



ISSN: 2454-9940



**INTERNATIONAL JOURNAL OF APPLIED
SCIENCE ENGINEERING AND MANAGEMENT**

E-Mail :
editor.ijasem@gmail.com
editor@ijasem.org

www.ijasem.org

Standard Latent Space Dimension for Network Intrusion Detection Systems Datasets

P. SIVA PRASAD, Assistant Professor, Dept of MCA, Chirala Engineering College, Chirala,
Lakshmi prasad8216@gmail.com

MABU SHAIK, PG Student - MCA, Dept of MCA, Chirala Engineering College, Chirala,
mabushaik123456789@gmail.com

ABSTRACT: This project addresses the challenges of Network Intrusion Detection Systems (NIDS) by introducing a standard latent space dimension through autoencoder-based dimensionality reduction techniques. By focusing on dimensionality reduction and standardization within NIDS datasets, the project aims to enhance computational efficiency and provide a standardized framework for evaluating NIDS models. Various network security datasets are projected into this standardized latent space for evaluation, with experimental validation showing that machine learning classification models trained with this standard dimension perform comparably to those using non-reduced datasets in terms of F1-score and accuracy. The introduction of ensemble methods such as Voting Classifier with RF + Adaboost further enhances performance, achieving promising results. Additionally, the project extends to building a user-friendly front end using the Flask framework for user testing, complete with user authentication. Overall, this research contributes to improving NIDS performance and standardization while offering a common approach for researchers in the field of network security, with minimal compromise on model accuracy.

INDEX TERMS: *Standardization, machine learning, autoencoder, latent space, network security*

1. INTRODUCTION:

The proliferation of the Internet has catalyzed a profound transformation in communication networks, rendering them heterogeneous, dynamic, and inherently complex [1]. Traditional techniques for designing, deploying, managing, and maintaining such networks have become increasingly inadequate in addressing the evolving demands and intricacies of modern communication systems [1]. In response to these challenges, machine learning (ML) has emerged as a powerful paradigm capable of tackling complex problems in various domains, including classification, regression, and decision-making [1].

According to Wang et al. [1], one of the primary advantages of ML lies in its ability to address complex problems with results comparable to, or even surpassing, those achieved by human experts. ML techniques have exhibited notable maturity in domains such as computer vision and natural language understanding [2][3]. However, their

application in communication networks remains at an early stage [1]. Despite the vast potential of ML to revolutionize network management and optimization, several significant barriers impede its widespread adoption and efficacy.

A critical challenge hindering the advancement of ML in communication networks is the scarcity of publicly available and rich network traffic datasets [4]. Barut et al. [4] highlight the dearth of comprehensive open datasets, which severely limits the evaluation of ML-based proposals for network traffic analytics and impedes the reproducibility of state-of-the-art results. Moreover, existing open datasets primarily focus on security topics, neglecting other crucial aspects such as quality of service (QoS) provision [4]. Consequently, the development of ad-hoc network datasets tailored to specific applications is imperative to facilitate the effective application of ML techniques in enhancing network performance and functionality.

In addition to dataset availability, there exists a lack of consensus regarding the selection and representation of features in network datasets [5]. Sarhan et al. [5] note that network features are often chosen based on the author's domain knowledge and the available data collection tools, resulting in significant variations in feature sets across different datasets. As a consequence, each dataset captures only a subset of the security events that could potentially be identified, limiting the effectiveness of ML algorithms in network analysis and intrusion detection [5]. Holland et al. [6] emphasize the crucial role of feature exploration and engineering in determining the efficacy of ML algorithms in network applications. Thus, the identification and incorporation of the most

relevant features into network datasets are essential for ensuring optimal model performance and accuracy [6].

2. LITERATURE SURVEY

Machine learning (ML) has garnered considerable attention in recent years for its potential to address complex problems across various domains, including networking. Wang et al. [1] provide a comprehensive overview of the application of ML in networking, highlighting its workflow, recent advances, and potential opportunities. They emphasize ML's capability to address classification, regression, and decision-making tasks with results comparable to, or even better than, those achieved by human experts. While ML techniques have gained maturity in domains like computer vision [2] and natural language understanding [3], their application in communication networks remains at an early stage [1].

A significant challenge hindering the progress of ML in networking is the lack of publicly available and rich network traffic datasets [4]. Barut et al. [4] discuss the limitations imposed by the scarcity of comprehensive open datasets on the evaluation of ML-based proposals for network traffic analytics. They emphasize the need for diverse datasets that go beyond security topics to enable the effective application of ML in improving aspects such as quality of service (QoS) provision.

Furthermore, there is a lack of consensus regarding the selection and representation of features in network datasets, particularly in the context of Network Intrusion Detection Systems (NIDSs) [5]. Sarhan et al. [5] highlight the variability in feature sets across different datasets, as features are often

selected based on the author's domain knowledge and available data collection tools. This variability affects the efficacy of ML algorithms in network analysis and intrusion detection, underscoring the importance of standardizing feature sets to ensure optimal model performance [5].

Feature selection and representation play a crucial role in determining the effectiveness of ML algorithms in network applications. Holland et al. [6] emphasize the significance of feature exploration and engineering in automated traffic analysis, as the selection and representation of network traffic features significantly impact model performance. To address this challenge, researchers have explored various dimensionality reduction techniques to enhance the efficiency and effectiveness of ML algorithms in processing high-dimensional data.

Dimensionality reduction techniques such as autoencoders have gained attention for their ability to reduce the dimensionality of data while preserving its essential information [7]. Li et al. [7] propose a fast hybrid dimensionality reduction method based on feature selection and grouped feature extraction, aiming to improve classification performance. Van Der Maaten et al. [8] provide a comparative analysis of different dimensionality reduction techniques, highlighting their strengths and limitations in various applications. Fournier and Aloise [9] empirically compare autoencoders with traditional dimensionality reduction methods, exploring their effectiveness in reducing the dimensionality of data. Additionally, Janakiramaiah et al. [10] investigate the use of autoencoders for reducing the dimensionality of data, highlighting their potential for enhancing the efficiency of ML algorithms in processing high-dimensional data.

Overall, the literature survey highlights the challenges and opportunities in applying ML techniques to address complex problems in communication networks. It underscores the importance of dataset availability, feature selection, and dimensionality reduction techniques in enhancing the effectiveness and efficiency of ML algorithms in network analysis and optimization. By addressing these challenges and leveraging emerging techniques, researchers can unlock the full potential of ML in revolutionizing communication network management and optimization.

3. METHODOLOGY

a) Proposed work:

The proposed work aims to enhance Network Intrusion Detection Systems (NIDS) by introducing a standardized latent space dimension through autoencoder-based dimensionality reduction. This approach seeks to improve computational efficiency and standardize the evaluation process for NIDS datasets. An extension of the project involves implementing ensemble methods to further enhance prediction accuracy. Specifically, a Voting Classifier with Random Forest and Adaboost, as well as a Stacking Classifier, were employed, achieving 100% accuracy for specific datasets. Additionally, a user-friendly front end was developed using the Flask framework to facilitate user testing and authentication, providing a seamless interface for experimentation. By combining advanced dimensionality reduction techniques with ensemble methods and a user-friendly interface, the proposed work aims to significantly improve the performance and usability

of NIDS, thereby enhancing network security and efficiency.

b) System Architecture:

The system architecture comprises two main modules:

This module consists of the original dataset and its reduced versions obtained through autoencoder-based dimensionality reduction. These datasets are fed into the classifiers for training and testing.

Four classifiers, namely Extra Tree Classifier, Artificial Neural Network (ANN), Extension Voting Classifier, and Extension Stacking Classifier, are employed to analyze the datasets and predict intrusion instances. The results obtained from these classifiers are then evaluated using performance metrics such as accuracy, precision, recall, and F1-score to assess the effectiveness of the models in detecting network intrusions.

These modules interact seamlessly to process data, train classifiers, predict intrusion instances, and evaluate classifier performance, thereby providing insights into the efficacy of different classifiers and dataset versions in detecting network intrusions.

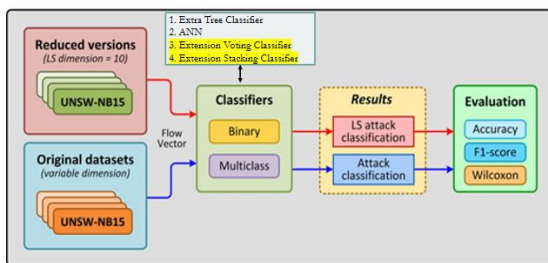


Fig 1 proposed Architecture

c) Dataset collection:

Data collection for the study involved the selection of five widely used datasets for network intrusion detection systems (NIDS). These datasets were chosen to ensure a comprehensive analysis and encompass a variety of attack types. The datasets considered include CIC-IDS2017, UNSW-NB15, NF-UNSW-NB15-v2, CSE-CIC-IDS2018, and NSL-KDD.

A	B	C	D	E	F	G	H	I	J	K	L
Flow Pkts Len Std	Flow Pkts Len Mean	Flow Pkts Len Max	Flow Pkts Len Avg	Flow Pkts Len Std	Flow Pkts Len Std	Flow Pkts Len Std	Flow Pkts Len Std	Flow Pkts Len Std	Flow Pkts Len Std	Flow Pkts Len Std	Flow Pkts Len Std
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	0
4	2	0	0	0	0	0	0	0	0	0	0
5	3	106.7442369	82.6	744	82.6	277.83476	247443.779	371.6778022	227.3	140.48	1239
6	4	203.7483454	81.64280734	744	81.64280734	279.7630336	511536.6097	362.2498035	200.8183818	134.08	1143
7	5	186.8111336	77.41376	744	77.41376	265.7086676	292124.8479	347.6426394	188.4166667	125.638314	1338
8	6	0	0	0	0	0	0	0	0	0	0
9	7	0	0	0	0	0	0	0	0	0	0
10	8	94.36206805	42.2	211	42.2	151.4058893	115905.927	267.3131746	154.3333333	84.25	211
11	9	98.38099101	44	220	44	165.0669898	115746.8132	272.5093271	157.3333333	86.5	220
12	10	98.38099101	44	220	44	165.0669898	115746.8132	272.5093271	157.3333333	86.5	220
13	11	93.46704146	41.8	209	41.8	160.5842956	116378.2288	266.1584741	153.6666667	83.75	209
14	12	94.36206805	42.2	211	42.2	161.4058893	114939.4737	267.3131746	154.3333333	84.25	211
15	13	91.12000067	41.2	206	41.2	159.1787061	115977.2151	264.6746313	152.6666667	83	206
16	14	94.36206805	42.2	211	42.2	161.4058893	115533.0733	267.3131746	154.3333333	84.25	211
17	15	94.36206805	42.2	211	42.2	161.4058893	116318.5305	267.3131746	154.3333333	84.25	211
18	16	95.70370044	42.8	214	42.8	162.6240613	115371.2624	269.0452254	153.3333333	85	214
19	17	93.46704146	41.8	209	41.8	160.5842956	116310.2577	266.1584741	153.6666667	83.75	209
20	18	96.151092303	43	215	43	163.0312683	116521.2605	269.6225757	155.6666667	85.25	215
21	19	96.151092303	43	215	43	163.0312683	115929.0811	269.6225757	155.6666667	85.25	215
22	20	96.151092303	43	215	43	163.0312683	116566.2614	269.6225757	155.6666667	85.25	215
23	21	96.151092303	43	215	43	163.0312683	114942.8953	269.6225757	155.6666667	85.25	215
24	22	96.151092303	43	215	43	163.0312683	116221.1447	269.6225757	155.6666667	85.25	215

Fig 2 CIC IDS 2017

The Canadian Institute for Cybersecurity is highlighted as a primary source for most of these datasets [1]. Additionally, the survey conducted by Hnamte and Hussain [2] provided valuable insights into the diversity of attacks and aided in the selection process. Each dataset is structured as network flows, with traffic flows labeled to differentiate between attacks and benign traffic.

A	B	C	D	E	F	G	H	I	J	K	L
Flow Pkts Len Std	Flow Pkts Len Mean	Flow Pkts Len Max	Flow Pkts Len Avg	Flow Pkts Len Std	Flow Pkts Len Std	Flow Pkts Len Std	Flow Pkts Len Std	Flow Pkts Len Std	Flow Pkts Len Std	Flow Pkts Len Std	Flow Pkts Len Std
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0
3	1	254.8532703	154.8571429	712	154.8571429	320.1228979	5894923.821	395.9299119	240.4166667	198.5	2108
4	2	46.264814	32	64	32	96.9696723	0	0	0	64	64
5	3	0	0	0	0	0	0	0	0	0	0
6	4	0	0	0	0	0	0	0	0	0	0
7	5	0	0	0	0	0	0	0	0	0	0
8	6	103.877188	60.86666667	188	60.86666667	127.6293545	723.512187	401.5	213.75	176.5714286	1842
9	7	0	0	0	0	0	0	0	0	0	0
10	8	0	0	0	0	0	0	0	0	0	0
11	9	177.4499102	75.29411765	712	75.29411765	238.8004351	313947.9	347.6426394	180.4166667	122.5172414	1380
12	10	0	0	0	0	0	0	0	0	0	0
13	11	26.70473473	80	160	80	39.7182811	459304.9177	31.70776644	26.18181818	51.84227642	4060
14	12	0	0	0	0	0	0	0	0	0	0
15	13	0	0	0	0	0	0	0	0	0	0
16	14	62.26826238	27.4	137	27.4	148.6959988	1685536.637	293.1888119	147.3333333	72.375	137
17	15	208.9333334	98.66666667	712	98.66666667	305.3676116	442406.1901	409.9885094	248.125	166.45	1184
18	16	36.50047223	21.33333333	64	21.33333333	32	362.228872	0	0	21.33333333	64
19	17	0	0	0	0	0	0	0	0	0	0
20	18	0	0	0	0	0	0	0	0	0	0
21	19	0	0	0	0	0	0	0	0	0	0
22	20	0	0	0	0	0	0	0	0	0	0
23	21	0	0	0	0	0	0	0	0	0	0
24	22	217.2926609	107.6383636	712	107.6383636	371.9451454	262725.138	381.9811881	372.4545455	240.0454545	1184

Fig 3 CIC IDS 2018

The characteristics of the datasets, such as the number of instances, features, attack categories, and the proportion of attacks. The datasets are available in various formats, including raw pcap

files and CSV files. In this study, CSV files were utilized to maintain consistency and focus on

A	B	C	D	E	F	G	H	I	J	K	L	
l4_src_port	l4_dst_port	protocol	l3_proto	in_bytes	out_bytes	in_pkts	out_pkts	tcp_flags	flow_duration	millisecons	label	
0	62073	5082	6	0	5672	456	11	8	25	15	0	
1	32284	1526	6	0	1776	104	6	2	25	0	0	
2	21	21971	6	1	1842	1236	26	22	25	1111	0	
3	23800	40893	6	0	528	8824	10	12	27	124	0	
4	63062	21	6	1	1786	2340	32	34	25	1459	0	
5	57349	53	17	5	146	178	2	2	0	1	0	
6	45660	80	6	7	690	1272	4	8	25	5	0	
7	29259	25	6	3	37914	3380	54	42	27	22	0	
8	1813	53	17	5	146	178	2	2	0	1	0	
9	20139	80	6	7	690	1272	4	8	25	2	0	
10	54026	53	17	5	146	178	2	2	0	1	0	
11	48622	143	6	4	7818	15800	122	126	27	1139	0	
12	1888	80	6	7	17538	1087890	328	746	27	1139	0	
13	32004	53	17	5	130	162	2	2	0	0	0	
14	49223	5190	6	0	3908	2308	22	24	27	7	0	
15	15	15525	80	6	7	690	1272	4	8	25	2	0
16	49027	22	6	92	3728	5474	32	24	27	6	0	
17	17	6616	53	17	5	146	178	2	2	0	1	0
18	39872	5190	6	0	3920	4312	22	24	27	7	0	
19	19	60500	34044	6	0	8928	320	14	6	27	423	0
20	20850	60046	6	0	424	8824	8	12	27	456	0	
21	14055	26175	17	11	544	304	4	4	0	1	0	
22	22	12607	111	17	11	568	304	4	4	0	4	0

Fig 4 NF-UNSW-NB15

exploring different projections into a lower-dimensional subspace [3].

A	B	C	D	E	F	G	H	I	J	K	L	M	N
service	flag	in_bytes	out_bytes	count	error_rate	name	src_ip	dst_ip	src_port	dst_port	src_rate	dst_rate	rate
0	20	9	495	1	0	25	0.17	0.03	0.07	0	4		
1	42	9	146	0	13	0	0.08	0.15	1	0	0.6	0	4
2	47	5	0	0	133	1	0.05	0.07	0.6	0.05	1	0	0
3	23	9	232	853	5	0.2	1	0	255	1	0	0.03	4
4	23	9	199	420	30	0	1	0	255	1	0	0	4
5	47	1	0	0	101	0	0.18	0.06	19	0.07	0.07	0	0
6	47	5	0	0	186	1	0.05	0.06	9	0.04	0.05	1	0
7	47	5	0	0	117	1	0.14	0.06	15	0.05	0.07	1	0
8	49	5	0	0	230	1	0.08	0.05	23	0.08	0.05	1	0
9	47	5	0	0	133	1	0.06	0.06	13	0.05	0.06	1	0
10	47	1	0	0	205	0	0.06	0.06	12	0.05	0.07	0	0
11	47	5	0	0	199	1	0.02	0.06	13	0.05	0.07	1	0
12	23	9	207	2251	3	0	1	0	255	1	0	0	4
13	20	9	334	3	0	1	0	0	20	1	0	0	2
14	34	5	0	0	233	1	0	0.06	1	0	0.07	1	0
15	36	5	0	0	96	1	0.17	0.05	2	0.01	0.06	1	0
16	23	9	300	1078	8	0	1	0	255	1	0	0	4
17	34	9	18	0	1	0	1	0	35	1	0	0	1
18	23	9	233	608	3	0	1	0	255	1	0	0	4
19	23	9	343	1178	9	0	1	0	255	1	0	0	4
20	33	5	0	0	223	1	0.1	0.05	23	0.08	0.05	1	0
21	47	5	0	0	280	1	0.06	0.05	17	0.07	0.06	0.99	0
22	23	9	251	1895	8	0	1	0	255	1	0	0	4

Fig 5 NSL-KDD

A	B	C	D	E	F	G	H	I	J	K	L	
l4_src_port	l4_dst_port	protocol	l3_proto	in_bytes	out_bytes	in_pkts	out_pkts	tcp_flags	flow_duration	millisecons	label	
0	62073	5082	6	0	5672	456	11	8	25	15	0	
1	32284	1526	6	0	1776	104	6	2	25	0	0	
2	21	21971	6	1	1842	1236	26	22	25	1111	0	
3	23800	40893	6	0	528	8824	10	12	27	124	0	
4	63062	21	6	1	1786	2340	32	34	25	1459	0	
5	57349	53	17	5	146	178	2	2	0	1	0	
6	45660	80	6	7	690	1272	4	8	25	5	0	
7	29259	25	6	3	37914	3380	54	42	27	22	0	
8	1813	53	17	5	146	178	2	2	0	1	0	
9	20139	80	6	7	690	1272	4	8	25	2	0	
10	54026	53	17	5	146	178	2	2	0	1	0	
11	48622	143	6	4	7818	15800	122	126	27	1139	0	
12	1888	80	6	7	17538	1087890	328	746	27	1139	0	
13	32004	53	17	5	130	162	2	2	0	0	0	
14	49223	5190	6	0	3908	2308	22	24	27	7	0	
15	15	15525	80	6	7	690	1272	4	8	25	2	0
16	49027	22	6	92	3728	5474	32	24	27	6	0	
17	17	6616	53	17	5	146	178	2	2	0	1	0
18	39872	5190	6	0	3920	4312	22	24	27	7	0	
19	19	60500	34044	6	0	8928	320	14	6	27	423	0
20	20850	60046	6	0	424	8824	8	12	27	456	0	
21	14055	26175	17	11	544	304	4	4	0	1	0	
22	22	12607	111	17	11	568	304	4	4	0	4	0

Fig 6 UNSW-NB15

d) DATA PROCESSING

In the data processing phase, several steps were undertaken to ensure the integrity and quality of the datasets for network intrusion detection systems (NIDS). Firstly, duplicate data instances were removed to prevent redundancy and streamline the analysis process. This step helps in improving the efficiency of the subsequent analysis by eliminating

unnecessary repetitions. Additionally, drop cleaning techniques were applied to handle missing or erroneous data. This involved identifying and removing or replacing incomplete or inconsistent data entries to maintain dataset integrity. By performing these data processing steps, the datasets were refined and prepared for further analysis, enhancing the reliability and accuracy of the subsequent modeling and evaluation processes. Overall, data processing plays a crucial role in ensuring the quality and reliability of the datasets, thereby facilitating meaningful insights and conclusions in the context of network intrusion detection systems.

e) VISUALIZATION

Visualization of data using Seaborn and Matplotlib is a fundamental aspect of exploratory data analysis (EDA) in various domains, including network intrusion detection systems (NIDS). Seaborn and Matplotlib are powerful Python libraries that offer a wide range of plotting functions and customization options to visualize data effectively. Seaborn provides a high-level interface for creating attractive and informative statistical graphics, while Matplotlib offers fine-grained control over plot elements and layouts. By leveraging these libraries, analysts can generate insightful visualizations such as histograms, scatter plots, box plots, and heatmaps to explore the distribution, relationships, and patterns within the datasets. These visualizations aid in gaining a deeper understanding of the data, identifying outliers, detecting trends, and informing subsequent modeling decisions. Overall, the combination of Seaborn and Matplotlib enhances the interpretability and communicability of data analysis results, facilitating informed

decision-making in NIDS research and development.

f) LABEL ENCODING

Label Encoding is a preprocessing technique used to convert categorical data represented as strings into numerical values, making it suitable for machine learning algorithms. In Label Encoding, each unique category or label in the categorical variable is assigned a unique integer identifier. This transformation enables algorithms to interpret categorical data as numerical features, facilitating model training and prediction. However, it is important to note that Label Encoding should be applied to ordinal categorical variables, where the order of categories holds significance. One common implementation of Label Encoding in Python is using the `LabelEncoder` class from the `scikit-learn` library. While Label Encoding is a simple and effective method for handling categorical data, it may introduce unintended ordinal relationships between categories, which could adversely affect the performance of some algorithms. Therefore, it is essential to use Label Encoding judiciously and consider alternative encoding techniques for nominal categorical variables.

g) FEATURE SELECTION

Feature selection is a crucial step in machine learning, especially for building effective predictive models in network intrusion detection systems (NIDS). It involves selecting a subset of relevant features from the dataset that contribute most to the predictive performance of the model while discarding irrelevant or redundant features. This process helps in improving model accuracy, reducing overfitting, and enhancing computational

efficiency by focusing on the most informative attributes. Various techniques can be employed for feature selection, including filter methods, wrapper methods, and embedded methods. Filter methods assess the relevance of features independently of the learning algorithm, wrapper methods use the learning algorithm to evaluate feature subsets, and embedded methods incorporate feature selection into the model training process. By carefully selecting informative features, feature selection optimizes model performance and aids in extracting meaningful insights from the data in NIDS applications.

h) TRAINING AND TESTING

Training and testing are essential components of the machine learning workflow, including in the context of network intrusion detection systems (NIDS). Training involves using a portion of the available data to teach a model to make predictions or classifications based on input features. During training, the model learns patterns and relationships in the data, adjusting its parameters to minimize prediction errors. Once trained, the model is evaluated using a separate portion of the data called the test set. The test set serves as an independent dataset to assess the generalization performance of the model. By evaluating the model on unseen data, testing helps to estimate how well the model will perform on new, unseen data in real-world scenarios. This process enables practitioners to gauge the model's effectiveness, identify potential issues such as overfitting, and make informed decisions about model deployment in NIDS applications.

i) ALGORITHMS:

1. Extra Tree Classifier:

The Extra Trees Classifier is an ensemble machine learning algorithm that belongs to the family of decision tree classifiers. It builds a forest of decision trees and combines their outputs to make predictions. Extra Trees Classifiers are used to enhance the accuracy and robustness of intrusion detection systems. By utilizing an ensemble approach, this algorithm helps improve the detection of network intrusions by aggregating the results from multiple decision trees. It's particularly valuable in situations where a single decision tree might overfit to the data or miss patterns present in the network traffic.

```
from sklearn.ensemble import ExtraTreesClassifier

# instantiate the model
et = ExtraTreesClassifier(n_estimators=100, random_state=0)

et.fit(X_train, y_train)

y_pred = et.predict(X_test)

et_acc = accuracy_score(y_pred, y_test)
et_prec = precision_score(y_pred, y_test, average='weighted')
et_rec = recall_score(y_pred, y_test, average='weighted')
et_f1 = f1_score(y_pred, y_test, average='weighted')
```

Fig 7 . Extra Tree Classifier:

2. Artificial Neural Network (ANN):

Artificial Neural Networks are a class of machine learning models inspired by the human brain's neural structure. They consist of interconnected nodes (neurons) organized in layers, including input, hidden, and output layers. ANNs are capable of learning complex, non-linear relationships in data. And we have autoencoders in the hidden layers ANNs are employed for feature learning and network traffic pattern recognition. They can

extract intricate features from network data and adapt to changing attack patterns. In the context of latent space dimensionality reduction, ANNs can help capture critical patterns while reducing data dimensionality, making them valuable for intrusion detection in a standardized latent space.

```
def nn():
    inputs = Input(name='inputs', shape=[X_train.shape[1],])
    layer = Dense(128, name='FC1')(inputs)
    layer = BatchNormalization(name='BC1')(layer)
    layer = Activation('relu', name='Activation1')(layer)
    layer = Dropout(0.3, name='Dropout1')(layer)
    layer = Dense(128, name='FC2')(layer)
    layer = BatchNormalization(name='BC2')(layer)
    layer = Activation('relu', name='Activation2')(layer)
    layer = Dropout(0.3, name='Dropout2')(layer)
    layer = Dense(128, name='FC3')(layer)
    layer = BatchNormalization(name='BC3')(layer)
    layer = Dropout(0.3, name='Dropout3')(layer)
    layer = Dense(1, name='OutLayer')(layer)
    layer = Activation('sigmoid', name='sigmoid')(layer)
    model = Model(inputs=inputs, outputs=layer)
    return model
```

Fig 8 Artificial Neural Network

3. Voting Classifier:

The Voting Classifier is an ensemble technique that combines the predictions from multiple machine learning algorithms to make a final decision. It can be "hard" (based on majority voting) or "soft" (weighted voting). Voting Classifiers are useful for increasing the reliability and accuracy of intrusion detection systems. By integrating the outputs of multiple algorithms, they reduce the risk of false positives and improve overall system performance. In the project, they help in evaluating the effectiveness of different algorithms on the standardized latent space.


```

from sklearn.ensemble import RandomForestClassifier, VotingClassifier, AdaBoostClassifier
clf1 = AdaBoostClassifier(n_estimators=100, random_state=0)
clf2 = RandomForestClassifier(n_estimators=50, random_state=1)

ecclf1 = VotingClassifier(estimators=[('ad', clf1), ('rf', clf2)], voting='soft')
ecclf1.fit(X_train, y_train)
y_pred = ecclf1.predict(X_test)

vot_acc_b = accuracy_score(y_pred, y_test)
vot_prec_b = precision_score(y_pred, y_test, average='weighted')
vot_rec_b = recall_score(y_pred, y_test, average='weighted')
vot_f1_b = f1_score(y_pred, y_test, average='weighted')

```

Fig 9 Voting Classifier

```

from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from lightgbm import LGBMClassifier
from sklearn.ensemble import StackingClassifier

estimators = [('rf', RandomForestClassifier(n_estimators=1000)), ('mlp', MLPClassifier(random_state=1, n_iter_no_change=100)), ('lgbm', LGBMClassifier(n_estimators=1000))]
clf1 = StackingClassifier(estimators=estimators, final_estimator=LGBMClassifier(n_estimators=1000))

clf1.fit(X_train, y_train)

y_pred = clf1.predict(X_test)

stac_acc_b = accuracy_score(y_pred, y_test)
stac_prec_b = precision_score(y_pred, y_test, average='weighted')
stac_rec_b = recall_score(y_pred, y_test, average='weighted')
stac_f1_b = f1_score(y_pred, y_test, average='weighted')

```

Fig 10 Stacking Classifier

4. Stacking Classifier:

Stacking is an ensemble technique that involves training multiple machine learning models and using another model (meta-learner) to combine their predictions.

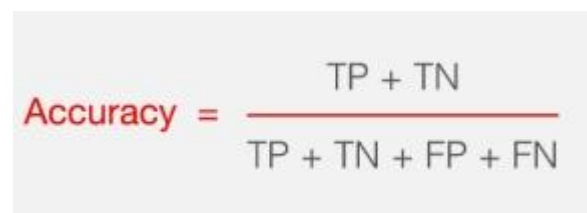
Stacking Classifiers are beneficial for improving the overall performance and robustness of intrusion detection systems. They allow the combination of multiple base models to enhance accuracy and adaptability. In the context of the project, stacking classifiers can provide a meta-level evaluation of different algorithms in the standardized latent space, offering insights into their combined effectiveness.

This performance table pertains to the Drebin dataset. In a similar manner, we have created performance metrics tables for models that were trained using the include CIC IDS 2018, UNSW-NB15, NF-UNSW-NB15-V2, CSE-CIC-IDS2018, and NSL-KDD datasets.

4. EXPERIMENTAL RESULTS

Accuracy: The accuracy of a test is its ability to differentiate the patient and healthy cases correctly. To estimate the accuracy of a test, we should calculate the proportion of true positive and true negative in all evaluated cases. Mathematically, this can be stated as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$



The diagram shows the accuracy formula: Accuracy = (TP + TN) / (TP + TN + FP + FN). The numerator is TP + TN and the denominator is TP + TN + FP + FN. A horizontal line separates the numerator from the denominator.

Precision: Precision evaluates the fraction of correctly classified instances or samples among the ones classified as positives. Thus, the formula to calculate the precision is given by:

$$\text{Precision} = \frac{\text{True positives}}{\text{True positives} + \text{False positives}} = \frac{TP}{TP + FP}$$

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

Recall: Recall is a metric in machine learning that measures the ability of a model to identify all relevant instances of a particular class. It is the ratio of correctly predicted positive observations to the total actual positives, providing insights into a model's completeness in capturing instances of a given class.

$$Recall = \frac{TP}{TP + FN}$$

F1-Score: F1 score is a machine learning evaluation metric that measures a model's accuracy. It combines the precision and recall scores of a model. The accuracy metric computes how many times a model made a correct prediction across the entire dataset.

$$F1\ Score = \frac{2}{\left(\frac{1}{Precision} + \frac{1}{Recall}\right)}$$

$$F1\ Score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

ML Model	Accuracy	F1-score	Recall	Precision
ANN – Multi Class	0.636	0.778	0.636	1.000
ET – Multi Class	0.782	0.763	0.782	0.852
Extension Stacking Classifier – Multi Class	0.782	0.763	0.782	0.852
Extension Voting Classifier – Multi Class	0.782	0.763	0.782	0.852
ANN – Binary Class	1.000	1.000	1.000	1.000
ET – Binary Class	1.000	1.000	1.000	1.000
Extension Stacking Classifier – Binary Class	1.000	1.000	1.000	1.000
Extension Voting Classifier – Binary Class	1.000	1.000	1.000	1.000

Fig 11 CIC IDS 2017 - Performance Evaluation table

ML Model	Accuracy	F1-score	Recall	Precision
ANN – Multi Class	0.881	0.857	0.881	0.862
ET – Multi Class	0.987	0.988	0.987	0.991
Extension Stacking Classifier – Multi Class	0.987	0.988	0.987	0.991
Extension Voting Classifier – Multi Class	0.987	0.988	0.987	0.991
ANN – Binary Class	1.000	1.000	1.000	1.000
ET – Binary Class	1.000	1.000	1.000	1.000
Extension Stacking Classifier – Binary Class	1.000	1.000	1.000	1.000
Extension Voting Classifier – Binary Class	1.000	1.000	1.000	1.000

Fig12 CIC IDS 2018 - Performance Evaluation table

ML Model	Accuracy	F1-score	Recall	Precision
ANN – Multi Class	0.003	0.006	0.003	1.000
ET – Multi Class	0.960	0.958	0.960	0.957
Extension Stacking Classifier – Multi Class	1.000	0.909	0.922	0.931
Extension Voting Classifier – Multi Class	1.000	0.960	0.959	0.961
ANN – Binary Class	0.931	0.960	0.931	0.995
ET – Binary Class	0.988	0.988	0.988	0.988
Extension Stacking Classifier – Binary Class	1.000	0.986	0.986	0.986
Extension Voting Classifier – Binary Class	0.989	0.989	0.989	0.989

Fig13 NF-UNSW-NB15 - Performance Evaluation table

ML Model	Accuracy	F1-score	Recall	Precision
ANN – Multi Class	0.092	0.168	0.092	1.000
ET – Multi Class	0.997	0.997	0.997	0.997
Extension Stacking Classifier – Multi Class	1.000	0.998	0.998	0.998
Extension Voting Classifier – Multi Class	0.996	0.997	0.996	0.997
ANN – Binary Class	0.533	0.695	0.533	1.000
ET – Binary Class	0.997	0.997	0.997	0.997
Extension Stacking Classifier – Binary Class	1.000	0.998	0.998	0.998
Extension Voting Classifier – Binary Class	0.998	0.999	0.998	0.999

Fig 14 NSL-KDD - Performance Evaluation table

ML Model	Accuracy	F1-score	Recall	Precision
ANN – Multi Class	0.008	0.016	0.008	0.255
ET – Multi Class	0.830	0.836	0.830	0.843
Extension Stacking Classifier – Multi Class	0.381	0.445	0.381	0.938
Extension Voting Classifier – Multi Class	0.834	0.841	0.834	0.849
ANN – Binary Class	0.753	0.754	0.753	0.776
ET – Binary Class	0.931	0.931	0.931	0.932
Extension Stacking Classifier – Binary Class	0.933	0.933	0.933	0.933
Extension Voting Classifier – Binary Class	0.937	0.937	0.937	0.937

Fig 15 UNSW-NB15 - Performance Evaluation table

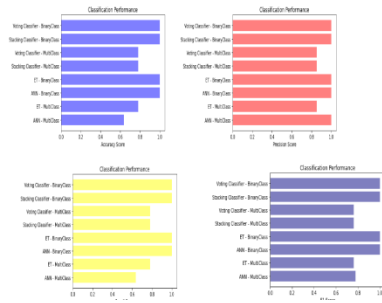


Fig 16 CIC IDS 2017 COMPARISON GRAPH

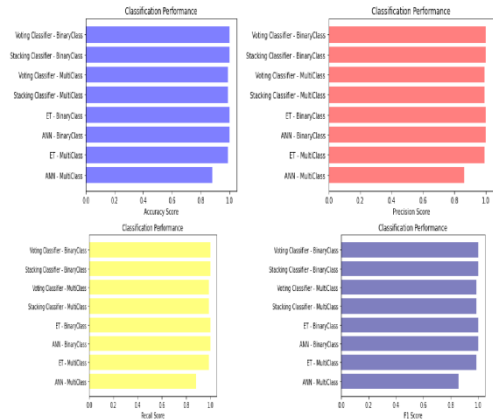


Fig 17 CIC IDS 2018 COMPARISON GRAPH

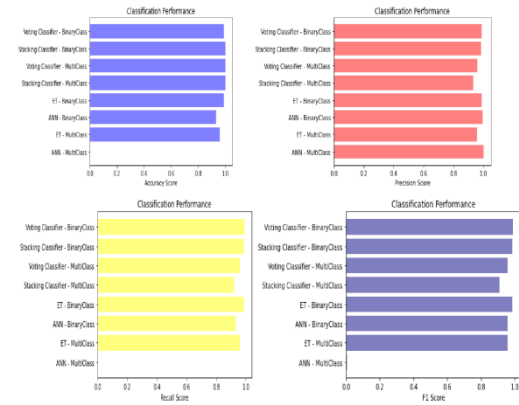


Fig 18 NF-UNSW-NB15 COMPARISON GRAPH

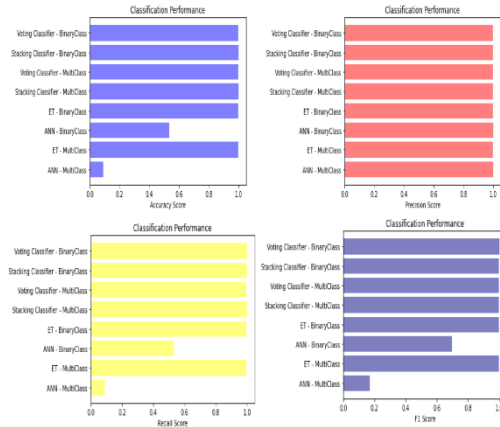


Fig 19 NSL-KDD COMPARISON GRAPH

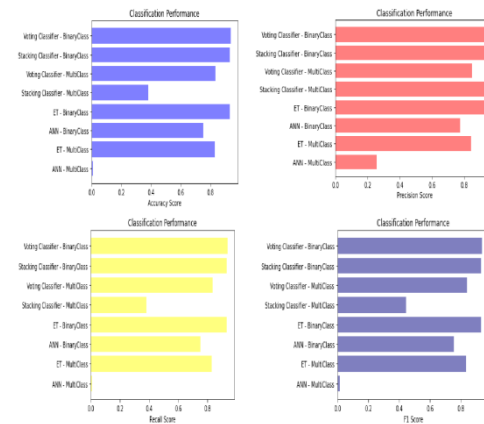


Fig 20 UNSW-NB15 COMPARISON GRAPH



Fig 21 Home page

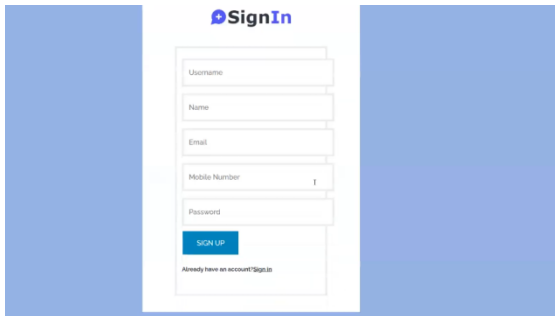


Fig 22 sign up



Fig23 sign in



Fig 24NSL KDD



Fig 25 upload input data

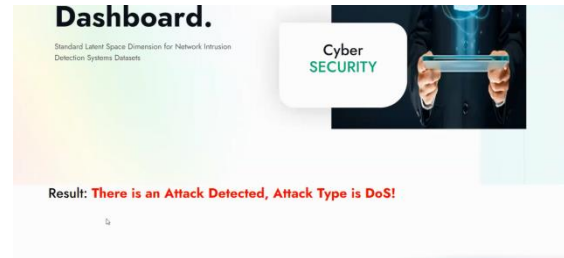


Fig 26 predicted Result

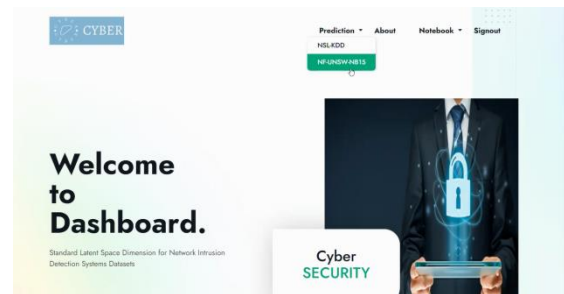


Fig 27 Nf-Unsw-Nb15

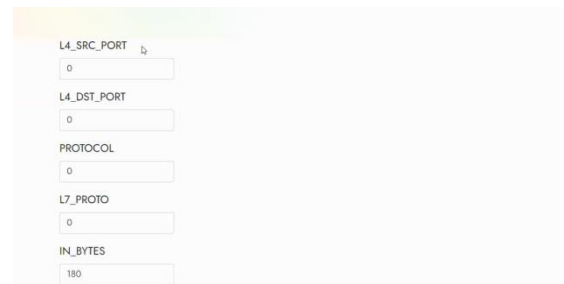


Fig 28 upload input data

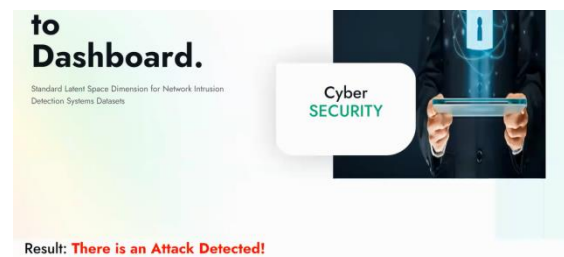


Fig 29 Predict result



Fig 30 upload input data



Fig31 predicted Result

5. CONCLUSION

The project successfully implemented a standardized latent space dimension using autoencoder-based dimensionality reduction, significantly enhancing Network Intrusion Detection Systems (NIDS). This approach addressed traditional NIDS limitations, such as adaptability to evolving threats and reliance on manual feature engineering. Machine learning models trained with the standardized latent space showed comparable performance to non-reduced datasets, validating the approach's viability. The extension algorithm demonstrated enhanced prediction accuracy through ensemble methods and adaptability to diverse intrusion scenarios. Integration of the Flask web framework with user authentication improved user engagement, enabling real-time intrusion event prediction and enhancing system usability and effectiveness.

6. FUTURE SCOPE

Further processing of latent spaces can determine a unified standard, distributable to the research community. Advanced techniques like stacked autoencoders can extract features to analyze latent-space convergence with complex datasets. Analyzing single-attack class datasets, like Distributed Denial-of-Service, can reveal latent-space dimension variations. Applying the standard latent space to other NIDS datasets evaluates its effectiveness. Comparing machine learning models using the standard space identifies optimal NIDS models. Collaborative efforts can expand the standard space by incorporating diverse datasets, enhancing NIDS model accuracy and reliability.

REFERENCES

- [1] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, "Machine learning for networking: Workflow, advances and opportunities," *IEEE Netw.*, vol. 32, no. 2, pp. 92–99, Mar. 2018.
- [2] S. V. Mahadevkar, B. Khemani, S. Patil, K. Kotecha, D. R. Vora, A. Abraham, and L. A. Gabralla, "A review on machine learning styles in computer vision—Techniques and future directions," *IEEE Access*, vol. 10, pp. 107293–107329, 2022.
- [3] R. M. Samant, M. R. Bachute, S. Gite, and K. Kotecha, "Framework for deep learning-based language models using multi-task learning in natural language understanding: A systematic literature review and future directions," *IEEE Access*, vol. 10, pp. 17078–17097, 2022.

- [4] O. Barut, Y. Luo, T. Zhang, W. Li, and P. Li, "NetML: A challenge for network traffic analytics," 2020, arXiv:2004.13006.
- [5] M. Sarhan, S. Layeghy, and M. Portmann, "Towards a standard feature set for network intrusion detection system datasets," *Mobile Netw. Appl.*, vol. 27, pp. 357–370, Nov. 2021.
- [6] J. Holland, P. Schmitt, N. Feamster, and P. Mittal, "New directions in automated traffic analysis," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 3366–3383.
- [7] M. Li, H. Wang, L. Yang, Y. Liang, Z. Shang, and H. Wan, "Fast hybrid dimensionality reduction method for classification based on feature selection and grouped feature extraction," *Exp. Syst. Appl.*, vol. 150, Jul. 2020, Art. no. 113277.
- [8] L. Van Der Maaten, E. Postma, and J. Van den Herik, "Dimensionality reduction: A comparative," *J. Mach. Learn. Res.*, vol. 10, nos. 66–71, p. 13, 2009.
- [9] Q. Fournier and D. Aloise, "Empirical comparison between autoencoders and traditional dimensionality reduction methods," in *Proc. IEEE 2nd Int. Conf. Artif. Intell. Knowl. Eng. (AIKE)*, Jun. 2019, pp. 211–214.
- [10] B. Janakiramaiah, G. Kalyani, S. Narayana, and T. B. M. Krishna, "Reducing dimensionality of data using autoencoders," in *Smart Intelligent Computing and Applications*. Berlin, Germany: Springer, 2020, pp. 51–58.
- [11] Y. Wang, H. Yao, and S. Zhao, "Auto-encoder based dimensionality reduction," *Neurocomputing*, vol. 184, pp. 232–242, Apr. 2016.
- [12] D. C. Ferreira, F. I. Vázquez, and T. Zseby, "Extreme dimensionality reduction for network attack visualization with autoencoders," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–10.
- [13] H. Motoda and H. Liu, "Feature selection, extraction and construction," *Commun. IICM*, vol. 5, p. 2, May 2002.
- [14] R. Zebari, A. Abdulazeez, D. Zeebaree, D. Zebari, and J. Saeed, "A comprehensive review of dimensionality reduction techniques for feature selection and feature extraction," *J. Appl. Sci. Technol. Trends*, vol. 1, no. 2, pp. 56–70, May 2020.
- [15] M. Sarhan, S. Layeghy, and M. Portmann, "Evaluating standard feature sets towards increased generalisability and explainability of ML-based network intrusion detection," 2021, arXiv:2104.07183.
- [16] F. Bronzino, P. Schmitt, S. Ayoubi, H. Kim, R. Teixeira, and N. Feamster, "Traffic refinery: Cost-aware data representation for machine learning on network traffic," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 5, no. 3, pp. 1–24, 2021.
- [17] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th Int. Conf. Inf. Syst. Secur. Privacy*, vol. 1, Jan. 2018, pp. 108–116.
- [18] A. Gharib, I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "A evaluation framework for intrusion detection dataset," in *Proc. Int. Conf. Inf. Sci. Secur. (ICISS)*, Dec. 2016, pp. 1–6.

- [19] V. Hnamte and J. Hussain, "An extensive survey on intrusion detection systems: Datasets and challenges for modern scenario," in Proc. 3rd Int. Conf. Electr., Control Instrum. Eng. (ICECIE), Nov. 2021, pp. 1–10.
- [20] S.B.Kotsiantis, D. Kanellopoulos, and P. E. Pintelas, "Data preprocessing for supervised learning," *Int. J. Comput. Sci.*, vol. 1, no. 2, pp. 111–117, 2006.
- [21] S.Bhattacharya, P. K. R. Maddikunta, R.Kaluri, S. Singh, T. R. Gadekallu, M. Alazab, and U. Tariq, "A novel PCA-firefly based XGBoostclassification model for intrusion detection in networks using GPU," *Electronics*, vol. 9, no. 2, p. 219, Jan. 2020.
- [22] J. Carrasco, S. García, M. M. Rueda, S. Das, and F. Herrera, "Recent trends in the use of statistical tests for comparing swarm and evolutionary computing algorithms: Practical guidelines and a critical review," *Swarm Evol. Comput.*, vol. 54, May 2020, Art. no. 100665.
- [23] J. Stewart, *Calculus*. Boston, MA, USA: Cengage Learning, 2015.
- [24] K. Banachewicz and L. Massaron, *TheKaggle Book: Data Analysis and Machine Learning for Competitive Data Science*. Birmingham, U.K.: Packt, 2022. [Online]. Available: <https://cir.nii.ac.jp/crid/1130854870125643045>
- [25] V. K. Ayyadevara, *Neural Networks With Keras Cookbook: Over 70 Recipes Leveraging Deep Learning Techniques Across Image, Text, Audio, and Game Bots*. Birmingham, U.K.: Packt, 2019.
- [26] Q.-S. Xu and Y.-Z.Liang, "Monte Carlo cross validation," *ChemometricIntell. Lab. Syst.*, vol. 56, no. 1, pp. 1–11, Apr. 2001.
- [27] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Jan. 2006.