A
PROJECT REPORT
ON

1. NexGen Cybersecurity: Applying Convolutional Neural Networks for Intrusion Detection in SDNs

Introduction

1.1 Introduction

With the development and improvement of Internet technology, the Internet is providing various convenient services for people. However, we are also facing various security threats. Network viruses, eavesdropping, and malicious attacks are on the rise, causing network security to become the focus of attention of society and government departments. Fortunately, these problems can be well solved via intrusion detection. Intrusion detection plays an important part in ensuring network information security. However, with the explosive growth of Internet business, traffic types in the network are increasing day by day, and network behavior characteristics are becoming increasingly complex, which brings great challenges to intrusion detection. How to identify various malicious network traffics, especially unexpected malicious network traffics, is a key problem that cannot be avoided.

Network traffic can be divided into two categories: normal traffic and malicious traffic. Furthermore, network traffic can also be divided into five categories: Normal, DoS (Denial of Service attacks), R2L (Root to Local attacks), U2R (User to Root attack), and Probe (Probing attacks). Hence, intrusion detection can be considered as a classification problem. By improving the performance of classifiers in effectively identifying malicious traffics, intrusion detection accuracy can be largely improved. Machine learning methods have been widely used in intrusion detection to identify malicious traffic. However, these methods belong to shallow learning and often emphasize feature engineering and selection. They have difficulty in features selection and cannot effectively solve the massive intrusion data classification problem, which leads to low recognition accuracy and high false alarm rate.

In recent years, intrusion detection methods based on deep learning have been proposed successively. In this project, we aim to explore intrusion detection methods using Convolutional Neural Networks (CNNs) in Software-Defined Networks (SDNs). CNNs have shown promise in effectively identifying malicious traffic without the need for manual feature engineering. However, existing methods often treat network traffic as a whole, neglecting the hierarchical structure and internal relations of network traffics. By leveraging CNNs and considering the hierarchical structure of network traffic, we aim to improve the accuracy and efficiency of intrusion detection in SDNs.

1.2 PROBLEM STATEMENT:

With the rapid development and wide application of 5G, IoT, Cloud Computing, and other technologies, network scale, and real-time traffic become more complex and massive, cyber-attacks have also become complex and diverse, bringing significant challenges to cyberspace security. As the second line of defense behind the firewall, the Network Intrusion Detection System (NIDS) needs to accurately identify malicious network attacks, provide real-time monitoring and dynamic protection measures, and formulate strategies.

1.3 OBJECTIVE:

In real cyberspace, normal activities occupy the dominant position, so most traffic data are normal traffic; only a few are malicious cyber-attacks, resulting in a high imbalance of categories. In the highly imbalanced and redundant network In real cyberspace, normal activities occupy the dominant position, so most traffic data are normal traffic; only a few are malicious cyber-attacks, resulting in a high imbalance of categories. In the highly imbalanced and redundant network.

2. literature review

2.1 literature review

In recent years, network security has become increasingly important due to rising cyber threats. Intrusion detection systems (IDS) play a crucial role in safeguarding networks against malicious activities. Traditional IDS methods often rely on shallow learning techniques, which struggle with feature selection and classifying massive amounts of intrusion data accurately. To address these challenges, researchers have turned to deep learning approaches, such as Convolutional Neural Networks (CNNs), for intrusion detection. These methods have shown promise in improving detection accuracy without requiring manual feature engineering. For instance, some studies have explored using CNNs to classify malware traffic and detect abnormal network behavior.

Additionally, Recurrent Neural Networks (RNNs) like Long Short-Term Memory (LSTM) networks have been investigated for modeling the sequential nature of network traffic, enabling better understanding of evolving network states over time. However, while these deep learning methods show potential, there's still room for improvement. Many existing approaches overlook the hierarchical structure of network traffic, treating it as a single entity rather than considering the relationships between different levels of traffic units.

In our project, we aim to build upon these advancements by developing a CNN-based intrusion detection system tailored for Software-Defined Networks (SDNs). By leveraging CNNs and considering the hierarchical structure of network traffic, we seek to enhance the accuracy and efficiency of intrusion detection in SDNs, thus contributing to improved network security.

3.SYSTEM ANALYSIS

3.1 SYSTEM ANALYSIS

3.1.1 Existing System:

Deep learning, originating from the pioneering work on Computer Vision and Natural Language Processing, has gained traction in the field of intrusion detection. It involves extracting pertinent features from complex network data through extensive model training. Despite its advancements, existing systems face challenges, particularly in addressing class imbalance within network traffic classification. Ensuring a seamless transition from a "degraded" state to "Normal" before potential system failures poses a critical monitoring challenge, especially in engineering systems like airplanes, where any deviation from expected functionality signifies a failure.

Disadvantages of Existing System:

The primary limitation lies in effectively managing the transition from a degraded state to normal functioning within a system. This monitoring endeavor is vital for maintaining safety and reliability, particularly in contexts like aviation, where system failures can have catastrophic consequences. Defining failure as any deviation from normal system operation underscores the complexity of ensuring system integrity.

3.1.2 Proposed System:

Our proposed system addresses these challenges by utilizing classic benchmark datasets NSL-KDD. We conduct comprehensive data analysis and cleaning to ensure accuracy. The key innovation lies in our machine learning algorithm, which addresses class imbalance in intrusion detection by reducing majority samples and augmenting minority samples in the difficult set. This approach enhances classifier learning during training. Our classification model incorporates Random Forest (RF), Support Vector Machine (SVM), XGBoost, and NLP alongside other methods.

Advantages of Proposed System:

Our method employs fine-grained feature analysis to determine the importance of packet vectors, enhancing the detection of malicious traffic. Additionally, we utilize an attention mechanism to generate features at the output layer, which are then fused in a fully connected layer to extract key features accurately characterizing network traffic behaviors. This approach promises improved accuracy and efficiency in intrusion detection, addressing the limitations of existing systems.

3.2 SYSTEM REQUIREMENTS

3.2.1 Hardware requirements:

➢ System	: Intel(R) Core(TM) i7-13650HX 2.60 GHz
➢ Hard Disk	: 1 TB SSD.
➢ Input Devices	: Keyboard, Mouse
➢ Ram	: 16 GB.

3.2.2 Software requirements:

➢ Operating system	:	Windows 10/11.
➢ Coding Language	:	Python
➢ Tool	:	Anaconda
➢ Interface	:	OPENCV

3.3 MODULES:

3.3.1 Data Collection:

There are three symbolic data types in NSL-KDD data features: protocol type, flag and service. We use one-hot encoder mapping these features into binary vectors. One-Hot Processing: NSL-KDD dataset is processed by one-hot method to transform symbolic features into numerical features. For example, the second feature of the NSL-KDD data sample is protocol type. The protocol type has three values: tcp, udp, and icmp. One-hot method is processed into a binary code that can be recognized by a computer, where tcp is [1, 0, 0], udp is [0, 1, 0], and icmp is [0, 0, 1].

3.3.2 Pre-Processing:

When the dataset is extracted, part of the data contains some noisy data, duplicate values, missing values, infinity values, etc. due to extraction errors or input errors. Therefore, we first perform data preprocessing. The main work is as follows. (1) Duplicate values: delete the sample's duplicate value, only keep one valid data. (2) Outliers: in the sample data, the sample size of missing values(Not a Number, NaN) and Infinite values(Inf) is small, so we delete this. (3) Features delete and transform: In CSE-CIC-IDS2018, we delete features such as ''Timestamp'', ''Destination Address'', ''Source Address'', ''Source Port'', etc. If features ''Init Bwd Win Byts'' and features ''Init Fwd Win Byts'' have a value of −1, we add two check dimensions. The mark of −1 is 1.

Otherwise, it is 0. In NSL-KDD, we use the One Hot encoder to complete this conversion. For example, ‘‘TCP’’, ‘‘UDP’’ and ‘‘ICMP’’ are functions of three protocol types. After OneHot encoding, they become binary vectors (1, 0, 0), (0, 1, 0), (0, 0, 1). The protocol type function can be divided into three categories, including 11 categories for flag function and 70 categories for service function. Therefore, the 41 dimensions initial feature vector becomes 122 dimensions. (4) Numerical standardization: In order to eliminate the dimensional influence between indicators and accelerate the gradient descent and model convergence, the data is standardized, that is, the method of obtaining Z-Score, so that the average value of each feature becomes 0 and the standard deviation becomes 1, converted to a standard normal distribution, which is related to the overall sample distribution, and each sample point can have an impact on standardization. The standardization formula is as follows, u is the mean of each feature, s is the standard deviation of each feature, and x 0 i is the element corresponding to each column’s features.

3.3.3 Train-Test Split and Model fitting:

Now, we divide our dataset into training and testing data. Our objective for doing this split is to assess the performance of our model on unseen data and to determine how well our model has generalized on training data. This is followed by a model fitting which is an essential step in the model building process.

3.3.4 Model Evaluation and Predictions:

This is the final step, in which we assess how well our model has performed on testing data using certain scoring metrics, I have used 'accuracy score' to evaluate my model. First, we create a model instance, this is followed by fitting the training data on the model using a fit method and then we will use the predict method to make predictions on x_test or the testing data, these predictions will be stored in a variable called y_test_hat. For model evaluation, we will feed the y_test and y_test_hat into the accuracy_score function and store it in a variable called test_accuracy, a variable that will hold the testing accuracy of our model. We followed these steps for a variety of classification algorithm models and obtained corresponding test accuracy scores.

3.4 ALGORITHMS:

- **Multinomial Naive Bayes:**

This algorithm is suitable for classification tasks with discrete features (e.g., word counts for text classification).It's commonly used in text classification, spam detection, and other tasks where features represent counts.

- **Gaussian Naive Bayes:**

It's based on Bayes' theorem and assumes that features are continuous and follow a Gaussian distribution. Suitable for classification tasks with continuous features.

- **Bernoulli Naive Bayes:**

Similar to Multinomial Naive Bayes, but it assumes that features are binary (e.g., presence or absence of a word). Often used in text classification tasks with binary features.

- **Support Vector Machine (SVM):**

SVM is a powerful classification algorithm that finds the hyperplane that best separates classes in a high-dimensional space. It's effective in both linear and non-linear classification tasks. SVM aims to maximize the margin between classes, making it robust to overfitting.

- **Multilayer Perceptron (MLP) Classifier:**

MLP is a type of feedforward neural network with one or more hidden layers between the input and output layers. It's capable of learning complex relationships in data and is suitable for various classification tasks. MLPs are known for their ability to approximate any function given enough neurons and layers.

- **AdaBoost Classifier:**

AdaBoost (Adaptive Boosting) is an ensemble learning method that combines multiple weak classifiers to create a strong classifier. It sequentially trains a series of weak learners, focusing on the examples that previous models misclassified. It's particularly effective in binary classification tasks and is less prone to overfitting.

- **Decision Tree Classifier:**

Decision trees recursively split the data based on features to create a tree-like structure for classification. Each internal node represents a feature, each branch represents a decision based on that feature, and each leaf node represents a class label. Decision trees are easy to interpret and visualize, making them useful for understanding feature importance.

- **Random Forest Classifier:**

Random Forest is an ensemble learning method that builds multiple decision trees and combines their predictions through averaging or voting. It reduces overfitting compared to individual decision trees and typically provides higher accuracy. Random Forests are versatile and can handle both classification and regression tasks effectively.

4.System DESIGN

4.1 system architecture

A system architecture or systems architecture is the conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system. Organized in a way that supports reasoning about the structures and behaviors of the system.

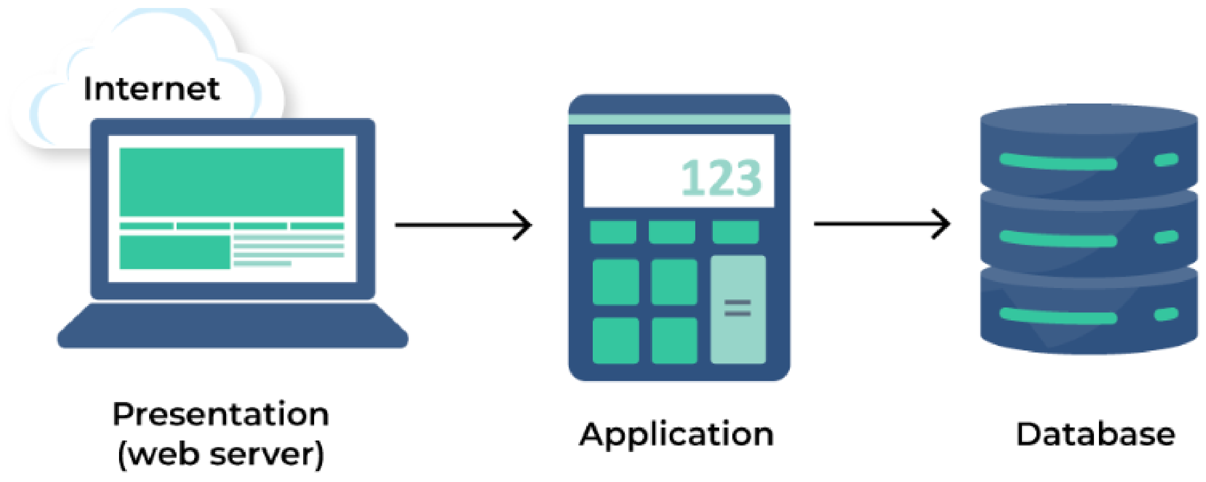


Fig 4.1 System Architecture

3-Tier Architecture:

The three-tier software architecture (a three-layer architecture) emerged in the 1990s to overcome the limitations of the two-tier architecture. The third tier (middle tier server) is between the user interface (client) and the data management (server) components. This middle tier provides process management where business logic and rules are executed and can accommodate hundreds of users (as compared to only 100 users with the two tier architecture) by providing functions such as queuing, application execution, and database staging.The three tier architecture is used when an effective distributed client/server design is needed that provides (when compared to the two tier) increased performance, flexibility, maintainability, reusability, and scalability, while hiding the complexity of distributed processing from the user. These characteristics have made three layer architectures a popular choice for Internet applications and net-centric information systems.

Advantages of Three-Tier:

- Separates functionality from presentation.
- Clear separation – better understanding.
- Changes limited to well define components.
- Can be running on WWW.
- Effective network performance.

4.2 UML DIAGRAMS

4.2.1 Construction of Use Case Diagrams:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

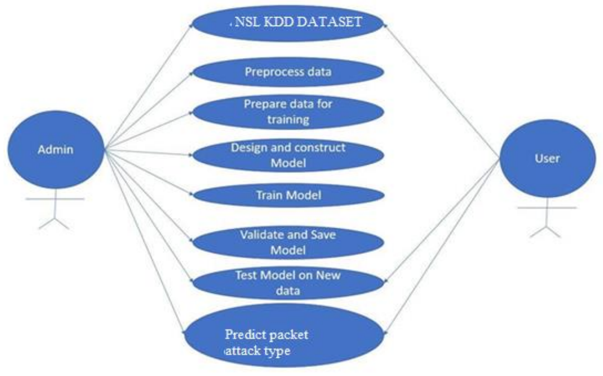


Fig 4.2 Use Case Diagram

4.2.2 Sequence diagrams :

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

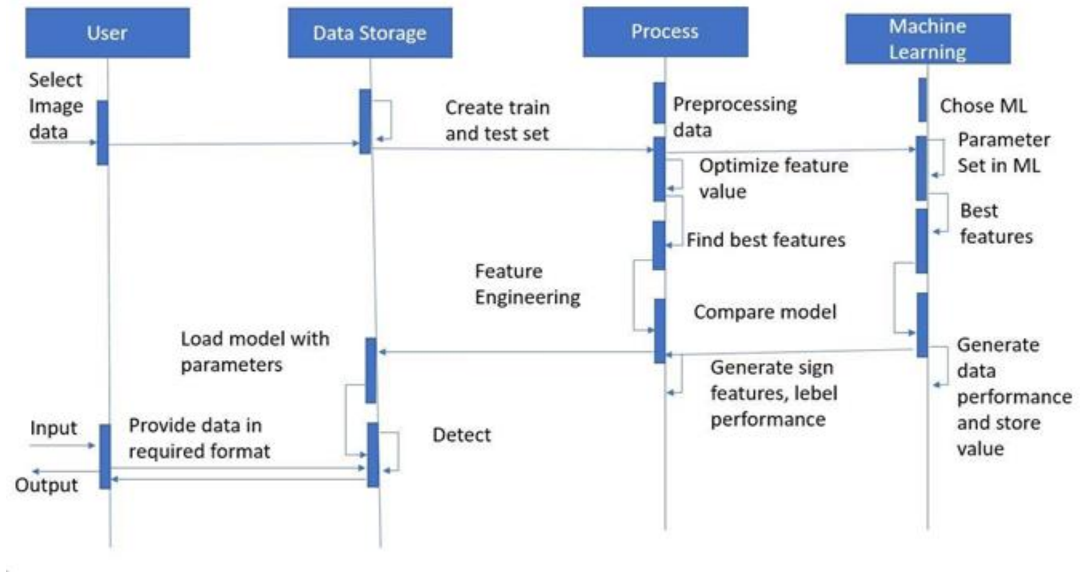


Fig 4.3 Sequence diagram

4.2.3 Class Diagram:

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

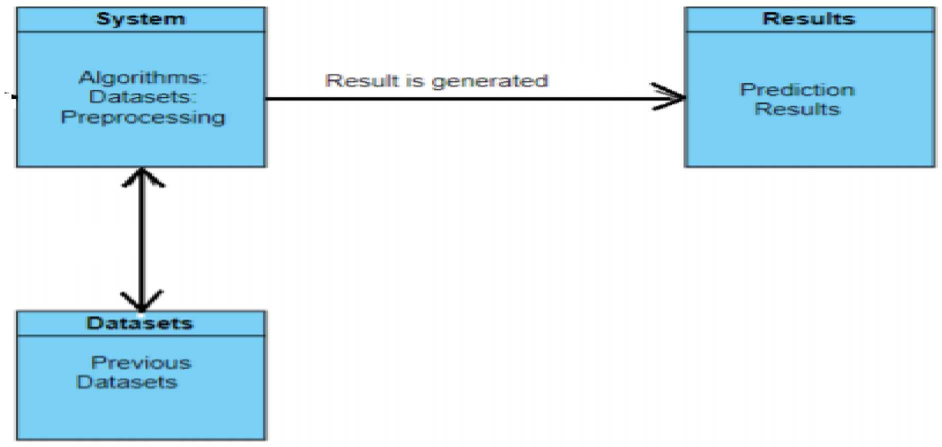


Fig 4.4Class Diagram

4.2.4 Activity Diagram:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

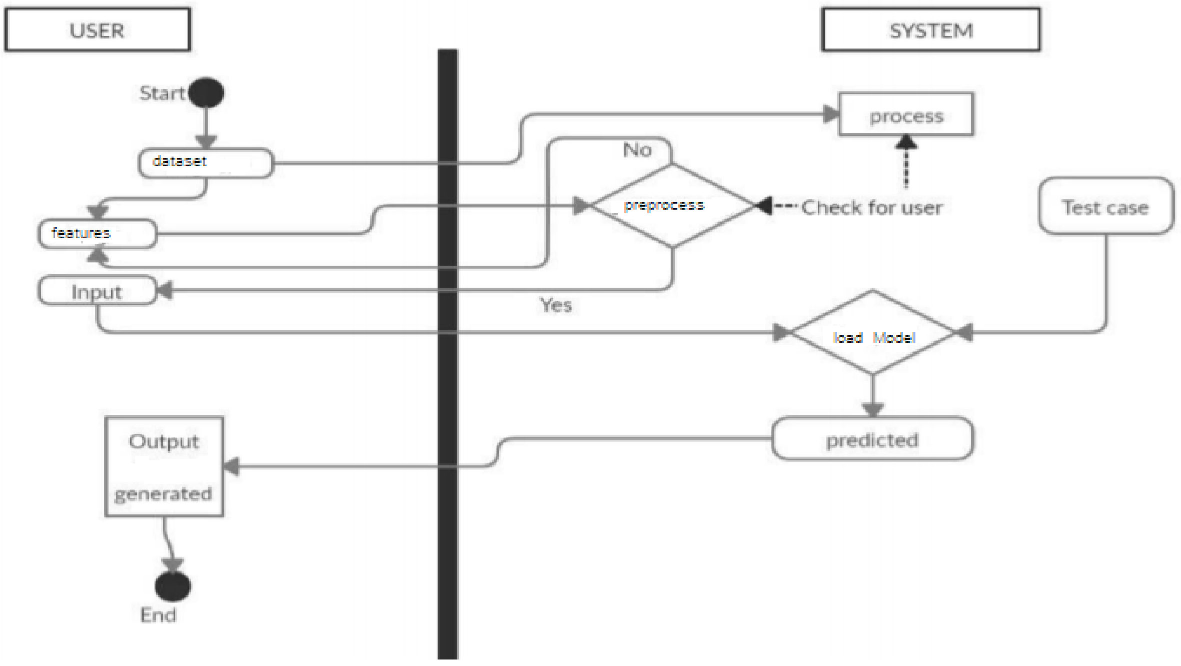


Fig 4.5Activity Diagram

5. System implementation

To conduct studies and analyses of an operational and technological nature, and To promote the exchange and development of methods and tools for operational analysis as applied to defense problems.

5.1 input and output designs

5.1.1 Logical design

The logical design of a system pertains to an abstract representation of the data flows, inputs and outputs of the system. This is often conducted via modeling, using an over-abstract (and sometimes graphical) model of the actual system. In the context of systems design are included. Logical design includes ER Diagrams i.e. Entity Relationship Diagrams

5.1.2 Physical Design

The physical design relates to the actual input and output processes of the system. This is laid down in terms of how data is input into a system, how it is verified / authenticated, how it is processed, and how it is displayed as output. In Physical design, following requirements about the system are decided.

1. Input requirement,
2. Output requirements,
3. Storage requirements,
4. Processing Requirements,
5. System control and backup or recovery.

Put another way, the physical portion of systems design can generally be broken down into three sub-tasks:

1. User Interface Design
2. Data Design
3. Process Design

User Interface Design is concerned with how users add information to the system and with how the system presents information back to them. Data Design is concerned with how the data is represented and stored within the system. Finally, Process Design is concerned with how data moves through the system, and with how and where it is validated, secured and/or transformed as it flows into, through and out of the system. At the end of the systems design phase, documentation describing the three sub-tasks is produced and made available for use in the next phase.

Physical design, in this context, does not refer to the tangible physical design of an information system. To use an analogy, a personal computer's physical design involves input via a keyboard, processing within the CPU, and output via a monitor, printer, etc. It would not concern the actual layout of the tangible hardware, which for a PC would be a monitor, CPU, motherboard, hard drive, modems, video/graphics cards, USB slots, etc. It involves a detailed design of a user and a product database structure processor and a control processor. The H/S personal specification is developed for the proposed system.

5.2 INPUT & OUTPUT REPRESENTATION

5.2.1 Input Design :

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

- What data should be given as input?
- How the data should be arranged or coded?
- The dialog to guide the operating personnel in providing input.
- Methods for preparing input validations and steps to follow when error occur.

5.2.2 Objectives:

Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.

It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.

When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow

5.3 Output Design

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

a. Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.
b. Select methods for presenting information.
c. Create document, report, or other formats that contain information produced by the system.

Code

```
import pandas as pd

from sklearn.model_selection import train_test_split

import numpy as np

from sklearn.utils import shuffle

train=pd.read_csv("multiData.csv")

#train=pd.read_csv("faultData.csv")

train.head()

train = shuffle(train)

train.columns

y_train=train['label2']

#y_train=train['label1']

train=train.drop(['label2'],axis=1)

#train=train.drop(['label1'],axis=1)

X_train, X_test, y_train, y_test = train_test_split(train, y_train, test_size=0.05,
random_state=42)

X_train.info()

from sklearn.ensemble import AdaBoostClassifier

from sklearn.metrics import accuracy_score
```

```
ada= AdaBoostClassifier()

ada.fit(X_train,y_train)

pred=ada.predict(X_test)

print("Adaboost Accuracy Score with Test Data")

accuracy_score(y_test, pred)*100

ada.score(X_train,y_train)*100

# import pickle

# filename = 'rf_model.sav'

# pickle.dump(DecisionTree, open(filename, 'wb'))

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score

DecisionTree= DecisionTreeClassifier()

DecisionTree.fit(X_train,y_train)

pred=DecisionTree.predict(X_test)

print(pred)

print(X_test[0:5])

print("DecisionTreeClassifier Accuracy Score with Test Data")

accuracy_score(y_test, pred)*100

DecisionTree.score(X_train,y_train)
```

```
from sklearn.linear_model import LogisticRegression
logreg= LogisticRegression()


logreg.fit(X_train,y_train)
pred=logreg.predict(X_test)
print("LogisticRegression Accuracy Score with Test Data")
accuracy_score(y_test, pred)
logreg.score(X_train,y_train)
from sklearn.ensemble import RandomForestClassifier
Decisionforest= RandomForestClassifier()
Decisionforest.fit(X_train,y_train)
pred=Decisionforest.predict(X_test)
print(pred)
print(X_test[0:20])
print("RandomForestClassifier Accuracy Score with Test Data")
accuracy_score(y_test, pred)*100
Decisionforest.score(X_train,y_train)
```

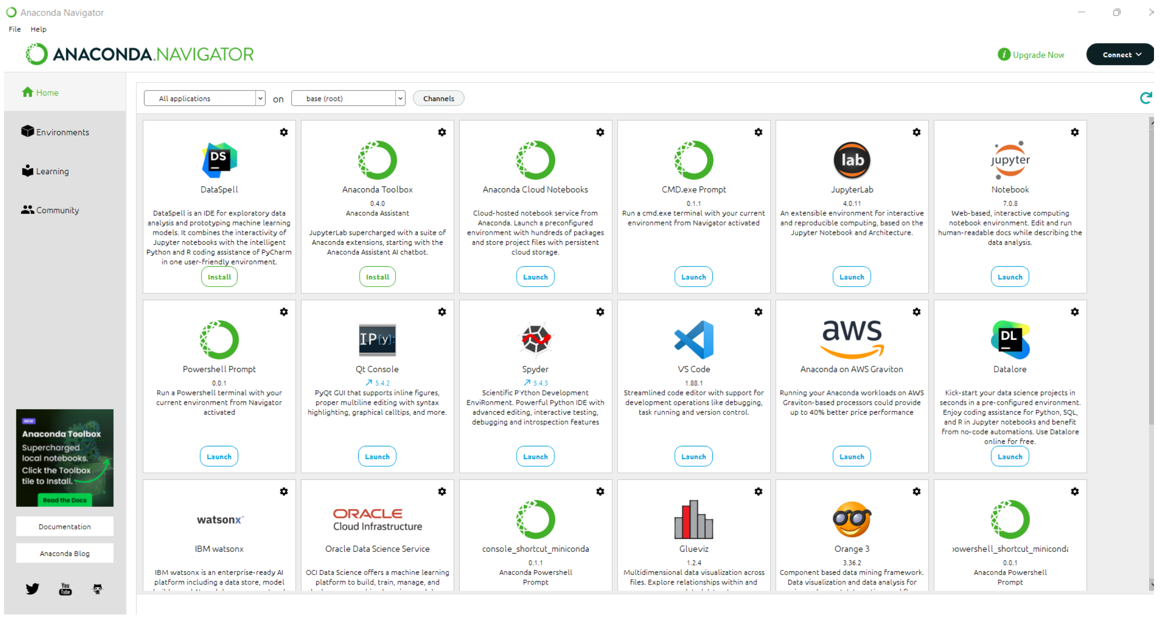
Server.py code

```
import flask

from flask import Flask, render_template, request, redirect

from eval import main

app = Flask(__name__)


#Basic Web Pages

#---------------------------------------------------------------------------------------

@app.route("/")

@app.route("/index")

def index():

    return render_template("index.html")


@app.route("/features")

def features():

    return render_template("features.html")


@app.route("/analysis")

def analysis():

    return render_template("analysis.html")
```

```
@app.route("/model")
def model():
    return render_template("model.html")
#-------------------------------------------------------------------------------------------


@app.route("/submit", methods=['POST'])
def submit():
    if request.method=="POST":
        type = request.form.get("traffic_type")
        expected = type
        type = type.lower()
        print(type)
        attacks = ["normal","dos","r2l","u2r","probe"]
        if type not in attacks:
            return render_template("index.html")
        pred, prob = main(type)
        dict = {"expected":expected,"predictions":attacks[pred], "normal":prob[0], 
"dos":prob[1], "u2r":prob[3], "r2l":prob[2], "probe":prob[4]}

        return render_template("result.html",dict=dict)
```

5.4 SYSTEM IMPLEMENTATION

1. Download Anaconda prompt and install necessary python packages.



DataSpell
DataSpell is an IDE for exploratory data analysis and prototyping machine learning models. It combines the interactivity of Jupyter notebooks with the intelligent Python and R coding assistance of PyCharm in one user-friendly environment.

Anaconda Toolbox
0.4.0
Anaconda Assistant
JupyterLab supercharged with a suite of Anaconda extensions, starting with the Anaconda Assistant AI chatbot.

Anaconda Cloud Notebooks
Cloud-hosted notebook service from Anaconda. Launch a preconfigured environment with hundreds of packages and store project files with persistent cloud storage.

CMD.exe Prompt
0.1.1
Run a cmd.exe terminal with your current environment from Navigator activated

JupyterLab
4.0.11
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook
7.0.8
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Powershell Prompt
0.0.1
Run a Powershell terminal with your current environment from Navigator activated

Qt Console
5.4.2
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Spyder
5.4.3
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

VS Code
1.88.1
Streamlined code editor with support for development operations like debugging, task running and version control.

Anaconda on AWS Graviton
Running your Anaconda workloads on AWS Graviton-based processors could provide up to 40% better price performance

Datalore
Kick-start your data science projects in seconds in a pre-configured environment. Enjoy coding assistance for Python, SQL, and R in Jupyter notebooks and benefit from no-code automations. Use Datalore online for free.

IBM watsonx
IBM watsonx is an enterprise-ready AI platform including a data store, model

Oracle Data Science Service
OCI Data Science offers a machine learning platform to build, train, manage, and

console_shortcut_miniconda
0.1.1
Anaconda Powershell Prompt

Glueviz
1.2.4
Multidimensional data visualization across files. Explore relationships within and

Orange 3
3.36.2
Component based data mining framework. Data visualization and data analysis for

owershell_shortcut_miniconda
0.0.1
Anaconda Powershell Prompt

Fig 5.1 Anaconda Console

2. Use Jupiter Notebook

Jupiter Notebook or so called IPython Notebook is an interactive web based computational mean for starting with Jupiter Notebook documents. The term notebook itself is a huge entity to represent the integration with different entity sets. JSON is the main document form from the same for the execution which follows the brief on the schema and the input and output means. It has high integration with several language set and has various flexibilities with the choices. The extension used for the same is “. ipynb” which runs in this platform. It’s an open-source software package with interactive communication means. It has it’s open standards for the same. It’s an open community best for budding programmers.

3. For Jupiter code to run, open anaconda prompt and run the below command to activate tensor flow in anaconda.

   ---- conda create -n tf tensorflow

   ---- conda activate tf

4. Run flash to set the server because the command sets an environment variable named FLASK_APP to server.py. In Flask, the FLASK_APP environment variable is used to specify the entry point of your Flask application. When you run commands like flask run in the command line, Flask looks for this variable to know which Python file contains your Flask application.

   By setting FLASK_APP=server.py, you're telling Flask that the server.py file is where your Flask application is defined. This allows Flask to locate and run your application correctly when you execute commands like flask run.

   ---- set FLASK_APP=server.py

5. Here we did set up server to run the application where it acts like Intrusion Detection System.

6. So, after setting up the server, run the program code and provide the input values.

7. The output will be show according to the input values provided and the results will show that if the traffic is normal or malicious.

6. System testing

6.1 INTRODUCTION:

Testing is the debugging program is one of the most critical aspects of the computer programming triggers, without programming that works, the system would never produce an output of which it was designed. Testing is best performed when user development is asked to assist in identifying all errors and bugs. The sample data are used for testing. It is not quantity but quality of the data used the matters of testing. Testing is aimed at ensuring that the system was accurately an efficiently before live operation commands.

Testing objectives:

The main objective of testing is to uncover a host of errors, systematically and with minimum effort and time. Stating formally, we can say, testing is a process of executing a program with intent of finding an error.

1 A successful test is one that uncovers an as yet undiscovered error.
2 A good test case is one that has probability of finding an error, if it exists.
3 The test is inadequate to detect possibly present errors.
4 The software more or less confirms to the quality and reliable standards.

6.2 Levels of Testing

Code testing:

This examines the logic of the program. For example, the logic for updating various sample data and with the sample files and directories were tested and verified.

Specification Testing:

Executing this specification starting what the program should do and how it should performed under various conditions. Test cases for various situation and combination of conditions in all the modules are tested.

Unit testing:

In the unit testing we test each module individually and integrate with the overall system. Unit testing focuses verification efforts on the smallest unit of software design in the module. This is also known as module testing. The module of the system is tested separately. This testing is carried out during programming stage itself. In the testing step each module is found to work satisfactorily as regard to expected output from the module. There are some validation checks for fields also. For example the validation check is done for varying the user input given by the user which validity of the data entered. It is very easy to find error debut the system.

Each Module can be tested using the following two Strategies:

1 Black Box Testing
2 White Box Testing

6.2.1 BLACK BOX TESTING

What is Black Box Testing?

Black box testing is a software testing techniques in which functionality of the software under test (SUT) is tested without looking at the internal code structure, implementation details and knowledge of internal paths of the software. This type of testing is based entirely on the software requirements and specifications.

In Black Box Testing we just focus on inputs and output of the software system without bothering about internal knowledge of the software program.

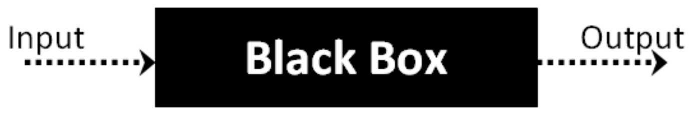


Fig 6.1 Black Box Testing

The above Black Box can be any software system you want to test. For example : an operating system like Windows, a website like Google ,a database like Oracle or even your own custom application. Under Black Box Testing , you can test these applications by just focusing on the inputs and outputs without knowing their internal code implementation.

Black box testing - Steps

Here are the generic steps followed to carry out any type of Black Box Testing.

- Initially requirements and specifications of the system are examined.
- Tester chooses valid inputs (positive test scenario) to check whether SUT processes them correctly. Also some invalid inputs (negative test scenario) are chosen to verify that the SUT is able to detect them.
- Tester determines expected outputs for all those inputs.
- Software tester constructs test cases with the selected inputs.
- The test cases are executed.

- Software tester compares the actual outputs with the expected outputs.
- Defects if any are fixed and re-tested.

Types of Black Box Testing

There are many types of Black Box Testing but following are the prominent ones -

- Functional testing – This black box testing type is related to functional requirements of a system; it is done by software testers.
- Non-functional testing – This type of black box testing is not related to testing of a specific functionality, but non-functional requirements such as performance, scalability, usability.
- Regression testing – Regression testing is done after code fixes , upgrades or any other system maintenance to check the new code has not affected the existing code.

6.2.2 WHITE BOX TESTING

White Box Testing is the testing of a software solution's internal coding and infrastructure. It focuses primarily on strengthening security, the flow of inputs and outputs through the application, and improving design and usability.White box testing is also known as clear, open, structural, and glass box testing.

It is one of two parts of the "box testing" approach of software testing. Its counter-part, blackbox testing, involves testing from an external or end-user type perspective. On the other hand, Whitebox testing is based on the inner workings of an application and revolves around internal testing. The term "whitebox" was used because of the see-through box concept. The clear box or whitebox name symbolizes the ability to see through the software's outer shell (or "box") into its inner workings. Likewise, the "black box" in "black box testing" symbolizes not being able to see the inner workings of the software so that only the end-user experience can be tested

WHAT DO YOU VERIFY IN WHITE BOX TESTING?

White box testing involves the testing of the software code for the following:

- Internal security holes

- Broken or poorly structured paths in the coding processes
- The flow of specific inputs through the code
- Expected output
- The functionality of conditional loops
- Testing of each statement, object and function on an individual basis

The testing can be done at system, integration and unit levels of software development. One of the basic goals of whitebox testing is to verify a working flow for an application. It involves testing a series of predefined inputs against expected or desired outputs so that when a specific input does not result in the expected output, you have encountered a bug.

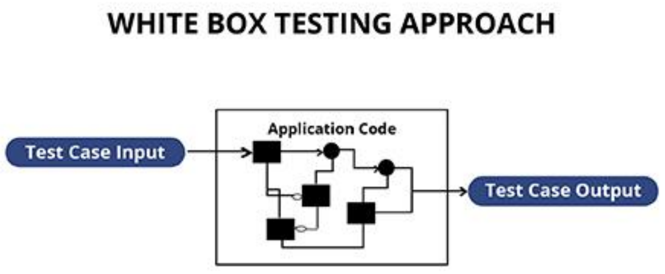


Fig 6.2 White Box Testing

HOW DO YOU PERFORM WHITE BOX TESTING?

To give you a simplified explanation of white box testing, we have divided it into ***two basic steps***. This is what testers do when testing an application using the white box testing technique:

STEP 1) UNDERSTAND THE SOURCE CODE

The first thing a tester will often do is learn and understand the source code of the application. Since white box testing involves the testing of the inner workings of an application, the tester must be very knowledgeable in the programming languages used in the applications they are testing. Also, the testing person must be highly aware of secure coding practices. Security is often one of the primary objectives of testing software. The tester should be able to find security issues and prevent attacks from hackers and naive users who might inject malicious code into the application either knowingly or unknowingly.

Step 2) CREATE TEST CASES AND EXECUTE

The second basic step to white box testing involves testing the application's source code for proper flow and structure. One way is by writing more code to test the application's source code. The tester will develop little tests for each process or series of processes in the application. This method requires that the tester must have intimate knowledge of the code and is often done by the developer. Other methods include manual testing, trial and error testing and the use of testing tools as we will explain further on in this article.

Unit testing:

Sl # Test Case :	UTC1
Name of Test:	Load dataset
Items being tested:	Dataset features and labels are displayed or not
Sample Input:	Dataset csv file
Expected output:	All features and labels should be displayed
Actual output:	Total data is displayed
Remarks:	**Pass.**

Table 6.1 Test Case UTC1

Sl # Test Case :	UTC2
Name of Test:	Split data
Items being tested:	Data is divided in to train and test set
Sample Input:	Test and train size
Expected output:	Dataset is divided in to 2 parts
Actual output:	Based on given test size data is divided and stored in train and test sets
Remarks:	pass

Table 6.2 Test Case UTC2

Integration Testing:

Integration testing is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interactionbetween integrated units. Test drivers and test stubs are used to assist in

Integration Testing. Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. It occurs after unit testing and before validation testing. Integration testing can be done in two ways: Bottom-up integration testing and Top-down integration testing.

Bottom-up Integration*:*

This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.

Top-down Integration*:*

In this testing, the highest-level modules are tested first and progressively, lower-level modules are tested thereafter.

In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple tests of the complete application, preferably in scenarios designed to mimic actual situations. Table 6.5shows the test cases for integration testing and their results

Sl # Test Case :	ITC1
Name of Test:	Train Model
Item being tested:	Model fit is performed
Sample Input:	Train x and train y
Expected output:	Fit is performed
Actual output:	Training is done and accuracy is displayed
Remarks:	Pass.

Table 6.3 Test Case ITC1

Sl # Test Case :	ITC2
Name of Test:	Accuracy calculation
Item being tested:	If accuracy of each algorithm is calculated
Sample Input:	Test x and test y
Expected output:	Accuracy of each algorithm
Actual output:	Accuracy of each model
Remarks:	Pass.

Table 6.4 Test Case ITC2

System testing:

System testing of software or hardware is testing conducted on a complete, integratedsystem to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black-box testing, and as such, should require no knowledge of the inner design of the code or logic. System testing is important becauseof the following reasons:

1. System testing is the first step in the Software Development Life Cycle, where the application is tested as a whole.

2. The application is tested thoroughly to verify that it meets the functional and technical specifications.

3. The application is tested in an environment that is very close to the production environment where the application will be deployed.

4. System testing enables us to test, verify, and validate both the business requirements as well as the application architecture.

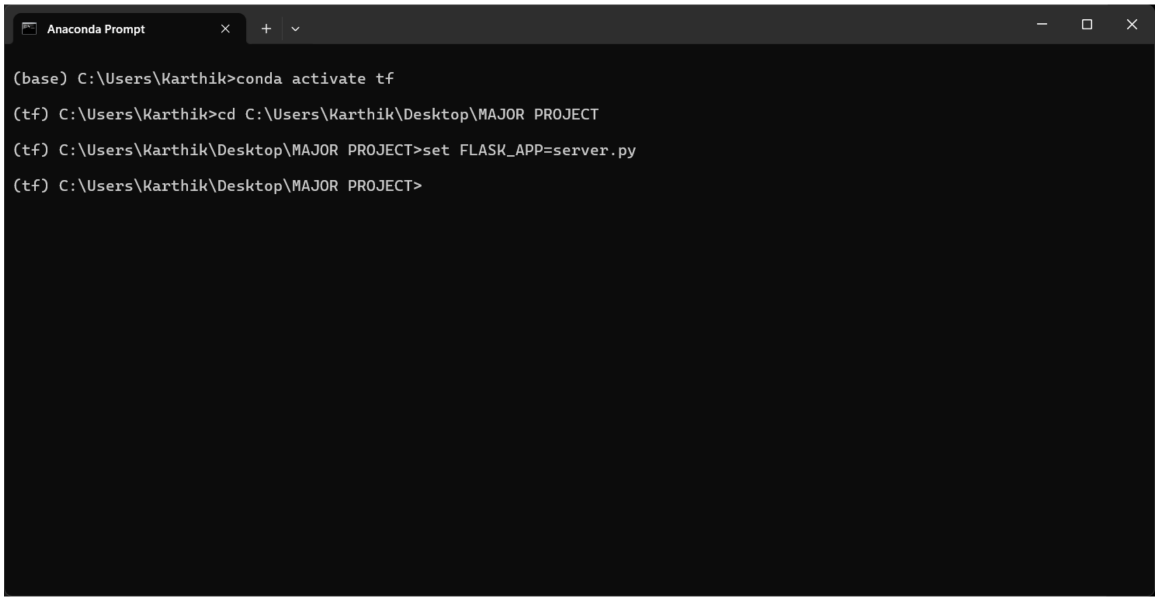
System Testing is shown in below tables

Sl # Test Case : -	STC-1
Name of Test: -	System testing in various versions of OS
Item being tested: -	OS compatibility.
Sample Input: -	Execute the program in windows XP/ Windows-7/8
Expected output: -	Performance is better in windows-7
Actual output: -	Same as expected output, performance is better inwindows-7
Remarks: -	Pass

Table 6.5 Test Case STC1

7. Output Screens

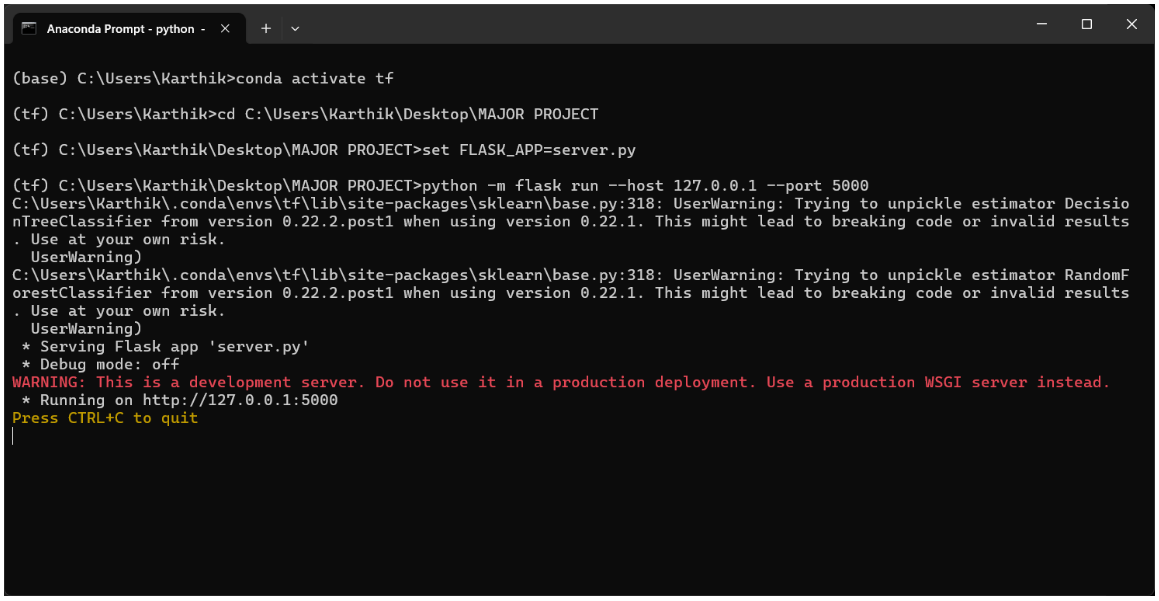
1. Anaconda prompt to activate tensor flow and set up the server using flask.



```
(base) C:\Users\Karthik>conda activate tf

(tf) C:\Users\Karthik>cd C:\Users\Karthik\Desktop\MAJOR PROJECT

(tf) C:\Users\Karthik\Desktop\MAJOR PROJECT>set FLASK_APP=server.py

(tf) C:\Users\Karthik\Desktop\MAJOR PROJECT>
```

Fig 7.1 Activate tensor flow and set up server

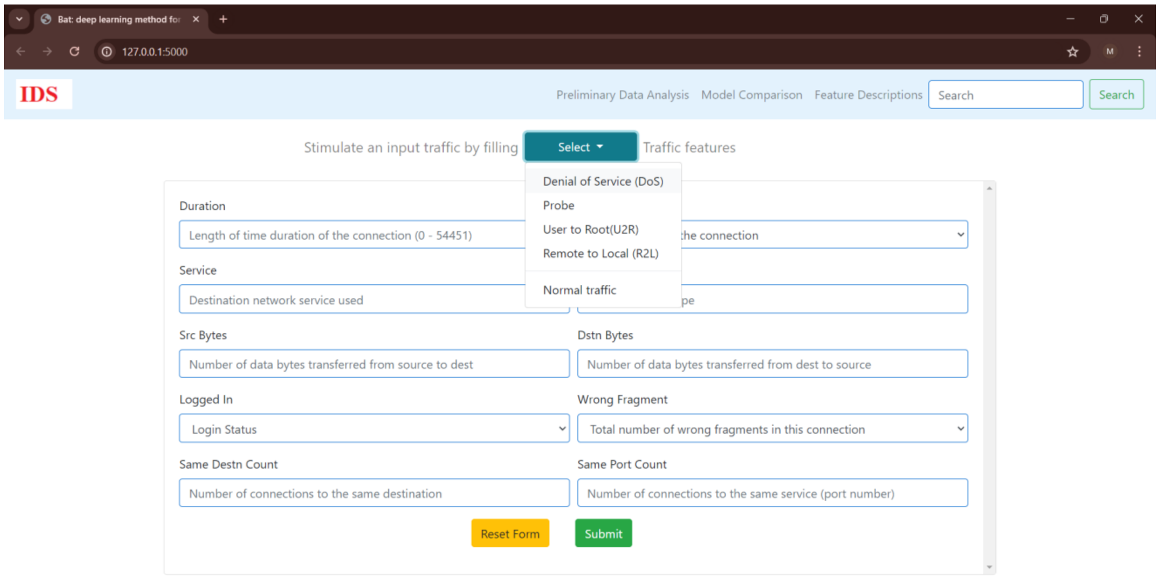
2. Assign the host and port to the flask to open the application.



```
Anaconda Prompt - python -

(base) C:\Users\Karthik>conda activate tf

(tf) C:\Users\Karthik>cd C:\Users\Karthik\Desktop\MAJOR PROJECT

(tf) C:\Users\Karthik\Desktop\MAJOR PROJECT>set FLASK_APP=server.py

(tf) C:\Users\Karthik\Desktop\MAJOR PROJECT>python -m flask run --host 127.0.0.1 --port 5000
C:\Users\Karthik\.conda\envs\tf\lib\site-packages\sklearn\base.py:318: UserWarning: Trying to unpickle estimator Decisio
nTreeClassifier from version 0.22.2.post1 when using version 0.22.1. This might lead to breaking code or invalid results
. Use at your own risk.
  UserWarning)
C:\Users\Karthik\.conda\envs\tf\lib\site-packages\sklearn\base.py:318: UserWarning: Trying to unpickle estimator RandomF
orestClassifier from version 0.22.2.post1 when using version 0.22.1. This might lead to breaking code or invalid results
. Use at your own risk.
  UserWarning)
 * Serving Flask app 'server.py'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Fig 7.2 Assign host and port

3. Provide the input values in the form the results



Bat: deep learning method for

127.0.0.1:5000

IDS   Preliminary Data Analysis   Model Comparison   Feature Descriptions   Search   Search

Stimulate an input traffic by filling [Select ▾] Traffic features

Denial of Service (DoS)
Probe
User to Root(U2R)
Remote to Local (R2L)
Normal traffic

Duration: Length of time duration of the connection (0 - 54451)

...the connection

Service: Destination network service used

...pe

Src Bytes: Number of data bytes transferred from source to dest

Dstn Bytes: Number of data bytes transferred from dest to source

Logged In: Login Status

Wrong Fragment: Total number of wrong fragments in this connection

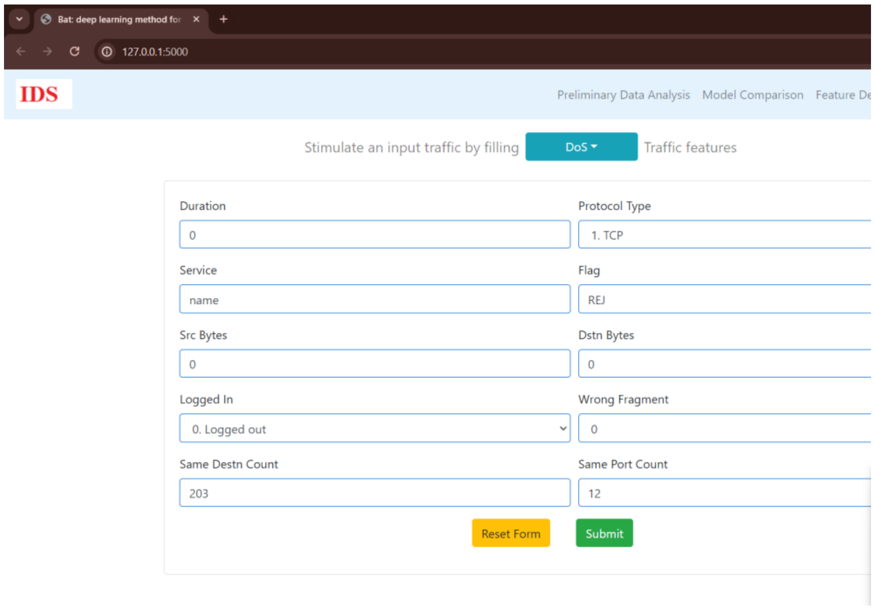
Same Destn Count: Number of connections to the same destination

Same Port Count: Number of connections to the same service (port number)

Reset Form   Submit

Fig 7.3 Traffic features

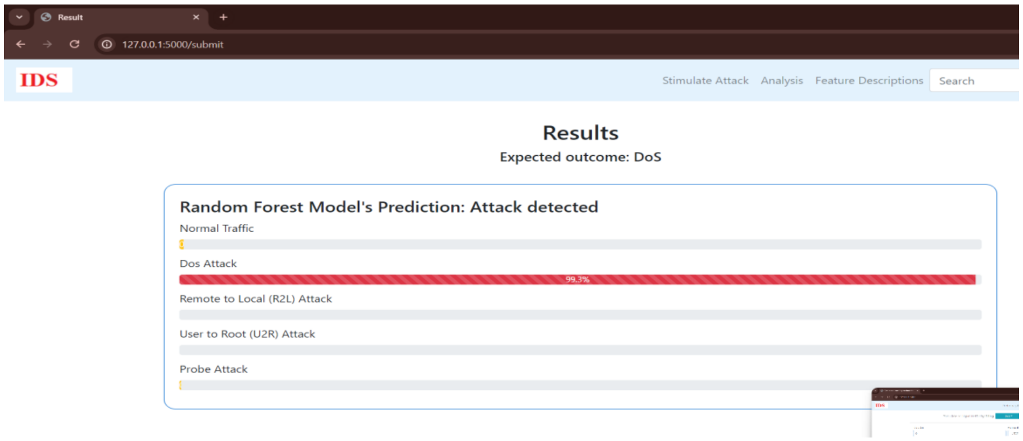
4. Select and attack type and provide the input values in the fields.


Bat: deep learning method for
127.0.0.1:5000
IDS
Preliminary Data Analysis
Model Comparison
Feature De
Stimulate an input traffic by filling
DoS
Traffic features
Reset Form
Submit

Duration	Protocol Type
0	1. TCP
Service	Flag
name	REJ
Src Bytes	Dstn Bytes
0	0
Logged In	Wrong Fragment
0. Logged out	0
Same Destn Count	Same Port Count
203	12

Fig 7.4 DoS Input Traffic

5. So the results will look like :



Results

Expected outcome: DoS

Random Forest Model's Prediction: Attack detected

Normal Traffic

Dos Attack

99.3%

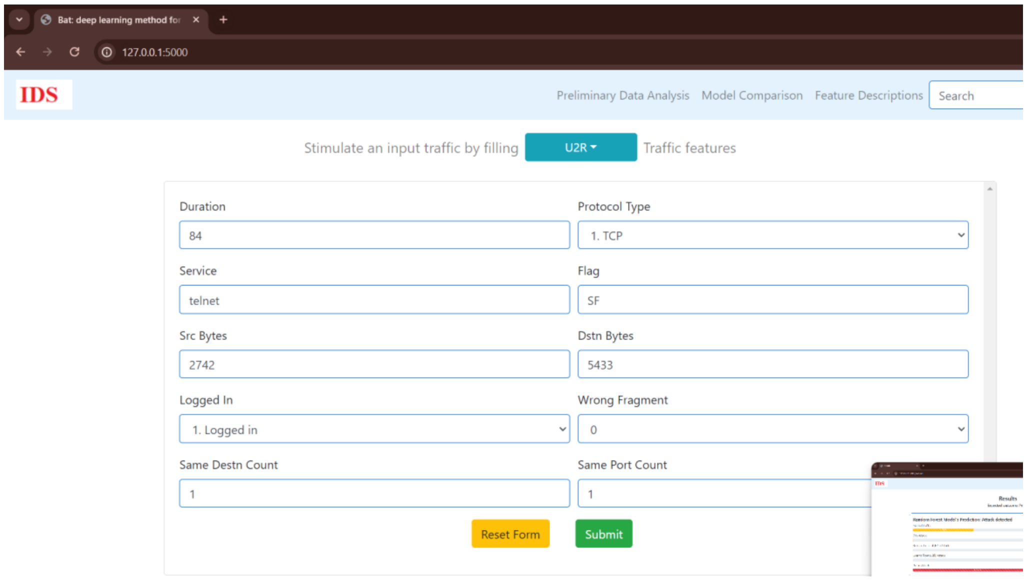
Remote to Local (R2L) Attack

User to Root (U2R) Attack

Probe Attack

Fig 7.5 DoS Results

6.

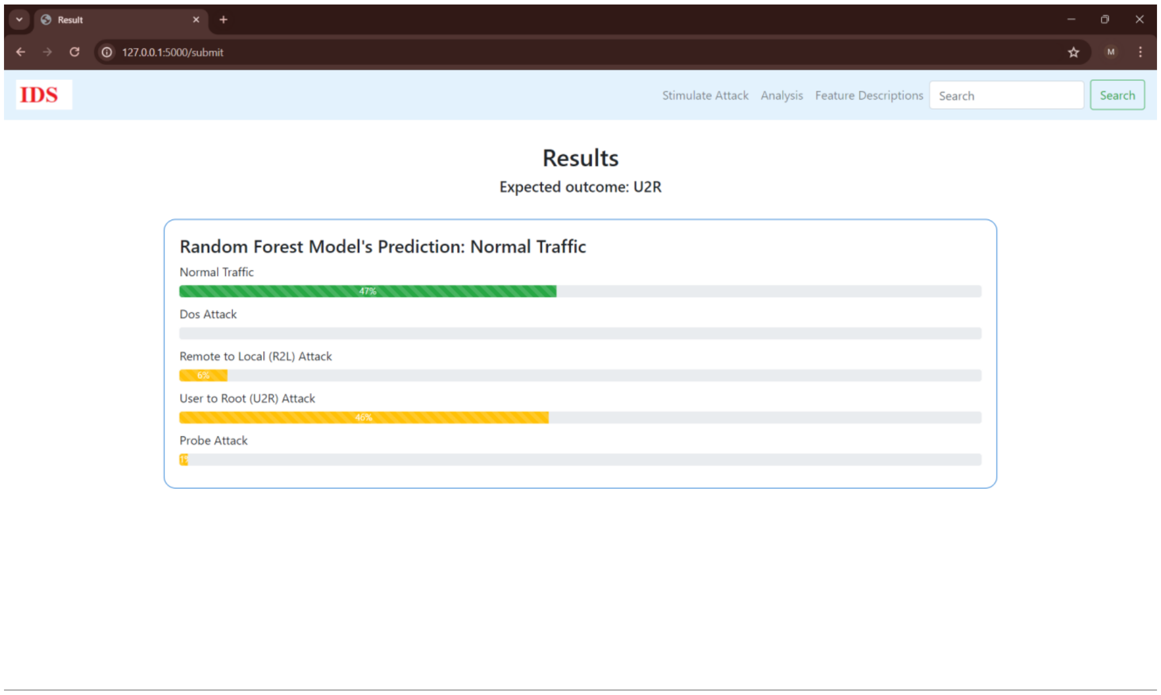


Stimulate an input traffic by filling U2R Traffic features

Duration: 84	Protocol Type: 1. TCP
Service: telnet	Flag: SF
Src Bytes: 2742	Dstn Bytes: 5433
Logged In: 1. Logged in	Wrong Fragment: 0
Same Destn Count: 1	Same Port Count: 1

Fig 7.6 U2R Input Traffic

7.



Results

Expected outcome: U2R

Random Forest Model's Prediction: Normal Traffic

Normal Traffic 47%

Dos Attack

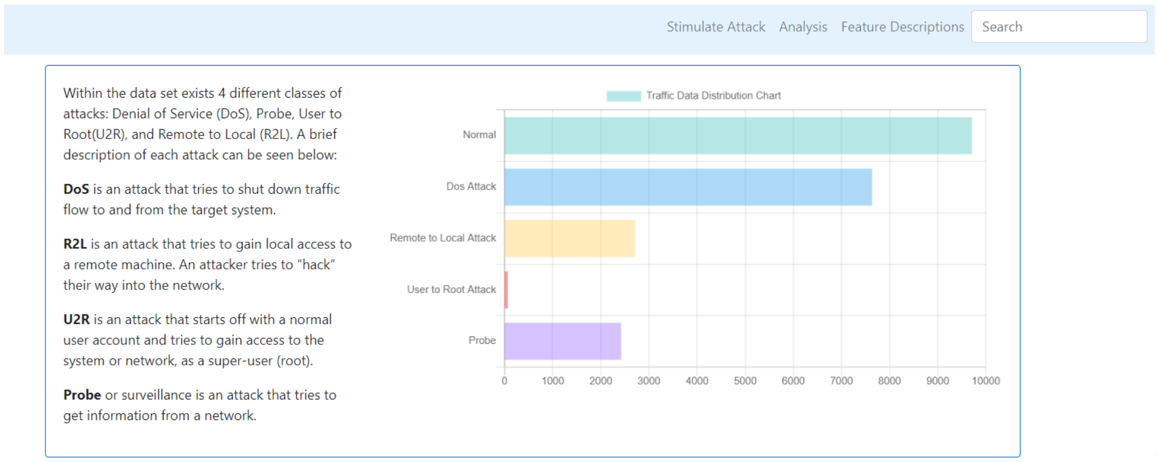
Remote to Local (R2L) Attack 6%

User to Root (U2R) Attack 46%

Probe Attack 1%

Fig 7.7 U2R Results

8. The Analyse of complete traffic looks like



Within the data set exists 4 different classes of attacks: Denial of Service (DoS), Probe, User to Root(U2R), and Remote to Local (R2L). A brief description of each attack can be seen below:

DoS is an attack that tries to shut down traffic flow to and from the target system.

R2L is an attack that tries to gain local access to a remote machine. An attacker tries to "hack" their way into the network.

U2R is an attack that starts off with a normal user account and tries to gain access to the system or network, as a super-user (root).

Probe or surveillance is an attack that tries to get information from a network.

Fig 7.8

Analysis Data

8.CONCLUSION

As network intrusion continues to evolve, the pressure on network intrusion detection is also increasing. In particular, the problems caused by imbalanced network traffic make it difficult for intrusion detection systems to predict the distribution of malicious attacks, making cyberspace security face a considerable threat. This paper proposed a novel Difficult Set Sampling Technique, which enables the classification model to strengthen imbalanced network data learning. A targeted increase in the number of minority samples that need to be learned can reduce the imbalance of network traffic and strengthen the minority's learning under challenging samples to improve the classification accuracy. We used six classical classification methods in machine learning and deep learning and combined them with other sampling techniques. Experiments show that our method can accurately determine the samples that need to be expanded in the imbalanced network traffic and improve the attack recognition more effectively. In the experiment, we found that deep learning performance is better than machine learning after sampling the imbalanced training set samples through the MLP algorithm. Although the neural networks strengthen data expression, the current public datasets have already extracted the data features in advance, which is more limited for deep learning to learn the preprocessed features and cannot take advantage of its automatic feature extraction. Therefore, in the next step, we plan to directly use the deep learning model for feature extraction and model training on the original network traffic data, performance the advantages of deep learning in feature extraction, reduce the impact of imbalanced data and achieve more accurate classification.

9.REFERENCES

[1] D. E. Denning, ‘‘An intrusion-detection model,’’ IEEE Trans. Softw. Eng., vol. SE-13, no. 2, pp. 222–232, Feb. 1987.

[2] N. B. Amor, S. Benferhat, and Z. Elouedi, ‘‘Naive Bayes vs decision trees in intrusion detection systems,’’ in Proc. ACM Symp. Appl. Comput. (SAC), 2004, pp. 420–424.

[3] M. Panda and M. R. Patra, ‘‘Network intrusion detection using Naive Bayes,’’ Int. J. Comput. Sci. Netw. Secur., vol. 7, no. 12, pp. 258–263, 2007.

[4] M. A. M. Hasan, M. Nasser, B. Pal, and S. Ahmad, ‘‘Support vector machine and random forest modeling for intrusion detection system (IDS),’’ J. Intell. Learn. Syst. Appl., vol. 6, no. 1, pp. 45–52, 2014.

[5] N. Japkowicz, ‘‘The class imbalance problem: Significance and strategies,’’ in Proc. Int. Conf. Artif. Intell., vol. 56, 2000, pp. 111–117.

[6] Y. LeCun, Y. Bengio, and G. Hinton, ‘‘Deep learning,’’ Nature, vol. 521, no. 7553, pp. 436–444, 2015.

[7] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, ‘‘Deep learning for visual understanding: A review,’’ Neurocomputing, vol. 187, pp. 27–48, Apr. 2016.

[8] T. Young, D. Hazarika, S. Poria, and E. Cambria, ‘‘Recent trends in deep learning based natural language processing [review article],’’ IEEE Comput. Intell. Mag., vol. 13, no. 3, pp. 55–75, Aug. 2018.

[9] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, ‘‘A deep learning approach to network intrusion detection,’’ IEEE Trans. Emerg. Topics Comput. Intell., vol. 2, no. 1, pp. 41–50, Feb. 2018.

[10] D. A. Cieslak, N. V. Chawla, and A. Striegel, ‘‘Combating imbalance in network intrusion datasets,’’ in Proc. IEEE Int. Conf. Granular Comput., May 2006, pp. 732–737.

Group Contributions:

HARSHINI GOUD NADIKUDE =Y00867056 -Project selection , uml Diagrams, system design,testing Output screens ,ppt, documentation.

Akhila –Integration Testing, System Implementation , black box testing,whitebox testing.

Pravalikha-References, ppts.