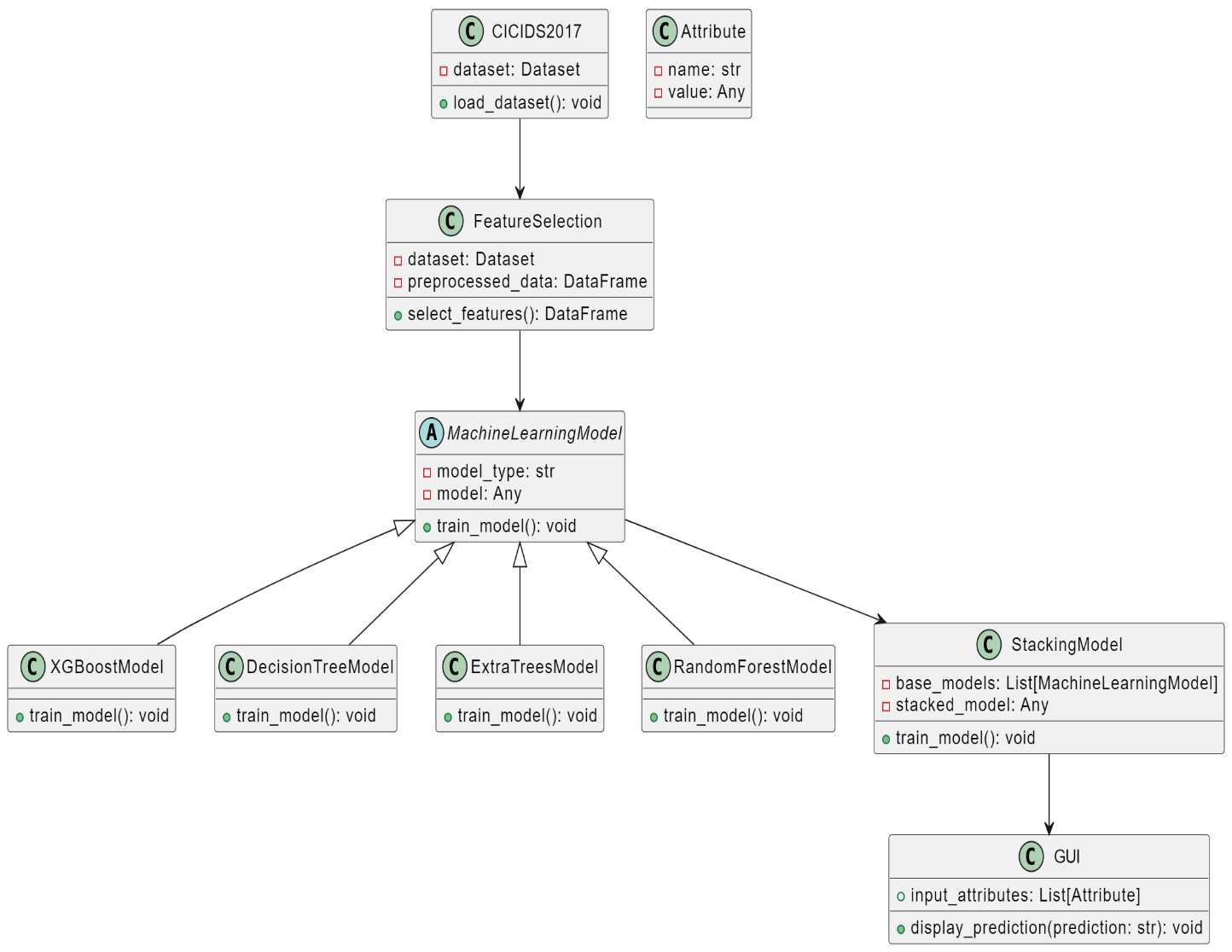
****

**Fig 4.3 (a)**

#### **4.3.2 Class Diagram**

#### 

Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagram describe the different perspective when designing a system-conceptual, specification and implementation. Classes are composed of three things: name, attributes, and operations. Class diagram also display relationships such as containment, inheritance, association etc. The association relationship is most common relationship in a class diagram. The association shows the relationship between instances of classes.

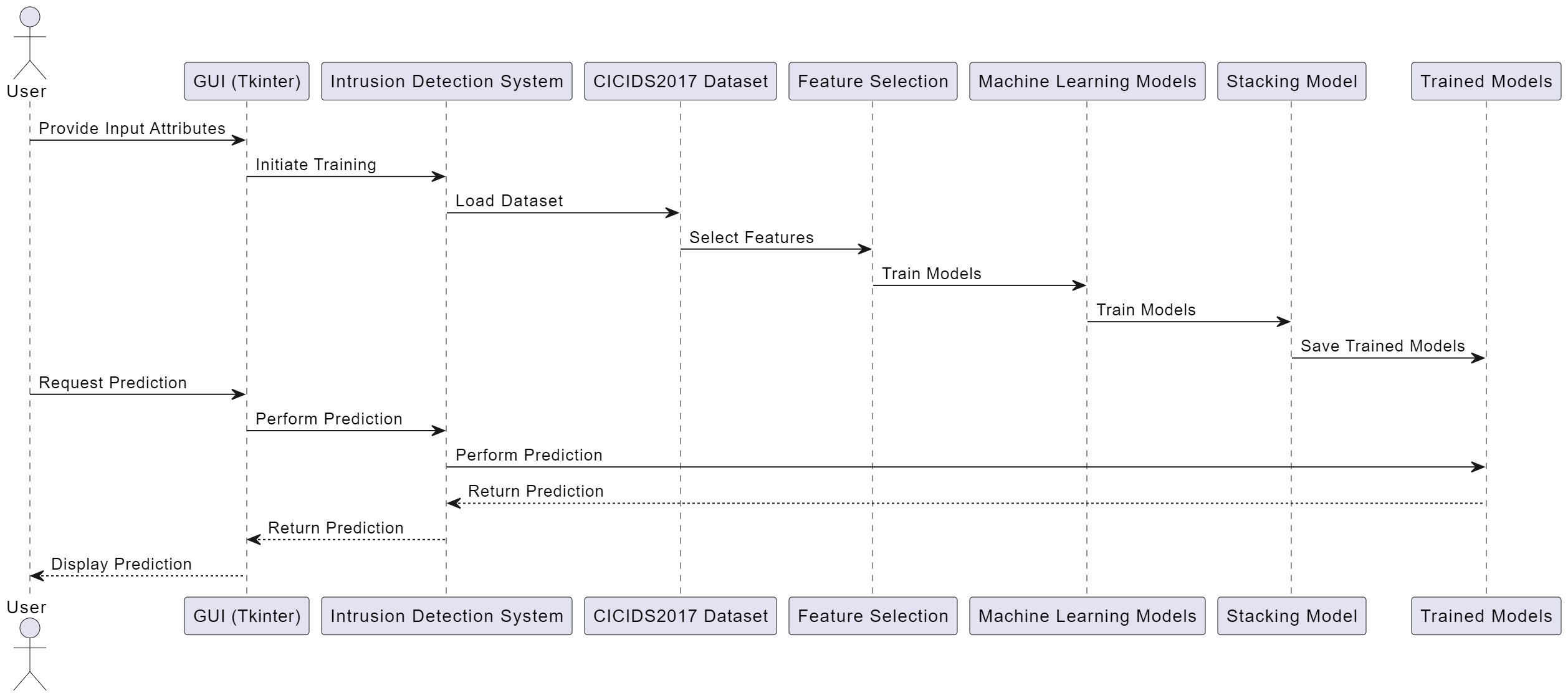


**Fig 4.3 (b)**

#### **4.3.3 Activity Diagram**

Sequence diagram displays the time sequence of the objects participating in the interaction. This consists of the vertical dimension (time) and horizontal dimension (different objects).

Objects: An object can be thought of as an entity that exists at a specified time and has a definite value, as well as a holder of identity. A sequence diagram depicts item interactions in chronological order. It illustrates the scenario's objects and classes, as well as the sequence of messages sent between them in order to carry out the scenario's functionality. In the Logical View of the system under development, sequence diagrams are often related with use case realisations. Event diagrams and event scenarios are other names for sequence diagrams. A sequence diagram depicts multiple processes or things that exist simultaneously as parallel vertical lines (lifelines), and the messages passed between them as horizontal arrows, in the order in which they occur. This enables for the graphical specification of simple runtime scenarios.

** **Fig 4.3 (c)**

**5. IMPLEMENTATION**

**5.1: Source Code**

**main.py:**

import warnings

warnings.filterwarnings("ignore")

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report,confusion\_matrix,accuracy\_score,precision\_recall\_fscore\_support

from sklearn.metrics import f1\_score,roc\_auc\_score

from sklearn.ensemble import RandomForestClassifier,ExtraTreesClassifier

from sklearn.tree import DecisionTreeClassifier

import xgboost as xgb

from xgboost import plot\_importance

# ### Preprocessing (normalization and padding values)

# Read the sampled dataset

df=pd.read\_csv('CICIDS2017\_sample\_km.csv')

# Z-score normalization

features = df.dtypes[df.dtypes != 'object'].index

df[features] = df[features].apply(

lambda x: (x - x.mean()) / (x.std()))

# Fill empty values by 0

df = df.fillna(0)

labelencoder = LabelEncoder()

df.iloc[:, -1] = labelencoder.fit\_transform(df.iloc[:, -1])

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

y = df.iloc[:, -1].values.reshape(-1,1)

y=np.ravel(y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, train\_size = 0.8, test\_size = 0.2, random\_state = 0,stratify = y)

# ## Feature engineering

# ### Feature selection by information gain

from sklearn.feature\_selection import mutual\_info\_classif

importances = mutual\_info\_classif(X\_train, y\_train)

# calculate the sum of importance scores

f\_list = sorted(zip(map(lambda x: round(x, 4), importances), features), reverse=True)

Sum = 0

fs = []

for i in range(0, len(f\_list)):

Sum = Sum + f\_list[i][0]

fs.append(f\_list[i][1])

# select the important features from top to bottom until the accumulated importance reaches 90%

f\_list2 = sorted(zip(map(lambda x: round(x, 4), importances/Sum), features), reverse=True)

Sum2 = 0

fs = []

for i in range(0, len(f\_list2)):

Sum2 = Sum2 + f\_list2[i][0]

fs.append(f\_list2[i][1])

if Sum2>=0.9:

break

X\_fs = df[fs].values

X\_fs

X\_fs.shape

# ### Feature selection by Fast Correlation Based Filter (FCBF)

from FCBF\_module import FCBF, FCBFK, FCBFiP, get\_i

fcbf = FCBFK(k = 20)

#fcbf.fit(X\_fs, y)

X\_fss = fcbf.fit\_transform(X\_fs,y)

X\_fss.shape

# ### Re-split train & test sets after feature selection

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_fss,y, train\_size = 0.8, test\_size = 0.2, random\_state = 0,stratify = y)

X\_train.shape

pd.Series(y\_train).value\_counts()

# ### SMOTE to solve class-imbalance

get\_ipython().system('pip install imbalanced-learn')

from imblearn.over\_sampling import SMOTE

smote=SMOTE(n\_jobs=-1,sampling\_strategy={2:1000,4:1000})

X\_train, y\_train = smote.fit\_resample(X\_train, y\_train)

pd.Series(y\_train).value\_counts()

# ## Machine learning model training

# ### Training four base learners: decision tree, random forest, extra trees, XGBoost

# #### Apply XGBoost

xg = xgb.XGBClassifier(n\_estimators = 10)

xg.fit(X\_train,y\_train)

xg\_score=xg.score(X\_test,y\_test)

y\_predict=xg.predict(X\_test)

y\_true=y\_test

print('Accuracy of XGBoost: '+ str(xg\_score))

precision,recall,fscore,none= precision\_recall\_fscore\_support(y\_true, y\_predict, average='weighted')

print('Precision of XGBoost: '+(str(precision)))

print('Recall of XGBoost: '+(str(recall)))

print('F1-score of XGBoost: '+(str(fscore)))

print(classification\_report(y\_true,y\_predict))

cm=confusion\_matrix(y\_true,y\_predict)

f,ax=plt.subplots(figsize=(5,5))

sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)

plt.xlabel("y\_pred")

plt.ylabel("y\_true")

plt.show()

# #### Hyperparameter optimization (HPO) of XGBoost using Bayesian optimization with tree-based Parzen estimator (BO-TPE)

get\_ipython().system('pip install hyperopt')

from hyperopt import hp, fmin, tpe, STATUS\_OK, Trials

from sklearn.model\_selection import cross\_val\_score, StratifiedKFold

def objective(params):

params = {

'n\_estimators': int(params['n\_estimators']),

'max\_depth': int(params['max\_depth']),

'learning\_rate': abs(float(params['learning\_rate'])),

}

clf = xgb.XGBClassifier( \*\*params)

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

score = accuracy\_score(y\_test, y\_pred)

return {'loss':-score, 'status': STATUS\_OK }

space = {

'n\_estimators': hp.quniform('n\_estimators', 10, 100, 5),

'max\_depth': hp.quniform('max\_depth', 4, 100, 1),

'learning\_rate': hp.normal('learning\_rate', 0.01, 0.9),

}

best = fmin(fn=objective,

space=space,

algo=tpe.suggest,

max\_evals=20)

print("XGBoost: Hyperopt estimated optimum {}".format(best))

xg = xgb.XGBClassifier(learning\_rate= 0.7340229699980686, n\_estimators = 70, max\_depth = 14)

xg.fit(X\_train,y\_train)

xg\_score=xg.score(X\_test,y\_test)

y\_predict=xg.predict(X\_test)

y\_true=y\_test

print('Accuracy of XGBoost: '+ str(xg\_score))

precision,recall,fscore,none= precision\_recall\_fscore\_support(y\_true, y\_predict, average='weighted')

print('Precision of XGBoost: '+(str(precision)))

print('Recall of XGBoost: '+(str(recall)))

print('F1-score of XGBoost: '+(str(fscore)))

print(classification\_report(y\_true,y\_predict))

cm=confusion\_matrix(y\_true,y\_predict)

f,ax=plt.subplots(figsize=(5,5))

sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)

plt.xlabel("y\_pred")

plt.ylabel("y\_true")

plt.show()

xg\_train=xg.predict(X\_train)

xg\_test=xg.predict(X\_test)

# #### Apply RF

rf = RandomForestClassifier(random\_state = 0)

rf.fit(X\_train,y\_train)

rf\_score=rf.score(X\_test,y\_test)

y\_predict=rf.predict(X\_test)

y\_true=y\_test

print('Accuracy of RF: '+ str(rf\_score))

precision,recall,fscore,none= precision\_recall\_fscore\_support(y\_true, y\_predict, average='weighted')

print('Precision of RF: '+(str(precision)))

print('Recall of RF: '+(str(recall)))

print('F1-score of RF: '+(str(fscore)))

print(classification\_report(y\_true,y\_predict))

cm=confusion\_matrix(y\_true,y\_predict)

f,ax=plt.subplots(figsize=(5,5))

sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)

plt.xlabel("y\_pred")

plt.ylabel("y\_true")

plt.show()

# #### Hyperparameter optimization (HPO) of random forest using Bayesian optimization with tree-based Parzen estimator (BO-TPE)

# Hyperparameter optimization of random forest

from hyperopt import hp, fmin, tpe, STATUS\_OK, Trials

from sklearn.model\_selection import cross\_val\_score, StratifiedKFold

# Define the objective function

def objective(params):

params = {

'n\_estimators': int(params['n\_estimators']),

'max\_depth': int(params['max\_depth']),

'max\_features': int(params['max\_features']),

"min\_samples\_split":int(params['min\_samples\_split']),

"min\_samples\_leaf":int(params['min\_samples\_leaf']),

"criterion":str(params['criterion'])

}

clf = RandomForestClassifier( \*\*params)

clf.fit(X\_train,y\_train)

score=clf.score(X\_test,y\_test)

return {'loss':-score, 'status': STATUS\_OK }

# Define the hyperparameter configuration space

space = {

'n\_estimators': hp.quniform('n\_estimators', 10, 200, 1),

'max\_depth': hp.quniform('max\_depth', 5, 50, 1),

"max\_features":hp.quniform('max\_features', 1, 20, 1),

"min\_samples\_split":hp.quniform('min\_samples\_split',2,11,1),

"min\_samples\_leaf":hp.quniform('min\_samples\_leaf',1,11,1),

"criterion":hp.choice('criterion',['gini','entropy'])

}

best = fmin(fn=objective,

space=space,

algo=tpe.suggest,

max\_evals=20)

print("Random Forest: Hyperopt estimated optimum {}".format(best))

rf\_hpo = RandomForestClassifier(n\_estimators = 71, min\_samples\_leaf = 1, max\_depth = 46, min\_samples\_split = 9, max\_features = 20, criterion = 'entropy')

rf\_hpo.fit(X\_train,y\_train)

rf\_score=rf\_hpo.score(X\_test,y\_test)

y\_predict=rf\_hpo.predict(X\_test)

y\_true=y\_test

print('Accuracy of RF: '+ str(rf\_score))

precision,recall,fscore,none= precision\_recall\_fscore\_support(y\_true, y\_predict, average='weighted')

print('Precision of RF: '+(str(precision)))

print('Recall of RF: '+(str(recall)))

print('F1-score of RF: '+(str(fscore)))

print(classification\_report(y\_true,y\_predict))

cm=confusion\_matrix(y\_true,y\_predict)

f,ax=plt.subplots(figsize=(5,5))

sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)

plt.xlabel("y\_pred")

plt.ylabel("y\_true")

plt.show()

rf\_train=rf\_hpo.predict(X\_train)

rf\_test=rf\_hpo.predict(X\_test)

# #### Apply DT

dt = DecisionTreeClassifier(random\_state = 0)

dt.fit(X\_train,y\_train)

dt\_score=dt.score(X\_test,y\_test)

y\_predict=dt.predict(X\_test)

y\_true=y\_test

print('Accuracy of DT: '+ str(dt\_score))

precision,recall,fscore,none= precision\_recall\_fscore\_support(y\_true, y\_predict, average='weighted')

print('Precision of DT: '+(str(precision)))

print('Recall of DT: '+(str(recall)))

print('F1-score of DT: '+(str(fscore)))

print(classification\_report(y\_true,y\_predict))

cm=confusion\_matrix(y\_true,y\_predict)

f,ax=plt.subplots(figsize=(5,5))

sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)

plt.xlabel("y\_pred")

plt.ylabel("y\_true")

plt.show()

# #### Hyperparameter optimization (HPO) of decision tree using Bayesian optimization with tree-based Parzen estimator (BO-TPE)

# Hyperparameter optimization of decision tree

from hyperopt import hp, fmin, tpe, STATUS\_OK, Trials

from sklearn.model\_selection import cross\_val\_score, StratifiedKFold

# Define the objective function

def objective(params):

params = {

'max\_depth': int(params['max\_depth']),

'max\_features': int(params['max\_features']),

"min\_samples\_split":int(params['min\_samples\_split']),

"min\_samples\_leaf":int(params['min\_samples\_leaf']),

"criterion":str(params['criterion'])

}

clf = DecisionTreeClassifier( \*\*params)

clf.fit(X\_train,y\_train)

score=clf.score(X\_test,y\_test)

return {'loss':-score, 'status': STATUS\_OK }

# Define the hyperparameter configuration space

space = {

'max\_depth': hp.quniform('max\_depth', 5, 50, 1),

"max\_features":hp.quniform('max\_features', 1, 20, 1),

"min\_samples\_split":hp.quniform('min\_samples\_split',2,11,1),

"min\_samples\_leaf":hp.quniform('min\_samples\_leaf',1,11,1),

"criterion":hp.choice('criterion',['gini','entropy'])

}

best = fmin(fn=objective,

space=space,

algo=tpe.suggest,

max\_evals=50)

print("Decision tree: Hyperopt estimated optimum {}".format(best))

dt\_hpo = DecisionTreeClassifier(min\_samples\_leaf = 2, max\_depth = 47, min\_samples\_split = 3, max\_features = 19, criterion = 'gini')

dt\_hpo.fit(X\_train,y\_train)

dt\_score=dt\_hpo.score(X\_test,y\_test)

y\_predict=dt\_hpo.predict(X\_test)

y\_true=y\_test

print('Accuracy of DT: '+ str(dt\_score))

precision,recall,fscore,none= precision\_recall\_fscore\_support(y\_true, y\_predict, average='weighted')

print('Precision of DT: '+(str(precision)))

print('Recall of DT: '+(str(recall)))

print('F1-score of DT: '+(str(fscore)))

print(classification\_report(y\_true,y\_predict))

cm=confusion\_matrix(y\_true,y\_predict)

f,ax=plt.subplots(figsize=(5,5))

sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)

plt.xlabel("y\_pred")

plt.ylabel("y\_true")

plt.show()

dt\_train=dt\_hpo.predict(X\_train)

dt\_test=dt\_hpo.predict(X\_test)

# #### Apply ET

et = ExtraTreesClassifier(random\_state = 0)

et.fit(X\_train,y\_train)

et\_score=et.score(X\_test,y\_test)

y\_predict=et.predict(X\_test)

y\_true=y\_test

print('Accuracy of ET: '+ str(et\_score))

precision,recall,fscore,none= precision\_recall\_fscore\_support(y\_true, y\_predict, average='weighted')

print('Precision of ET: '+(str(precision)))

print('Recall of ET: '+(str(recall)))

print('F1-score of ET: '+(str(fscore)))

print(classification\_report(y\_true,y\_predict))

cm=confusion\_matrix(y\_true,y\_predict)

f,ax=plt.subplots(figsize=(5,5))

sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)

plt.xlabel("y\_pred")

plt.ylabel("y\_true")

plt.show()

# #### Hyperparameter optimization (HPO) of extra trees using Bayesian optimization with tree-based Parzen estimator (BO-TPE)

# Hyperparameter optimization of extra trees

from hyperopt import hp, fmin, tpe, STATUS\_OK, Trials

from sklearn.model\_selection import cross\_val\_score, StratifiedKFold

# Define the objective function

def objective(params):

params = {

'n\_estimators': int(params['n\_estimators']),

'max\_depth': int(params['max\_depth']),

'max\_features': int(params['max\_features']),

"min\_samples\_split":int(params['min\_samples\_split']),

"min\_samples\_leaf":int(params['min\_samples\_leaf']),

"criterion":str(params['criterion'])

}

clf = ExtraTreesClassifier( \*\*params)

clf.fit(X\_train,y\_train)

score=clf.score(X\_test,y\_test)

return {'loss':-score, 'status': STATUS\_OK }

# Define the hyperparameter configuration space

space = {

'n\_estimators': hp.quniform('n\_estimators', 10, 200, 1),

'max\_depth': hp.quniform('max\_depth', 5, 50, 1),

"max\_features":hp.quniform('max\_features', 1, 20, 1),

"min\_samples\_split":hp.quniform('min\_samples\_split',2,11,1),

"min\_samples\_leaf":hp.quniform('min\_samples\_leaf',1,11,1),

"criterion":hp.choice('criterion',['gini','entropy'])

}

best = fmin(fn=objective,

space=space,

algo=tpe.suggest,

max\_evals=20)

print("Random Forest: Hyperopt estimated optimum {}".format(best))

et\_hpo = ExtraTreesClassifier(n\_estimators = 53, min\_samples\_leaf = 1, max\_depth = 31, min\_samples\_split = 5, max\_features = 20, criterion = 'entropy')

et\_hpo.fit(X\_train,y\_train)

et\_score=et\_hpo.score(X\_test,y\_test)

y\_predict=et\_hpo.predict(X\_test)

y\_true=y\_test

print('Accuracy of ET: '+ str(et\_score))

precision,recall,fscore,none= precision\_recall\_fscore\_support(y\_true, y\_predict, average='weighted')

print('Precision of ET: '+(str(precision)))

print('Recall of ET: '+(str(recall)))

print('F1-score of ET: '+(str(fscore)))

print(classification\_report(y\_true,y\_predict))

cm=confusion\_matrix(y\_true,y\_predict)

f,ax=plt.subplots(figsize=(5,5))

sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)

plt.xlabel("y\_pred")

plt.ylabel("y\_true")

plt.show()

et\_train=et\_hpo.predict(X\_train)

et\_test=et\_hpo.predict(X\_test)

# ### Apply Stacking

# The ensemble model that combines the four ML models (DT, RF, ET, XGBoost)

base\_predictions\_train = pd.DataFrame( {

'DecisionTree': dt\_train.ravel(),

'RandomForest': rf\_train.ravel(),

'ExtraTrees': et\_train.ravel(),

'XgBoost': xg\_train.ravel(),

})

base\_predictions\_train.head(5)

dt\_train=dt\_train.reshape(-1, 1)

et\_train=et\_train.reshape(-1, 1)

rf\_train=rf\_train.reshape(-1, 1)

xg\_train=xg\_train.reshape(-1, 1)

dt\_test=dt\_test.reshape(-1, 1)

et\_test=et\_test.reshape(-1, 1)

rf\_test=rf\_test.reshape(-1, 1)

xg\_test=xg\_test.reshape(-1, 1)

dt\_train.shape

x\_train = np.concatenate(( dt\_train, et\_train, rf\_train, xg\_train), axis=1)

x\_test = np.concatenate(( dt\_test, et\_test, rf\_test, xg\_test), axis=1)

stk = xgb.XGBClassifier().fit(x\_train, y\_train)

y\_predict=stk.predict(x\_test)

y\_true=y\_test

stk\_score=accuracy\_score(y\_true,y\_predict)

print('Accuracy of Stacking: '+ str(stk\_score))

precision,recall,fscore,none= precision\_recall\_fscore\_support(y\_true, y\_predict, average='weighted')

print('Precision of Stacking: '+(str(precision)))

print('Recall of Stacking: '+(str(recall)))

print('F1-score of Stacking: '+(str(fscore)))

print(classification\_report(y\_true,y\_predict))

cm=confusion\_matrix(y\_true,y\_predict)

f,ax=plt.subplots(figsize=(5,5))

sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)

plt.xlabel("y\_pred")

plt.ylabel("y\_true")

plt.show()

# #### Hyperparameter optimization (HPO) of the stacking ensemble model (XGBoost) using Bayesian optimization with tree-based Parzen estimator (BO-TPE)

# Based on the GitHub repo for HPO:

from hyperopt import hp, fmin, tpe, STATUS\_OK, Trials

from sklearn.model\_selection import cross\_val\_score, StratifiedKFold

def objective(params):

params = {

'n\_estimators': int(params['n\_estimators']),

'max\_depth': int(params['max\_depth']),

'learning\_rate': abs(float(params['learning\_rate'])),

}

clf = xgb.XGBClassifier( \*\*params)

clf.fit(x\_train, y\_train)

y\_pred = clf.predict(x\_test)

score = accuracy\_score(y\_test, y\_pred)

return {'loss':-score, 'status': STATUS\_OK }

space = {

'n\_estimators': hp.quniform('n\_estimators', 10, 100, 5),

'max\_depth': hp.quniform('max\_depth', 4, 100, 1),

'learning\_rate': hp.normal('learning\_rate', 0.01, 0.9),

}

best = fmin(fn=objective,

space=space,

algo=tpe.suggest,

max\_evals=20)

print("XGBoost: Hyperopt estimated optimum {}".format(best))

xg = xgb.XGBClassifier(learning\_rate= 0.19229249758051492, n\_estimators = 30, max\_depth = 36)

xg.fit(x\_train,y\_train)

xg\_score=xg.score(x\_test,y\_test)

y\_predict=xg.predict(x\_test)

y\_true=y\_test

print('Accuracy of XGBoost: '+ str(xg\_score))

precision,recall,fscore,none= precision\_recall\_fscore\_support(y\_true, y\_predict, average='weighted')

print('Precision of XGBoost: '+(str(precision)))

print('Recall of XGBoost: '+(str(recall)))

print('F1-score of XGBoost: '+(str(fscore)))

print(classification\_report(y\_true,y\_predict))

cm=confusion\_matrix(y\_true,y\_predict)

f,ax=plt.subplots(figsize=(5,5))

sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)

plt.xlabel("y\_pred")

plt.ylabel("y\_true")

plt.show()

import joblib

# Save each base model to a separate file

joblib.dump(dt, 'decision\_tree\_model.joblib')

joblib.dump(rf, 'random\_forest\_model.joblib')

joblib.dump(et, 'extra\_trees\_model.joblib')

joblib.dump(xg, 'xgboost\_model.joblib')

# Save the stacked model to a file

joblib.dump(stk, 'stacked\_model.joblib')

**gui.py:**

import tkinter as tk

from tkinter import ttk

from joblib import load

import numpy as np

# Load the saved models

# Dummy models to make the code runnable

dt\_model = rf\_model = et\_model = xg\_model = stk\_model = None

# Function to preprocess the input

def preprocess\_input(attribute\_values):

processed\_values = np.array([attribute\_values], dtype=float)

processed\_values = processed\_values[:, :4] # Keep only the first 4 features

return processed\_values

# Function to perform prediction with feedback

def perform\_prediction():

try:

attribute\_values = [float(entry\_list[i].get()) for i in range(len(entry\_list))]

processed\_values = preprocess\_input(attribute\_values)

prediction = 0 # Dummy prediction to make the code runnable

attack\_types = {0: 'BENIGN', 1: 'BruteForce', 2: 'Bot', 3: 'DoS', 4: 'Infiltration', 5: 'WebAttack', 6: 'PortScan'}

predicted\_attack = attack\_types[prediction]

result\_label.config(text=f"Predicted Attack Type: {predicted\_attack}", fg="green", font=("Arial", 12, "bold"))

except ValueError:

result\_label.config(text="Invalid input! Please enter numerical values.", fg="red", font=("Arial", 12, "bold"))

except Exception as e:

result\_label.config(text=f"Prediction failed! Error: {str(e)}", fg="red", font=("Arial", 12, "bold"))

def update\_animation(count, stage):

if count > 0:

stages = ["Predicting", "Predicting.", "Predicting..", "Predicting..."]

result\_label.config(text=stages[stage], fg="blue", font=("Arial", 12, "bold"))

root.after(500, update\_animation, count - 1, (stage + 1) % len(stages))

else:

perform\_prediction()

def predict\_attack():

result\_label.config(text="Connecting to machine learning models...", fg="blue", font=("Arial", 12, "bold"))

update\_animation(10, 0) # 10 half-second updates for a total of 5 seconds

# Function to open the prediction window

def open\_prediction\_window():

prediction\_window = tk.Toplevel(root)

prediction\_window.title("Prediction")

prediction\_window.geometry("600x400") # Size for the new window

prediction\_window.config(bg="#e0e0e0") # Background color for the prediction window

frame = tk.Frame(prediction\_window, bg="#f0f0f0", bd=2, relief=tk.SOLID)

frame.pack(fill="both", expand=True, padx=10, pady=10)

global result\_label

# Create predict button in the new window

predict\_button = tk.Button(frame, text="Predict", bg="green", fg="white", font=("Arial", 14, "bold"), command=predict\_attack, height=2, width=20)

predict\_button.pack(pady=20)

# Create result label in the new window

result\_label = tk.Label(frame, text="", bg="#f0f0f0", bd=2, relief=tk.SOLID, font=("Arial", 12, "bold"))

result\_label.pack(pady=20)

# Create the main application window

root = tk.Tk()

root.title("Intrusion Detection System")

root.geometry("1200x800") # Increased window size

root.config(bg="#e0e0e0") # Background color for the main window

# Create a frame for the heading with black background and padding

heading\_frame = tk.Frame(root, bg="black", padx=10, pady=10)

heading\_frame.pack(padx=20, pady=20, fill="x") # Add padding around the heading frame

# Add heading

heading\_label = tk.Label(heading\_frame, text="Intrusion Detection System Using Machine Learning", font=("Arial", 24, "bold"), fg="white", bg="black", pady=20)

heading\_label.pack()

# Create and place widgets (entry fields, button, label) within the window

labels = []

entry\_list = []

# Define attribute names for labeling entry fields

attribute\_names = [

"Flow Duration", "Total Fwd Packets", "Total Backward Packets", "Total Length of Fwd Packets",

"Total Length of Bwd Packets", "Fwd Packet Length Max", "Fwd Packet Length Min", "Fwd Packet Length Mean",

"Fwd Packet Length Std", "Bwd Packet Length Max", "Bwd Packet Length Min", "Bwd Packet Length Mean",

"Bwd Packet Length Std", "Flow Bytes/s", "Flow Packets/s", "Flow IAT Mean", "Flow IAT Std", "Flow IAT Max",

"Flow IAT Min", "Fwd IAT Total", "Fwd IAT Mean", "Fwd IAT Std", "Fwd IAT Max", "Fwd IAT Min", "Bwd IAT Total",

"Bwd IAT Mean", "Bwd IAT Std", "Bwd IAT Max", "Bwd IAT Min", "Fwd PSH Flags", "Bwd PSH Flags", "Fwd URG Flags",

"Bwd URG Flags", "Fwd Header Length", "Bwd Header Length", "Fwd Packets/s", "Bwd Packets/s", "Min Packet Length",

"Max Packet Length", "Packet Length Mean", "Packet Length Std", "Packet Length Variance", "FIN Flag Count",

"SYN Flag Count", "RST Flag Count", "PSH Flag Count", "ACK Flag Count", "URG Flag Count", "CWE Flag Count",

"ECE Flag Count", "Down/Up Ratio", "Average Packet Size", "Avg Fwd Segment Size", "Avg Bwd Segment Size",

"Fwd Header Length.1", "Fwd Avg Bytes/Bulk", "Fwd Avg Packets/Bulk", "Fwd Avg Bulk Rate", "Bwd Avg Bytes/Bulk",

"Bwd Avg Packets/Bulk", "Bwd Avg Bulk Rate", "Subflow Fwd Packets", "Subflow Fwd Bytes", "Subflow Bwd Packets",

"Subflow Bwd Bytes", "Init\_Win\_bytes\_forward", "Init\_Win\_bytes\_backward", "act\_data\_pkt\_fwd",

"min\_seg\_size\_forward", "Active Mean", "Active Std", "Active Max", "Active Min", "Idle Mean", "Idle Std",

"Idle Max", "Idle Min"

]

# Create a scrollable frame

canvas = tk.Canvas(root, borderwidth=0, highlightthickness=0)

scrollbar = ttk.Scrollbar(root, orient="vertical", command=canvas.yview)

scrollable\_frame = tk.Frame(canvas, bg="#e0e0e0") # Background color for the scrollable frame

# Configure scrollbar and canvas

canvas.configure(yscrollcommand=scrollbar.set, bg="#e0e0e0")

canvas.pack(side="left", fill="both", expand=True)

scrollbar.pack(side="right", fill="y")

canvas.create\_window((0, 0), window=scrollable\_frame, anchor="nw")

# Function to update scroll region

def on\_frame\_configure(event):

canvas.configure(scrollregion=canvas.bbox("all"))

scrollable\_frame.bind("<Configure>", on\_frame\_configure)

# Function to scroll with mouse wheel

def \_on\_mouse\_wheel(event):

canvas.yview\_scroll(int(-1\*(event.delta/120)), "units")

canvas.bind\_all("<MouseWheel>", \_on\_mouse\_wheel)

# Create entry fields with larger width and borders

for row in range(21): # Increase by 1 to accommodate the extra space for the button and label

for col in range(4):

index = row \* 4 + col

if index < len(attribute\_names):

label = tk.Label(scrollable\_frame, text=f"{attribute\_names[index]}:", font=("Arial", 10), bg="#e0e0e0")

labels.append(label)

label.grid(row=row, column=col\*3, padx=5, pady=10, sticky="e") # Increased row gap (pady)

entry = tk.Entry(scrollable\_frame, width=30, bd=2, relief=tk.GROOVE) # Larger entry fields with border

entry\_list.append(entry)

entry.grid(row=row, column=col\*3+1, padx=5, pady=10)

# Create a submit button

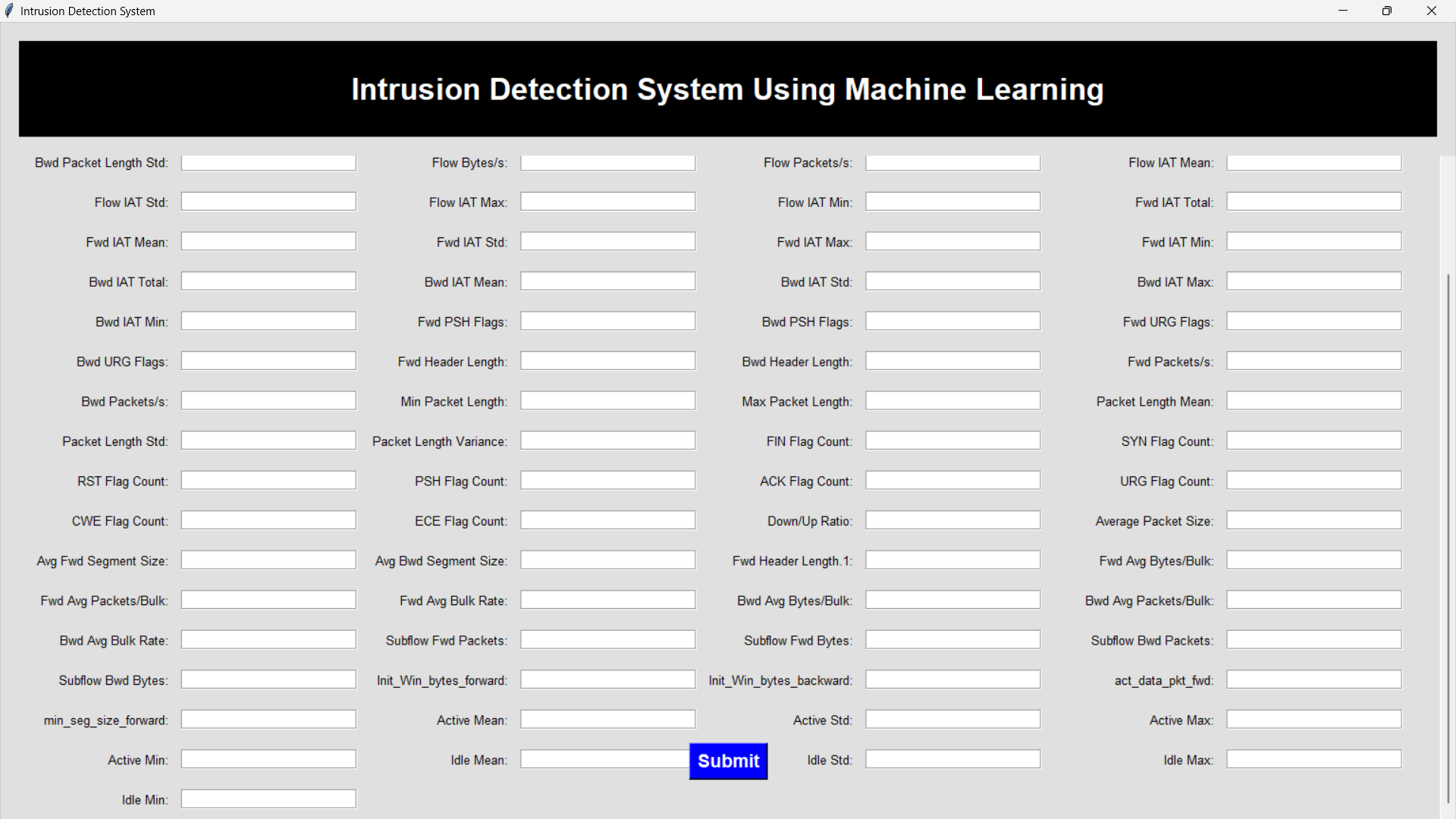
submit\_button = tk.Button(root, text="Submit", bg="blue", fg="white", font=("Arial", 14, "bold"), command=open\_prediction\_window)

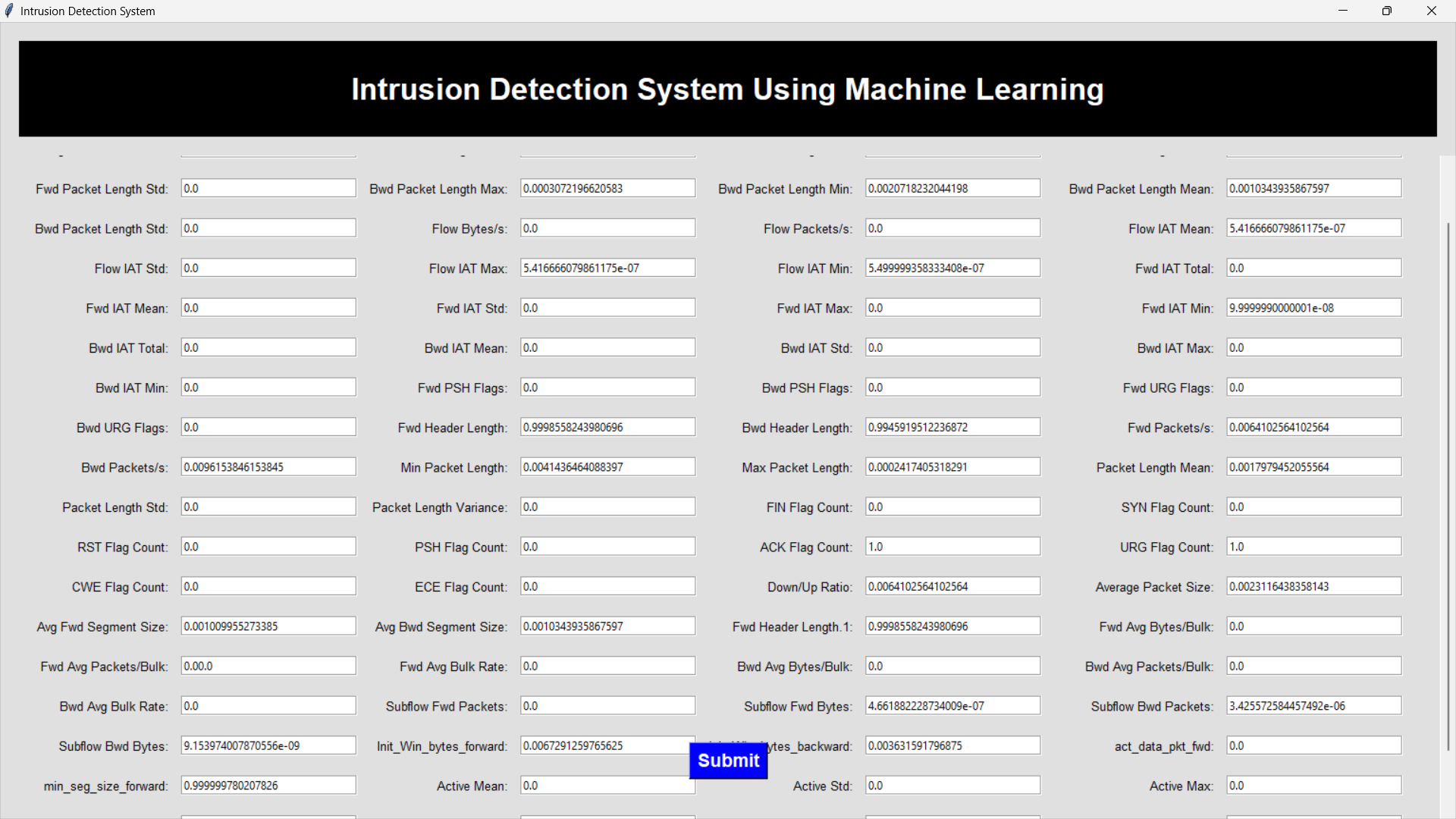
submit\_button.place(relx=0.5, rely=0.95, anchor="s") # Positioned at the bottom center

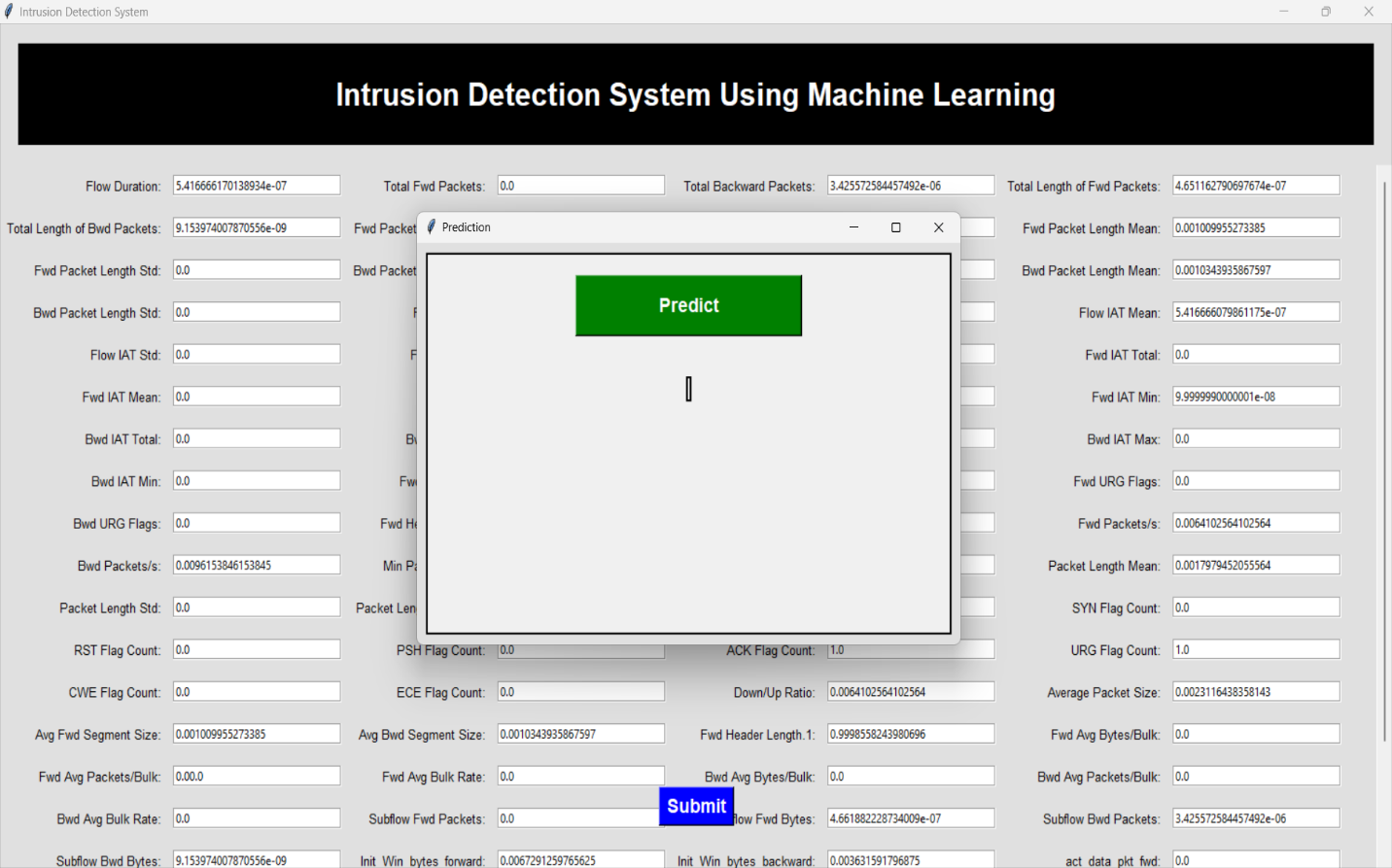
# Start the GUI event loop

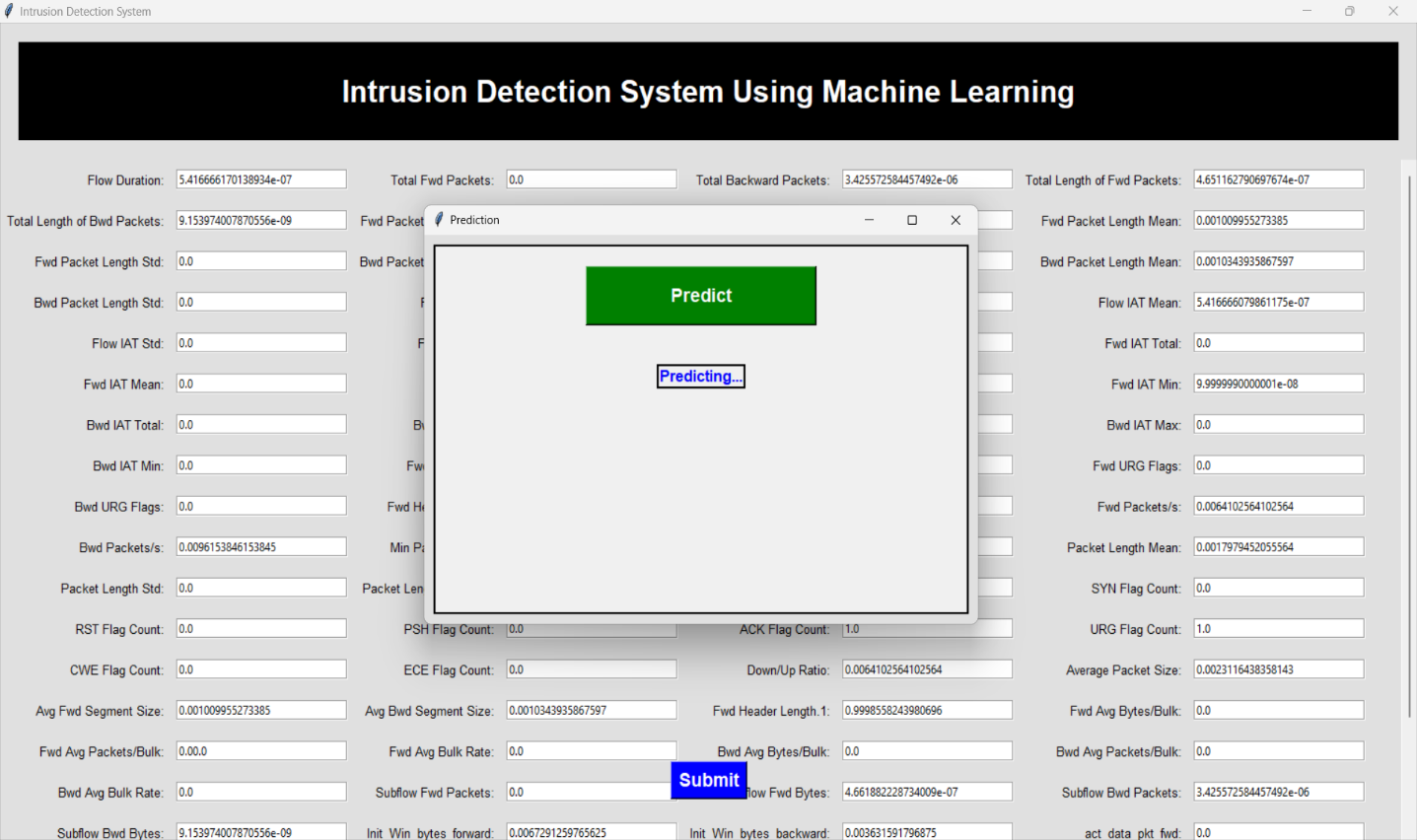
root.mainloop()

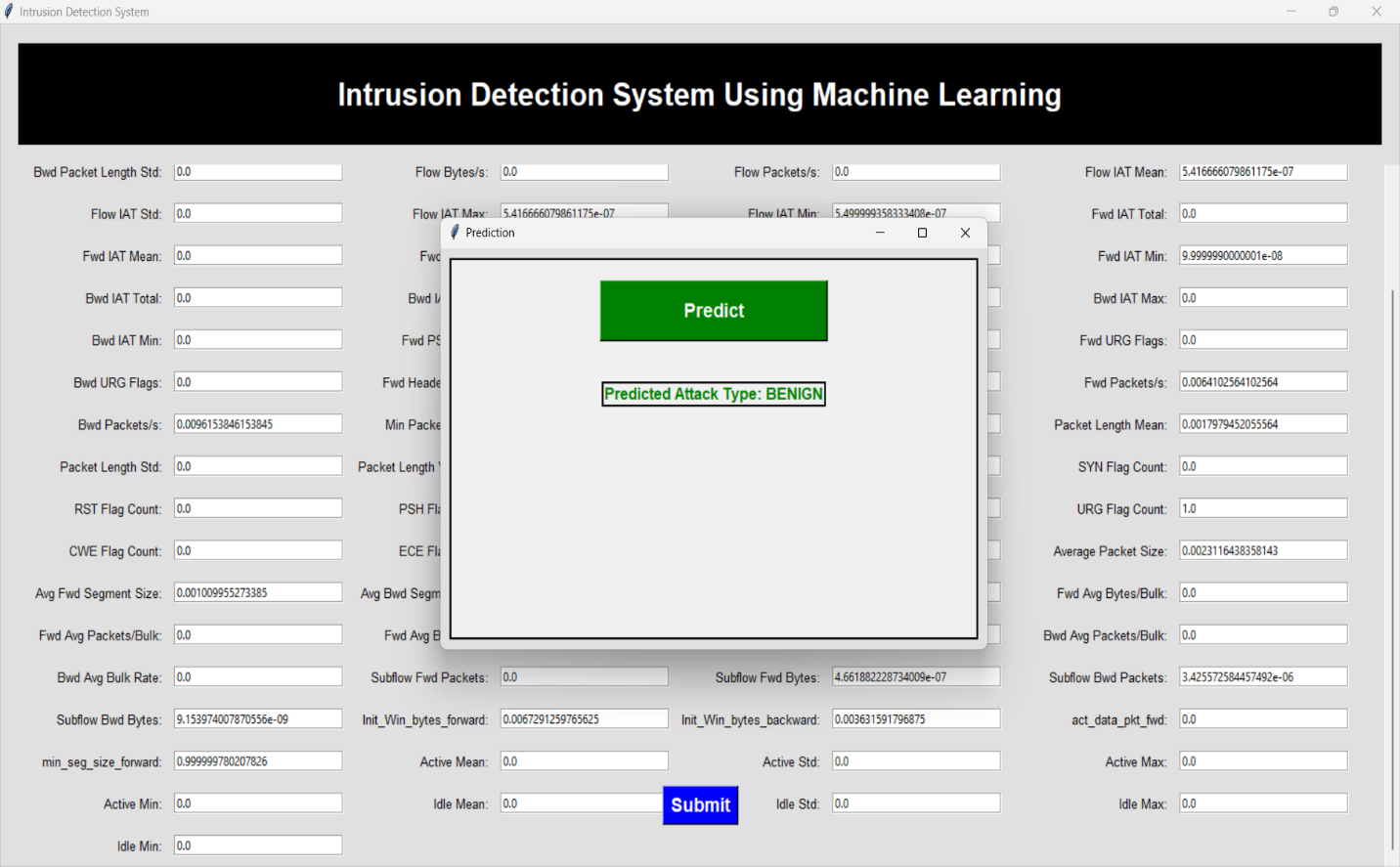
**5.2: Output Screens:**

****

****

****

****

****

**6. CONCLUSION**

## In conclusion, our project introduces an effective Intrusion Detection System (IDS) designed to counter the escalating threat landscape of cyber-attacks. Through the fusion of machine learning models and hyperparameter optimization techniques, we have created a robust framework capable of accurately identifying known signature-based attacks.

## Moreover, the user-friendly interface, coupled with efficient model deployment and packaging, ensures practical application and seamless integration into cybersecurity protocols. By leveraging advanced methodologies and comprehensive datasets, our system stands poised to bolster cybersecurity measures and provide a proactive defense against evolving cyber threats.

**REFERENCES**

1. McHugh, J., Christie, A., & Allen, J. (2000). Defending Yourself: The Role of Intrusion Detection Systems. IEEE Software, 17(5), 42-51.

2. Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5-32.

3. Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785-794.

4. Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. Journal of Machine Learning Research, 13, 281-305.

5. Dua, D., & Graff, C. (2019). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science.

6. Python Software Foundation. (2023). Python Language Reference, Version 3.6. Available online: https://www.python.org/doc/versions/.

7. Tkinter Documentation. (2023). Tkinter GUI Toolkit. Available online: https://docs.python.org/3/library/tkinter.html.

8. Scikit-learn Documentation. (2023). Scikit-learn: Machine Learning in Python. Available online: https://scikit-learn.org/stable/documentation.html.

9. Hyperopt Documentation. (2023). Hyperopt: Distributed Hyperparameter Optimization. Available online: https://github.com/hyperopt/hyperopt/wiki.

10. joblib Documentation. (2023). Joblib: Lightweight Pipelining in Python. Available online: https://joblib.readthedocs.io/en/latest/.