Data Visualization and Preprocessing for Network Traffic Analysis in Cybersecurity

Mayank Kapadia[1], Andrew Dunton[1], Chelsea Jaculina[1], David Thach[1], Albert Ong[1] and Shih Yu Chang[1]
[1]San Jose State University (SJSU), San Jose, California, USA

In the age of cybersecurity, visualization and interpretation of network traffic data is very crucial for real-time intrusion detection. The proposed paper provides a data visualization driven approach for analyzing network intrusions using the CICIDS 2017 dataset. The study also uses different preprocessing techniques, such as data cleaning, transformation, and feature selection, to make the data set ready for analysis. Principal Component Analysis, or PCA, is used for reducing dimensionality, helping to optimize memory, and clarifying visualizations. We will focus on the visualization of network attack patterns, model performance, and PCA results that provides actionable insights. Python libraries are used in conjunction with Power BI to create a data visualization platform that has interactive real-time visualizations for users to explore across attack types, feature importance, and model evaluation metrics. The goal is to show how effective data visualization can improve the understanding of complex network traffic data and help make better decisions in cybersecurity.

Index Terms—**Cybersecurity, Intrusion Detection Systems (IDS), Network Traffic Analysis, Data Visualization, Machine Learning, Primary Component Analysis (PCA), CICIDS 2017 Dataset, Dimensionality Reduction, Power BI, Real-time Visualization, Anomaly Detection**

I. Introduction

CYBERSECURITY is one of the biggest challenges facing the digital world today. As the frequency and complexity of cyber attacks continue to rise, organizations are more reliant on Intrusion Detection Systems to identify malicious network activities and secure their infrastructures. However, the high volume and complexity of network traffic data pose quite a few challenges to the effective detection and understanding of such attacks. Traditional network traffic analysis methods, based on raw statistics and numerical data sets, usually become insufficient when dealing with complexities inherent in high-dimensional data. In addition, although machine learning techniques can efficiently classify network intrusions, model interpretability and real-time decision-making capabilities are still limited. This research proposes one such methodology by combining strategies in data visualization with frameworks in machine learning. The CICIDS 2017(Canadian Institute for Cybersecurity Intrusion Detection System) dataset was chosen because it is replete with network traffic information. This study, therefore, uses Principal Component Analysis(PCA) for the reduction of dimensionality so that real-time visualization of network traffic can be achieved over platforms like Power BI for comprehension, model evaluation, and proactive security measures in detecting complex attack patterns.

A. Motivation

As network traffic is increasing in terms of volume and sophistication with dynamic and evolving cyber attacks, traditional intrusion detection methods become more ineffective. Standard Network Intrusion Detection Systems (NIDS) generate enormous amounts of data, which can be hard to inspect and comprehend. Also, though machine learning algorithms have improved intrusion detection significantly, the nature of such models is to be very opaque, making it challenging for security professionals to identify which features are influencing predictions.

1) Limitations of existing technologies:

- **Scalability Issues** – As network traffic grows, traditional NIDS cannot process high-dimensional data, leading to performance bottlenecks.
- **Lack of Interpretability** – The majority of machine learning models are black boxes, giving accurate predictions but no insight into the decision-making process.
- **Computational Costs** – Analysing high-dimensional data is computationally expensive, which makes real-time intrusion detection difficult.

2) Overcoming Such Challenges:

This paper is driven by the need to improve interpretability in cybersecurity via the application of data visualization techniques. Visualization provides an intuitive platform for exploring complex network data, revealing hidden patterns of attacks, feature significance, and interdependencies not readily apparent within raw datasets. Furthermore,dimensionality reduction using Principal Component Analysis (PCA)[20] makes processing more efficient without compromising useful information.

3) Research Motivation and Goals:

The primary goal is to provide cybersecurity professionals with actual and practical tools to improve their efficiency in identifying, understanding and responding to network attacks. By bringing together machine learning and visualization, this study will bridge the model accuracy interpretability gap to offer more understandable and actionable results, especially for real-time intrusion detection systems where prompt decision-making is crucial.

B. Contribution

This article contributes to the field of research in network intrusion detection and data visualization by addressing funda-

Manuscript received January 6, 2025; revised May 14, 2025. Corresponding author: Mayank Kapadia (email: mayankkapadia12@gmail.com).

mental limitations of existing technologies: scalability issues, lack of interpretability, and computationally costly.

The major contributions of this paper are as follows:

1) Enhancing Interpretability with Data Visualization:

We devise an innovative solution with the integration of machine learning and data visualization to facilitate better interpretability for network intrusion detection. This solution enables cybersecurity professionals to graphically explore complex intrusion patterns, feature significance, and attack trends, addressing the issue of machine learning models that cannot be interpreted (Lack of Interpretability).

2) Successful Dimensionality Reduction with PCA:

The paper applies Principal Component Analysis (PCA)citehowto:olive2017principal to the CICIDS 2017 datasetcitehowto:sharafaldin2018, reducing 85 features to a more manageable number without sacrificing principal variance. This enhances computational efficiency, making real-time intrusion detection possible without compromising model accuracy, thus mitigating high computational cost.

3) Real-Time Interactive Visualizations using Power BI:

We also extend the use of Power BI[21] for real-time visualization of network traffic, enabling an interactive simulation of network traffic, attack detection, and model predictions. This addresses the scalability issue by facilitating large amounts of intrusion detection data to be efficiently monitored in real time.

4) Actionable Intelligence for Cybersecurity Decision-Making:

The research showcases interactive dashboards that provide insights into:Feature importance,Attack detection trends and Model performance.These insights allow cybersecurity practitioners to make decisions faster and more confidently, overcoming the poor transparency of traditional machine learning models.

5) Real-Time Attack Simulation for Proactive Defense:

We simulate real-time attack scenarios in Power BI to demonstrate the effectiveness of machine learning-based intrusion detection. This provides a practical roadmap for dynamic threat monitoring and response, which is necessary for real-time cybersecurity applications.

C. Organization

The rest of the paper is organized as follows:

Section 2: Related Work

This section provides an overview of previous research, including studies on network intrusion detection, the application of machine learning for cybersecurity, and the role of dimensionality reduction techniques, like PCA[20], in improving model performance.

Section 3: Proposed Methodology

This section presents in detail the sequence of methodology adopted for this work, starting from preprocessing, cleaning, and transforming the data using the CICIDS 2017[1] Dataset. Further, the section describes the reduction of dimensionality by PCA[20] and feature selection techniques. Later, the section debates machine learning models used in intrusion detection, thereby analyzing accuracy, feature importance, and classification results. It also includes a table comparing the best performing model resulting from this study with other studies in the same domain.

Section 4: Data Visualization

This section shows the visual outputs of the study in terms of how data insights and model results were represented. Discussion will cover visualizations created through Python and Power BI[21], real-time simulations, and interactive dashboards.

Section 5: Future Work

This section describes some possible future work: advanced machine learning models could be used, other dimensionality reduction techniques could be applied, the scalability for larger datasets, and the development of better real-time data visualization frameworks.

Section 6: Paper Highlights

It provides an overview of what was contributed and found in this study, focused on the integration of machine learning into data visualization as part of improving the effectiveness of network intrusion detection.

Section 7: Conclusion

The paper concludes by summarizing the major insights gained from the research. It highlights how combining dimensionality reduction, machine learning, and data visualization techniques can have a greater impact on enhancing the interpretability and efficiency of cybersecurity systems.

II. Related Work

IDS(Intrusion Detection System) and network traffic analysis have gained much attention in the cybersecurity domain due to their capability for the detection of malicious activities that may affect the security of networks. Different works present the use of preprocessing techniques, visualization tools, and machine learning models in intrusion detection.

Preprocessing techniques play a very important role in the dimensionality reduction of data, improvement of computational efficiency, and enhancing the performance of IDS models. Sharafaldin et al. (2018)[1] proposed the CICIDS 2017[1] dataset, which has become the benchmark for intrusion detection research by providing a complete characterization of network traffic. Wang et al. (2020)[2] performed PCA[20] on this dataset, which reduced the feature space while maintaining the accuracy of detection. Similarly, Moustafa and Slay (2015)[3] performed feature engineering on the UNSW-NB15 dataset by addressing challenges in class imbalance and noisy data. Feature selection techniques such as information gain have also been shown to further enhance anomaly detection accuracy by Kurniabudi et al. (2020)[4].

Various visualization techniques have also been significantly explored to uncover complex patterns and anomalies in network traffic. Chen et al. (2021)[5] applied Tableau to visualize network flows with the aim of detecting attack patterns, although limitations on further drill down capabilities into the detected anomalies were discussed. Ullah et al. (2021)[6] extended ML models with Power BI[21] dashboards and considerably improved interpretability of insights for a

non-specialist user. These point out the necessity of having an interactive and dynamic visualization framework in place while analyzing network traffic.

Machine learning (ML) and deep learning(DL) models have automated intrusion detection by identifying malicious behavior. Sharafaldin et al. (2018)[1] investigated some classic ML methods, including Random Forest[12] and SVM[16] on the CICIDS 2017 dataset, which reported high detection rates. Mondal and Sanchez (2021)[7] proposed a Docker environment based on supervised ML for real-time detection, with scalability still an issue.

Standardization ensures devices work appropriately with the others. Deep learning approaches, such as LSTM networks for anomaly detection in sequential traffic data (Ullah et al., 2021)[6], achieved improved detection rates but lacked interpretability. To address these challenges, ensemble learning methods (Zhou et al., 2020)[8] and evolutionary deep learning frameworks (Elmasry et al., 2020)[9] have been introduced.

Benchmark datasets like CICIDS 2017[1] and UNSW-NB15 have played an important role in the advancement of IDS research. These datasets encompass a wide variety of attack types, offering robust testbeds for evaluating IDS models. Sharafaldin et al. (2018)[1] mentioned that the high dimensionality of CICIDS 2017 presents a challenge for real-time processing. Feature selection techniques and transformations, such as logarithmic transformations, have been effective in mitigating these challenges, as shown by Vinayakumar et al. (2019)[10] and Gamage & Samarabandu (2020)[23]. Furthermore, Yulianto et al. (2019)[24] showed the utility of enhancing AdaBoost-based IDS performance on the CICIDS 2017[1] dataset.

Coupled with these, the development of preprocessing, feature visualization, and machine learning techniques, as well as benchmark datasets, greatly enhanced IDS capabilities. Nevertheless, research in this domain has been continuously driven by challenges such as scalability, interpretability, and real-time processing.

III. Proposed Methodology

Here, we discuss how the CICIDS 2017[1] dataset has been preprocessed and what network attack type analysis technique has been used. The main motivation behind this approach is to utilize PCA[20] with minimum loss of information such that the reduced representation might retain the strength of accuracy in attack prediction. We try to achieve quicker training times and lower memory usage using PCA[20] without a significant loss of model accuracy. Besides, real-time simulation and visualization are done with Power BI[21] in order to monitor and assess the performance of the model in dynamic conditions. Further, in the subsections, the pre-processing of data, application of PCA[20], and development of real-time visualizations for model evaluation are explained.

A. Dataset

1) *Dataset Overview*

The CICIDS 2017[1] dataset, released by the Canadian Institute for Cybersecurity, includes tagged examples of various network attacks. It is a popular dataset for network intrusion detection research, containing 85 features such as flow statistics and network traffic patterns. These features collect critical network traffic information such as source and destination addresses, packet counts, and flow time, among others.

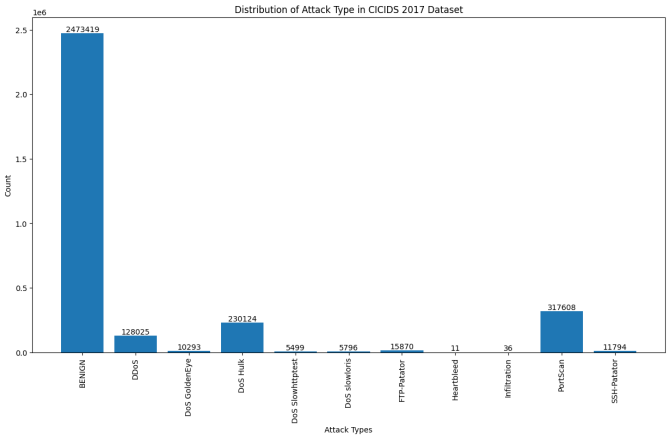

Fig. 1. Distribution of Attack Type in CICIDS 2017 Dataset

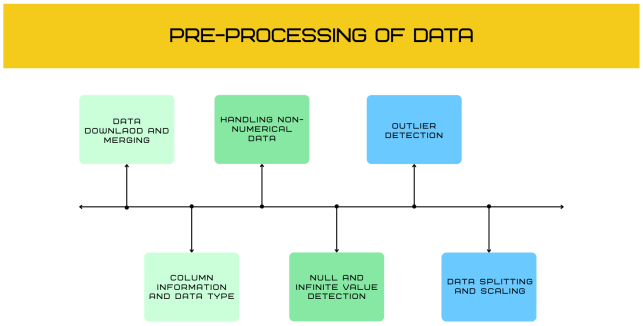

Fig. 2. Pre-Processing Steps

The dataset is separated into attack types, such as DoS (Denial of Service), DDoS (Distributed Denial of Service), Brute Force, and SQL Injection and many more as shown in Fig. 1 which allows for attack detection based classification. For this study, we concentrate on preprocessing the dataset to eliminate extraneous features and convert the data into a format appropriate for machine learning analysis. In this research, we use the CICIDS 2017[1] dataset to predict the type of network assault, with a focus on improving model performance through dimensionality reduction using PCA[20] and presenting the findings in real time with Power BI[21].

2) *Pre-Processing*

As shown in Fig. 2, to create a consistent and high quality dataset, we performed the following pre-processing steps:

*a) **Data Download and Merging**:* The CICIDS 2017[1] dataset, containing daily logs from an experiment conducted over six days (Monday to Sunday), was downloaded from the official website as shown in Fig. 3. We then merged each CSV file into one unified dataset to simplify the analysis.

*b) **Column Information and Data Types**:* For a well-structured dataset, we verified for column names, data types,

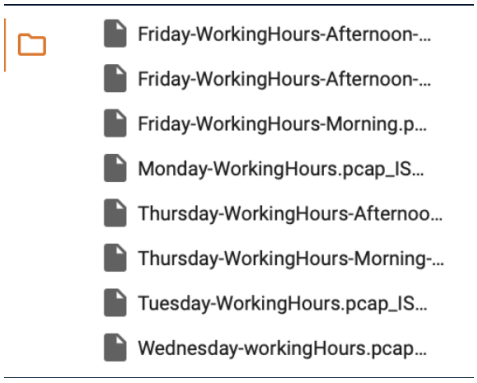

Fig. 3. Various CSV Files

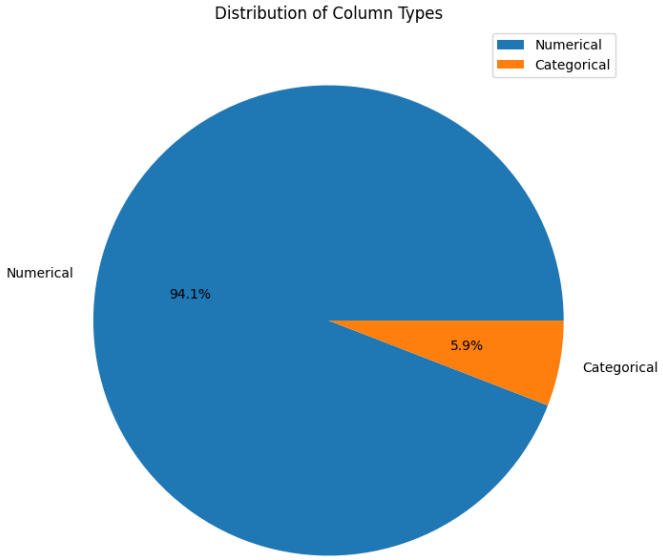

Fig. 4. Distribution of Column Types

and unique values. Numerical values were dominant in the majority of the columns, whereas the minority were non-numerical in nature, i.e., Label, Flow ID, Timestamp, Source IP, and Destination IP. Column type distribution is presented in Fig. 4.

*c) **Handling Non-Numerical Data**:* The Label column was categorical and consisted of different types of attacks and benign traffic. We encoded these numerically using Label Encoding, with each unique category being assigned an integer.

Feature Selection: We removed non-informative features such as Flow ID, Source IP, and Destination IP.

These features were removed since: They contain special identifiers that are not useful for learning patterns. Their presence can lead to data leakage because an IP address can uniquely determine the attack sources in the training set.

*d) **Null and Infinite Value Detection**:* The dataset was examined for null values, particularly in the Flow Bytes/s column, which contained missing data. We opted to impute the missing values with the median of each label group, as the median is more robust to outliers than the mean. Additionally, we checked for infinite values in columns like Flow Bytes/s and Packet, which were found and replaced with null values to ensure consistency.

*e) **Outlier Detection**:* After cleaning the data, we checked for outliers in the numerical columns but found none. Given the nature of this dataset and its relevance for real-time prediction, every piece of information was considered important.

*f) **Data Splitting and Scaling**:* Data was split into 80-20 ratio for training and testing sets to ensure that the testing of models was done on unseen data.

Standardization:

We employed Standard Scaling (Z-score normalization) with StandardScaler() from sklearn.preprocessing to normalize all the numerical features to the same scale. Scaling was performed after splitting to prevent data leakage and ensure that the test set was scaled based only on the training set statistics.

This keeps features with large numeric values (e.g., Packet Length Mean, Flow Duration) from overpowering features at a smaller scale.

Data Preprocessing and Its Impact on Model Performance: By downloading and merging the CICIDS 2017[1] dataset, followed by a series of preprocessing steps including handling non-numerical columns, addressing missing and infinite values, and applying feature selection, we ensured that our dataset was clean and ready for analysis. In addition, we standardized the data after splitting them into training and testing sets, which improved the model's ability to learn effectively. This rigorous preprocessing ensured fewer discrepancies during model training, leading to more efficient learning. By carefully cleaning and preparing the dataset, we improved the quality of the input data, resulting in a model that could learn more generalized features and perform well across a variety of attack types.

B. Machine Learning Models:

After preprocessing and scaling the dataset using Standard Scaling (Step f in Preprocessing), we trained and tested different supervised machine learning models for classifying network traffic into attack and benign classes. Scaling was required to ensure that all the features would have equal influence on the model so that large numerical value features would not disproportionately bias the prediction. We scaled the dataset and then trained the following models:

- Decision Tree Classifier[11]: A tree model that partitions data based on feature importance.
  Strengths: Interpretable, good non-linearity handling.
  Weaknesses: Overfits if not pruned.
- Random Forest Classifier[12]: An ensemble of decision trees that generalizes through averaging over many trees.
  Strengths: Less likely to overfit, handles imbalanced data well.
  Weaknesses: Computationally intensive for large data.
- XGBoost Classifier[13]: A gradient boosting algorithm with an emphasis on speed and performance.
  Strengths: High accuracy, handles missing values, efficient on big data.

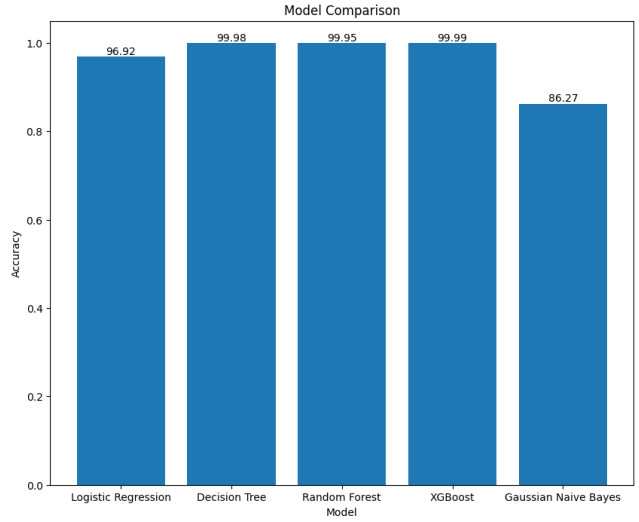

Fig. 5. Accuracy of Different Model

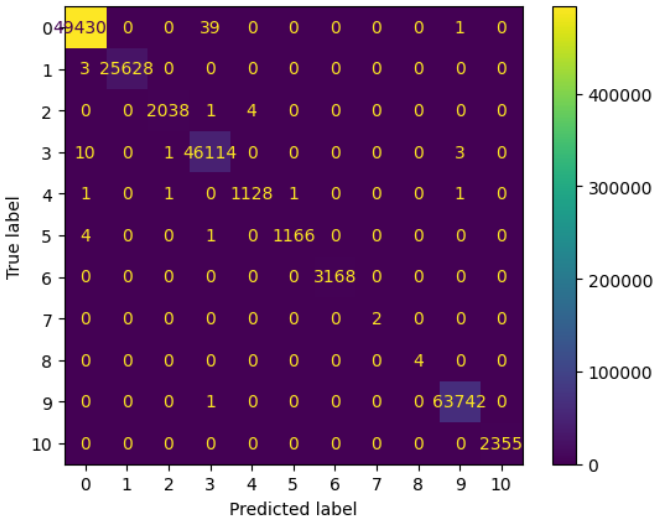

Fig. 6. Confusion Matrix of XGBoost Classifier

Weaknesses: Requires careful hyperparameter tuning.

- Logistic Regression[14]: Statistical model predicting probability of class membership.
  Strengths: Simple, simple to understand, good default classifier.
  Weaknesses: Maintains a linear relationship assumption between features and target variable.
- Gaussian Naïve Bayes[15]: Probabilistic classifier according to Bayes' Theorem with Gaussian distributions.
  Strengths: Good for small datasets, fast training.
  Weaknesses: Assumes feature independence, potentially broken in real data.

Hyperparameter Tuning: To achieve optimum performance, we applied Grid Search hyperparameter optimization on Random Forest and XGBoost, and tuned parameters such as:

1.max_depth (tree depth)

2.n_estimators (size of trees in ensemble methods)

3.learning_rate (in case of XGBoost)

4.min_samples_split (number of samples needed to split a node)

Model Comparison: After training the models, we compared their performance and found that the XGBoost model[13] produced the highest accuracy (as shown in Fig. 5), making it the best-performing model for this task. This was confirmed by calculating the accuracy for each model and evaluating it against the test dataset.

Confusion Matrix: To gain deeper insights into the performance of each model, we analyzed the confusion matrix for each. The confusion matrix is a valuable tool for understanding the types of errors each model made, especially in distinguishing between different attack types. For each model, we plotted the confusion matrix.

By carefully analyzing the confusion matrices presented in Fig. 6, 7, 8, 9, and 10, we observed distinct patterns in the classification performance of each model. Among all the models, the Gaussian Naive Bayes classifier[15] demonstrated the weakest performance, as evidenced by the poor values along its diagonal, which correspond to correct predictions. This indicates a higher rate of misclassification compared to the other models, making it unsuitable for our analysis.

The Logistic Regression model[14] also exhibited limitations, particularly with an elevated number of misclassified instances concentrated in the first row of its confusion matrix. This suggests a bias in its predictions, which compromises its overall reliability when compared to tree-based models. Focusing on the tree-based classifiers — Decision Tree[11], Random Forest[12] , and XGBoost[13] — we evaluated their confusion matrices based on the number of off-diagonal elements, which represent misclassified samples. XGBoost[13] emerged as the most robust model, with the fewest off-diagonal values, implying the highest precision in its predictions. Additionally, XGBoost's[13] confusion matrix contained the largest proportion of zero entries outside the diagonal, further supporting its superior ability to accurately classify data across all categories. In summary, the confusion matrix analysis reinforces our conclusion that the XGBoost classifier[13] significantly outperforms the other models, not only in terms of accuracy metrics but also in its ability to minimize misclassifications. This finding aligns with and substantiates the results derived from the accuracy metrics, solidifying XGBoost[13] as the optimal choice for our dataset.

Feature Importance and Model Interpretability:

For the tree-based models (Decision Tree[11], Random Forest[12], and XGBoost[13]), we plotted the feature importance, as shown in Fig. 11, 12, and 13, to analyze the relative contribution of each feature to the model's predictions. Feature importance quantifies how much each feature influences the model's decision-making process, providing insights into the underlying factors driving predictions. This analysis is critical for understanding the model's behavior and identifying the key indicators of network traffic anomalies or attacks.

Similarly, for Logistic Regression[14], we analyzed the coefficients of the model as shown in Fig. 14, which indicate the relationship between each feature and the model's predictions. Positive coefficients suggest that higher values of a feature

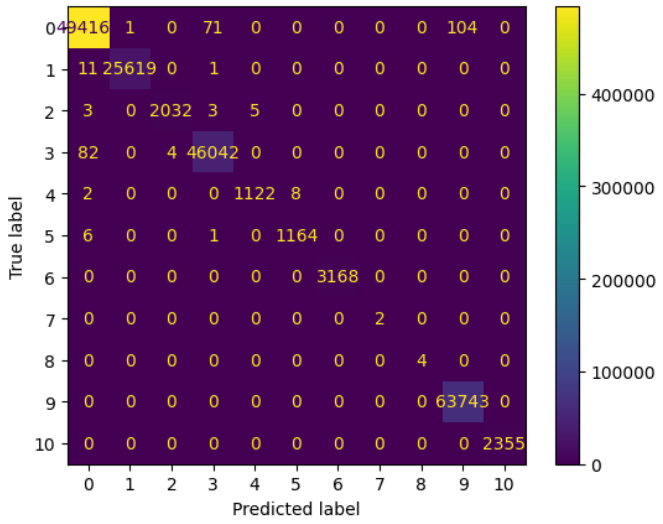

Fig. 7. Confusion Matrix of Random Forest Classifier

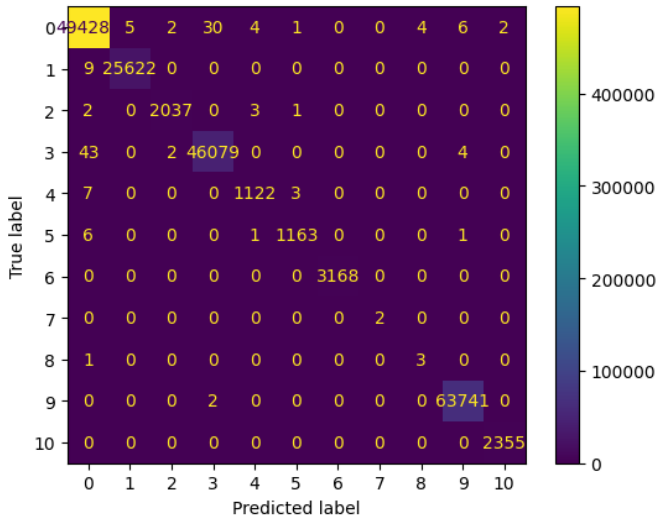

Fig. 8. Confusion Matrix of Decision Tree Classifier

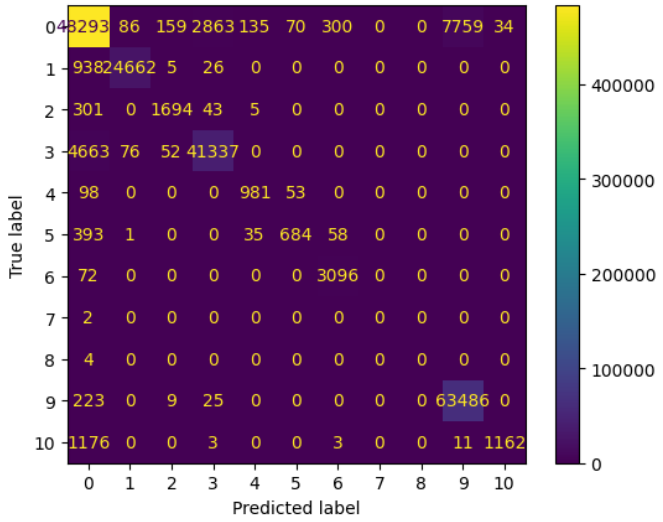

Fig. 9. Confusion Matrix of Logistic Regression

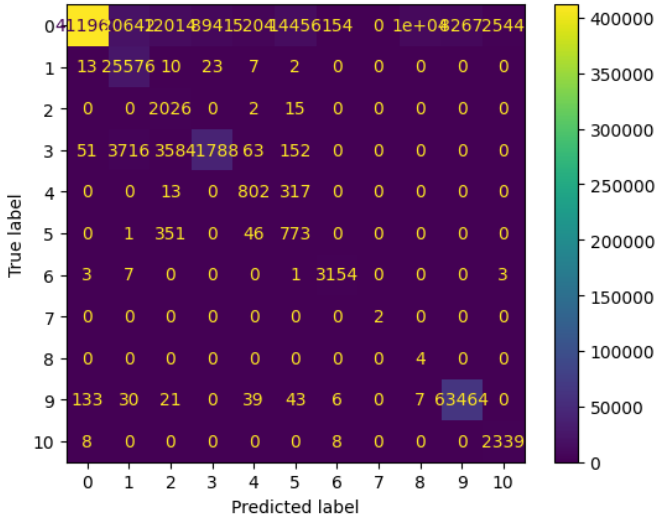

Fig. 10. Confusion Matrix of Gaussian Naive Bayes

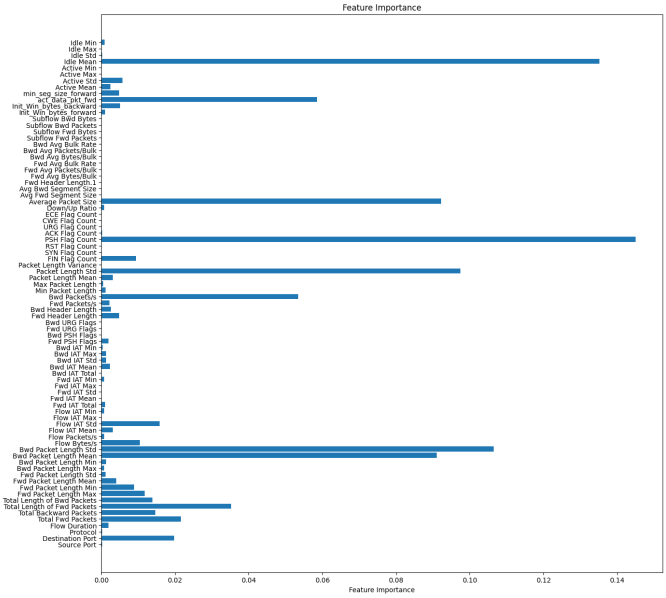

Fig. 11. Feature Importance from XGBoost

increase the likelihood of an attack being classified, while negative coefficients suggest the opposite. We visualized these coefficients using bar plots for easier interpretation.

Bar plots were used to visualize the feature importance for each tree-based model, making it easier to compare their interpretations. From the feature importance plots, we observed that Random Forest[12] emphasizes a broader range of features, assigning importance to many of them. In contrast, XGBoost[13] and Decision Tree classifiers[11] identify a more focused subset of significant features. This suggests that Random Forest[12] may overfit the dataset by relying on too many features, which could reduce its generalization performance. On the other hand, XGBoost[13] and Decision Tree models[11] highlight fewer, more critical features, making them more efficient for our analysis.

Comparing XGBoost[13] and Decision Tree[11],

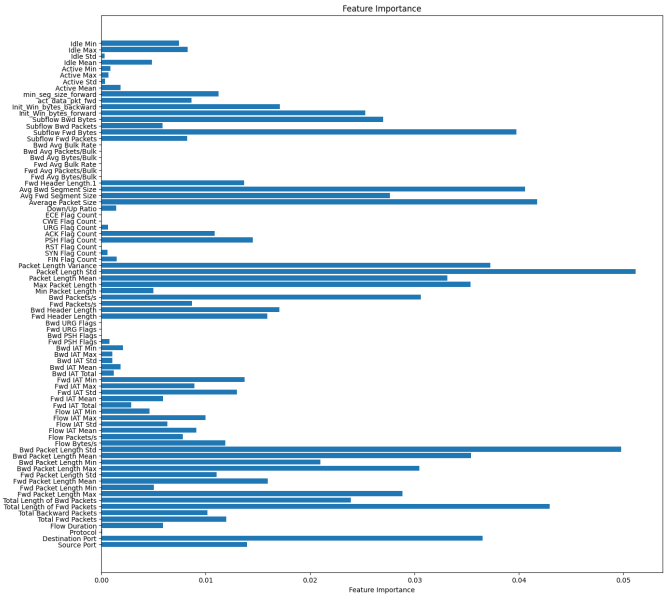

Fig. 12. Feature Importance from Random Forest

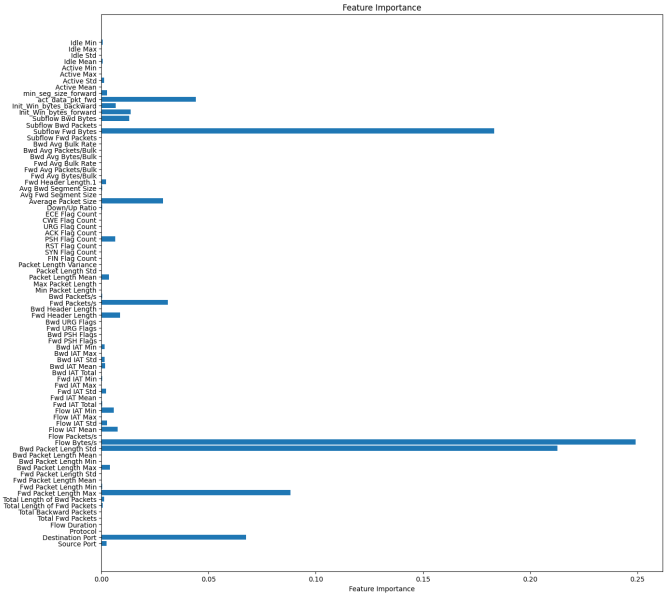

Fig. 13. Feature Importance from Decision Tree

XGBoost[13] provides a clearer and more distinct ranking of feature importance, supporting its overall superior performance in the accuracy metrics and confusion matrix analysis. This alignment across multiple evaluation criteria reinforces the conclusion that XGBoost[13] is the best performing model for our dataset. Consequently, we have chosen XGBoost[13] for further visualization and analysis in this study.

While Logistic Regression[14] offers interpretable coefficients that indicate the contribution of each feature, its feature importance plot shows a less clear delineation of significant predictors compared to tree-based models. Logistic Regression[14] assumes a linear relationship between features and the target variable, which may oversimplify the complex

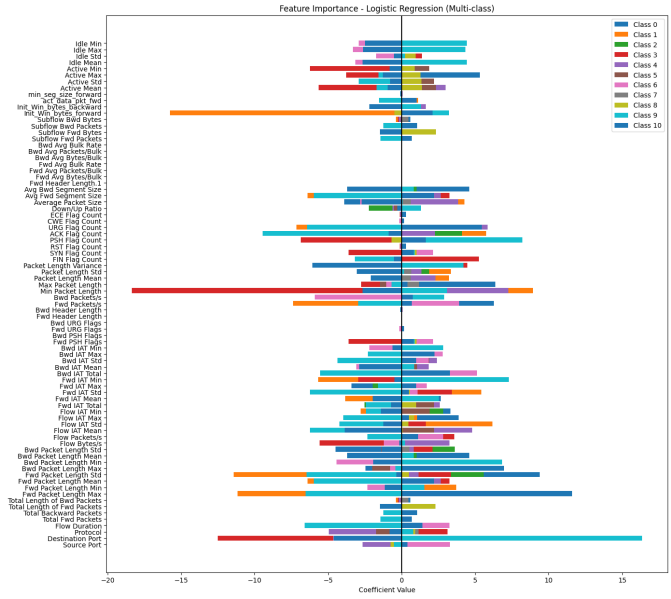

Fig. 14. Feature Importance from Logistic Regression

patterns inherent in our dataset. Additionally, its performance, as seen in the confusion matrix, is inferior to XGBoost[13], Random Forest[12], and Decision Tree[11], particularly in handling imbalanced classes. These limitations make Logistic Regression[14] unsuitable for our objectives, and we prioritize tree-based models, with XGBoost[13] being the most optimal choice.

C. PCA(Principal Component Analysis)

*a) **PCA Overview**:* Principal Component Analysis (PCA)[20] is a dimensionality reduction technique that transforms the original feature space into a smaller set of uncorrelated features, known as principal components. These components capture the maximum variance present in the data, reducing the number of features while retaining the most significant information. In the context of our project, PCA[20] was applied after preprocessing to reduce the number of features, helping to mitigate the risk of overfitting and enhance computational efficiency.PCA[20] allows the model to focus on the most important underlying patterns, which enables faster training times and lower memory consumption, while still allowing a high degree of accuracy.

As this classifier performed best, based on the observations of the previous sections, we extend our analysis by taking its performance from a different perspective. Instead of just accuracy, we look into other metrics like time taken for training the model and memory usage. These factors are very crucial in practical applications, specially for real-time or resource constrained environments.

Further to investigate possible optimizations, we have applied PCA[20] in order to see its effect on these performance metrics. PCA[20] is a technique for reducing the dimensionality of data to a lower dimensional space in such a manner that most of the information (variance) can be preserved. This helps in reducing computational complexity and memory

requirements without affecting the predictive power of the model significantly.

*b) **Implementation of PCA**:* This section considers the influence of applying PCA[20] on model performance, taking into account the variance threshold with accuracy, memory usage, and training time. We applied PCA[20] using the built-in Python library to reduce the dimensionality of the dataset, retaining various proportions of variance of the dataset. A threshold based on explained variance is another way of specifying the amount of explained variance one wants to retain; the higher the threshold, the more information is preserved, but at greater computational cost. To understand the trade-offs, we experimented with four different variance thresholds: 0.900, 0.950, 0.990, and 0.999, and evaluated their impact on three important metrics.

Accuracy (%): The predictive performance of the model.

Memory Usage (GB): The memory consumed during training.

Time Taken to Train Model(s): The computational efficiency of the model. The model used was the XGBClassifier.

Results and Observations:

The results, summarized in Table I, reveal that increasing the variance threshold slightly improves accuracy. However, this improvement comes at the cost of significantly higher training time, while memory usage remains relatively stable at higher thresholds (0.990 and 0.999). The model exhibits faster training times for thresholds 0.900 and 0.950.

To provide a clearer understanding, Fig. 15 visually illustrates the relationship between variance thresholds and the three performance metrics: Accuracy, Memory Usage, and Time Taken to Train. Below is an analysis of the trends observed in the graph:

1.Accuracy:

- Represented by the blue line, accuracy shows a gradual increase as the variance threshold rises.

- A noticeable improvement is observed from 0.90 to 0.99. However, beyond 0.99, the gains diminish, highlighting a point of diminishing returns.

2.Memory Usage:

- Depicted by the orange line, memory usage remains stable across higher thresholds (0.99 and 0.999).

- At lower thresholds (0.90 and 0.95), there is a minor reduction, but the effect on overall resource efficiency is minimal, suggesting that memory usage is not significantly impacted by variance thresholds beyond a certain point.

3.Time Taken to Train:

- The green line indicates a steep rise in training time as the threshold increases.

- Notably, the jump between thresholds 0.99 and 0.999 is substantial, demonstrating the trade-off between achieving marginally better accuracy and the increased computational cost.

The significant rise in training time between 0.99 and 0.999 emphasizes the inefficiency of extremely high variance thresholds. Since the differences in accuracy between 0.99 and 0.999 are minimal, and the training time for 0.99 is significantly lower (235 seconds compared to 272 seconds for 0.999), choosing 0.99 as the optimal variance threshold offers a practical balance. This threshold effectively retains sufficient information while conserving computational resources, making it a more efficient choice for further analysis.

Variance Threshold	Accuracy (%)	Memory_Usage (GB)	Time Taken to Train Model(s)
0.900	99.745191	3.160116	160.895469
0.950	99.754102	3.026665	190.082128
0.990	99.806783	3.007600	235.115457
0.999	99.826480	3.007600	272.662835

TABLE I
PERFORMANCE METRICS FOR VARIANCE THRESHOLD

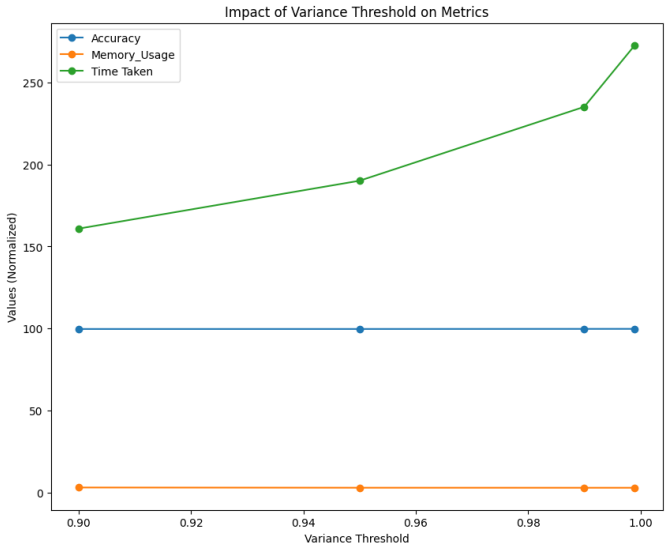

Fig. 15. Impact of Variance Threshold on Metrics

Based on the selected variance threshold (0.99) and the chosen model (XGBoost)[13], PCA[20] was applied to assess its impact on key performance metrics. The results, summarized in Table II and visually depicted in Fig. 16, compare the model's performance with and without PCA. The evaluated metrics include Accuracy, Memory Usage, and Time Taken for training.

To quantify the impact of PCA[20], a detailed percentage based comparison is provided below, highlighting the trade-offs in terms of accuracy reduction, memory savings, and training time improvement. These insights help determine the practical benefits of applying PCA[20] in the given context.

Observations and Calculations:

- **Accuracy:**

$$Difference = 99.989370 - 99.806158 = 0.183212\,(\%)$$

$$PercentageReduction = \frac{0.183212}{99.989370} \times 100 \approx 0.18\,(\%)$$

- **Memory Usage:**

$$Difference = 24.395108 - 24.390344 = 0.004764\,(MB)$$

Dataset	Accuracy(%)	Memory Usage (MB)	Time Taken(s)
Without PCA	99.989370	24.395108	290.131819
With PCA	99.806158	24.390344	210.937284

TABLE II
COMPARISON OF METRICS WITH PCA AND WITHOUT PCA

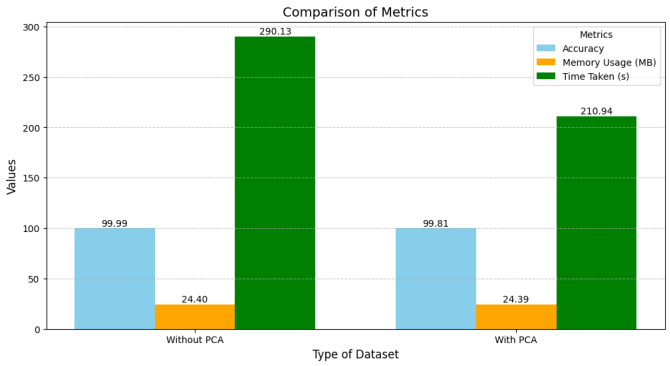

Fig. 16. Peformance Metrics Comparison with and without PCA

$$PercentageReduction = \frac{0.004764}{24.395108} \times 100 \approx 0.02\,(\%)$$

- **Time Taken:**

$$Difference = 290.131819 - 210.937284 = 79.194535\,(s)$$

$$PercentageReduction = \frac{79.194535}{290.131819} \times 100 \approx 27.29\,(\%)$$

Analysis:

From the above comparison, it is evident that applying PCA[20] with a variance threshold of 0.99 provides a significant reduction in training time while maintaining comparable accuracy and memory usage. The minor accuracy drop of 0.18% is an acceptable trade-off for the 27.29% reduction in training time, especially for scenarios where computational efficiency is prioritized.

This comparison and the Fig. 16 highlights the practicality of incorporating PCA[20] in the preprocessing pipeline, particularly for large datasets where reducing training time can lead to faster iterations and model optimization.

D. Real Time Data Simulation in Power BI

a) Models and Tools Overview: Before diving into the implementation, the following models and tools were finalized for real-time data simulation and prediction:

- **PCA Instance:**Used for dimensionality reduction in the PCA[20]based approach.Saved and loaded using the joblib package.
- **XGB Model:**Two models were used:
  1. XGB Model trained with PCA
  2. XGB Model trained without PCA.

  The built in save and load functions of XGBoost[13] were used for these models to preserve model weights and configurations.
- **Standard Scaler:**A separate scaler instance was trained for PCA-transformed data and the original dataset.Saved and loaded using the joblib package.
- **Label Encoder:**Common for both approaches to encode the predicted labels into human readable or categorical formats. Saved and loaded using the joblib package.

This combination of tools and models ensures consistency, efficiency, and accuracy throughout the workflow as shown in Fig. 17.

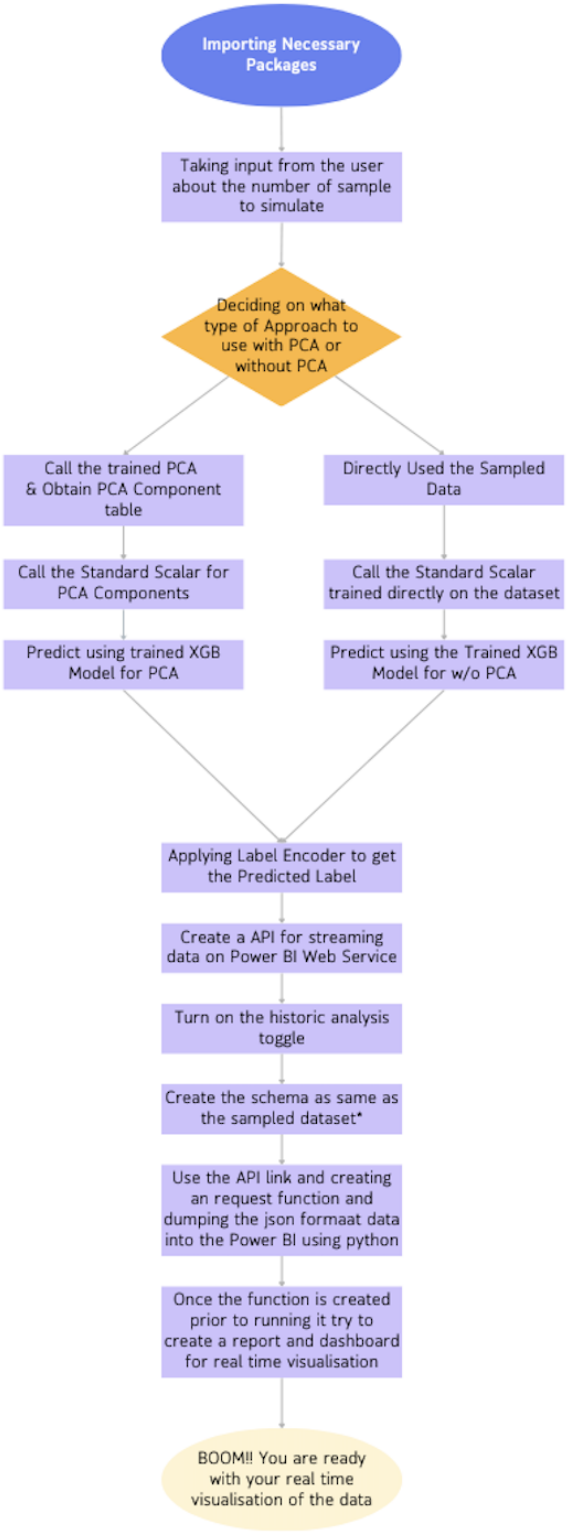

Fig. 17. Flowchart of the Implementation

b) Real-Time Data Simulation and Visualization Flowchart:

1. Importing Necessary Packages

Start by importing essential Python libraries required for preprocessing, scaling, encoding, modeling, and integration with Power BI[21].

2. Taking User Input

Collect input from the user to determine the number of samples to simulate for real-time prediction and visualization.

3. Deciding on the Approach

Based on the requirements, select one of the following approaches:

- With PCA: Use the pre-trained PCA[20] model to transform the dataset into its principal components.Apply the Standard Scaler instance trained specifically for PCA[20] transformed data to ensure consistent scaling.Use the XGB Model trained with PCA[20] to predict the outcome.
- Without PCA: Directly scale the sampled data using the Standard Scaler instance trained on the original dataset.Use the XGB Model trained without PCA to predict the outcome.

4. Applying Label Encoder

Use the Label Encoder instance to convert the predicted labels into human readable or categorical outputs as required for Power BI[21] visualization.

5. Streaming Data to Power BI

Step 1: Enable the Power BI[21] service with the appropriate workspace and dataset for streaming.

Step 2: Turn on the 'Historic Data Analysis' toggle for the Power BI[21] dataset to allow visualization of both past and real-time data.

Step 3: Create the schema for the streaming dataset, ensuring it matches the structure of the predicted data.

6. Building Reports and Dashboards

Create a report in Power BI[21] that connects to the streaming dataset. Design visualizations to represent predictions, trends, and insights from the data. Publish a dashboard linked to the report for live updates.

7. Sending Data to Power BI Using Python

Use the Power BI[21] API endpoint and Python's requests library to send real-time simulated data (in JSON format) into Power BI[21]. Ensure proper authentication and test the data pipeline to verify successful ingestion of data into Power BI[21].

c) Discussion:Real-Time Data Simulation in Power BI: The project integrates machine learning driven anomaly detection with real-time data streaming in Power BI. By simulating network traffic data and running it through a predictive model, we can show network anomalies in real-time, enhancing security monitoring.

Implementation Details:

- Data Preprocessing: Load network traffic dataset and drop unnecessary columns (Flow ID, Timestamp, IP Addresses).Standardize numerical features using pre-trained scaler.
- Feature Transformation:For PCA-based models, do dimensionality reduction. Ensure the processed data format matches the trained model input.
- Model Prediction: Load a pre-trained XGBoost classifier. Predict attack types from network traffic data. Convert numerical labels back to readable categories.
- Streaming to Power BI: Format the data into JSON. Push data rows sequentially to Power BI's REST API. Ensure smooth transmission without exceeding API limits.

The pseudo code below implements the mentioned steps:

```
# Load dataset
data = read_csv("network_traffic.csv")

# Preprocess Data
processed_data = preprocess_data(data)

# Apply Feature Transformation
if use_pca:
    transformed_data = apply_pca(processed_data)
else:
    transformed_data = apply_scaling(processed_data)

# Load Model & Predict
model = load_trained_model("xgboost_model")
predictions = model.predict(transformed_data)

# Decode Predictions
decoded_labels = decode_labels(predictions)

# Stream to Power BI
for each row in transformed_data:
    payload = format_to_json(row, decoded_labels)
    send_to_power_bi(payload)
```

Technical Challenges:

- Managing API Rate Limits: Challenge: Power BI implements request limits, which will fail when data streams too fast. Solution: Apply batch processing or implement a time gap between API calls.
- Real-Time Performance: Challenge: The detection speed may be affected by data latency. Solution: Maximize network transmission and utilize multi-threading to execute requests simultaneously.
- Data Security & Compliance: Challenge: Network traffic data may contain sensitive information (IP addresses, user logs). Solution: Anonymize the data before processing and enable secure API authentication (e.g., OAuth tokens).

Model Deployment in a Real Network Environment:

To integrate this system into a production network, the following are crucial considerations:

- Data Ingestion Pipeline: Use an real-time network packet capture tool (i.e., Wireshark, Zeek).Use Apache Kafka or AWS Kinesis for streaming at scale.
- Model Hosting & Inference: Deploy the trained model on AWS SageMaker or Azure ML for real-time inference. Set up a REST API with FastAPI or Flask to receive network packets and make predictions.
- Integration with Power BI: Save logs in a database (PostgreSQL, Snowflake, or Redshift) before streaming into Power BI. Use Power BI Dataflows or DirectQuery for dashboard refreshes in real-time.

E. Methods Result Comparison

The summary of different models' performance on intrusion detection is presented in Table III. Among the deep learning techniques, high accuracy was reported, first by Elmasry et al.[9], at 99.91% with DBN; then Kurniabudi et al.[4] presented an accuracy of 99.88% using J-48/C4.5. In ensemble methods, 99.89% was attained by Yuyang Zhou et al.[8] with C4.5+RF+Forest PA. Hybrid models like Zhang et al.[18] combined LSTM and Random Forest for 95.8% accuracy, showing the potential of combining techniques. Other machine learning methods also did well, such as Random Forest by Gamage et al.[23] with 99.86% and SVM by Wang et al.[2] with 93.7%. Without PCA, XGBoost topped them all with the highest accuracy of 99.99%, thereby showing the powers of advanced gradient boosting models in intrusion detection.

Method	Technique	Model	Accuracy (%)
Sharafaldin et al.[1]	Machine Learning	Random Forest	95.2
Ullah et al.[6]	Deep Learning	LSTM	99.95
Chen et al.[5]	Machine Learning	Decision Tree + PCA	92.8
Wang et al.[2]	Machine Learning	SVM	93.7
Ferrag et al.[17]	Deep Learning	CNN	94.5
Mondal & Sanchez[7]	Machine Learning	Gradient Boosted Trees	92.5
Zhang et al.[18]	Hybrid Approach	LSTM + Random Forest	95.8
Subramaniam et al.[19]	Deep Learning	1D-CNN	95.6
Kurniabudi et al.[4]	Machine Learning	J-48/C4.5	99.88
Gamage et al.[23]	Machine Learning	Random Forest	99.86
Vinayakumar et al.[10]	Deep Learning	DNN	95.6
Yuyang Zhou et al.[8]	Ensemble Machine Learning	C4.5+RF +Forest PA	99.89
Elmasry et al.[9]	Deep Learning	DBN	99.91
Yulianto et al.[24]	Machine Learning	Adaboost+EFS +SMOTE	81.83
Our Approach	Machine Learning	XGBoost+W/O PCA	99.99

TABLE III
COMPARISON OF VARIOUS METHODS AND MODELS IN TERMS OF ACCURACY

IV. DATA VISUALIZATION

Based on Fig. 18, we observe that the average flow duration was highest on 5th July 2017 and lowest on 6th July 2017. This could indicate that the experiments conducted on 5th July generated longer-lasting flows, potentially focusing on sustained traffic patterns. Conversely, the experiments on 6th July may have emphasized shorter duration flows, or fewer experiments might have been performed that day.

The box plot in Fig. 19 visualizes the flow duration across various attack types. It shows that benign traffic, represented by the orange box, has the shortest flow duration, suggesting

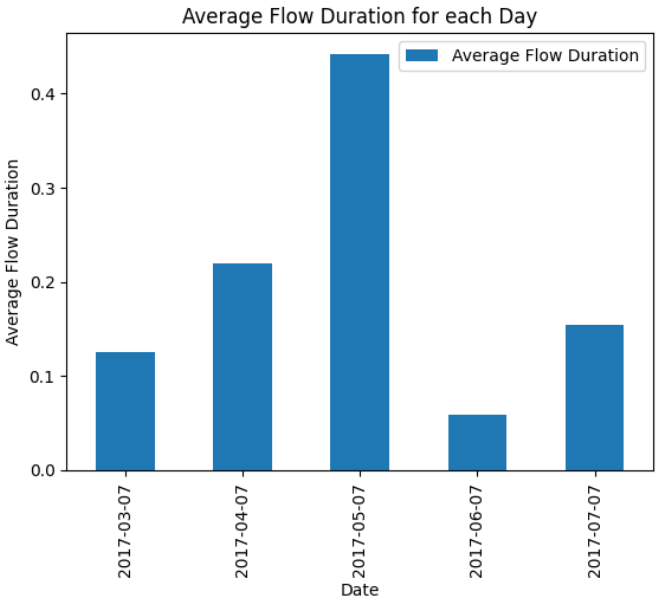

Fig. 18. Average Flow Duration for Each Day

regular traffic has brief interactions. Attack types like FTP-Patator and SSH-Patator (yellow and green boxes) exhibit similarly short flow durations, reflecting the repetitive nature of these attacks, which involve quick, repeated login attempts. In contrast, attacks such as DDoS and PortScan (light blue and green boxes) demonstrate more variability in flow duration, indicating differing attack behaviors. More persistent attacks like Infiltration and DoS Slowloris (teal and blue boxes) show higher median flow durations, with wider interquartile ranges, suggesting longer attack durations. DoS Slowhttptest and DoS Hulk (light purple and purple boxes) exhibit particularly high median flow durations, indicating prolonged, intensive attacks. DoS GoldenEye and Heartbleed (purple and pink boxes) have the highest flow durations with significant outliers, reflecting highly persistent attacks. These outliers, representing extreme flow durations, are normalized during preprocessing to ensure that the model is trained on the full spectrum of values, preventing the outliers from disproportionately affecting the model's performance. This normalization helps in training the model with all possible values, allowing it to generalize better across different attack scenarios and network conditions.

The bar chart in Fig. 20 displays the average count of various flags (FIN, SYN, RST, PSH, ACK, and URG) for each attack type. The BENIGN label shows relatively low values across all flags, indicating normal traffic with fewer network control signals. In contrast, attack types like DDoS, DoS GoldenEye, and DoS Hulk display significantly higher counts, especially for the URG and PSH flags, suggesting more intensive use of these flags in these attacks. Specifically, DoS Slowhttptest and DoS Slowloris exhibit high SYN and ACK flag counts, reflecting the slow and persistent nature of these attacks. FTP-Patator and SSH-Patator show relatively higher counts for the SYN and RST flags, which are typical for attempts to establish or reset connections. Heartbleed, Infiltration, PortScan, and SSH-Patator also have varying flag

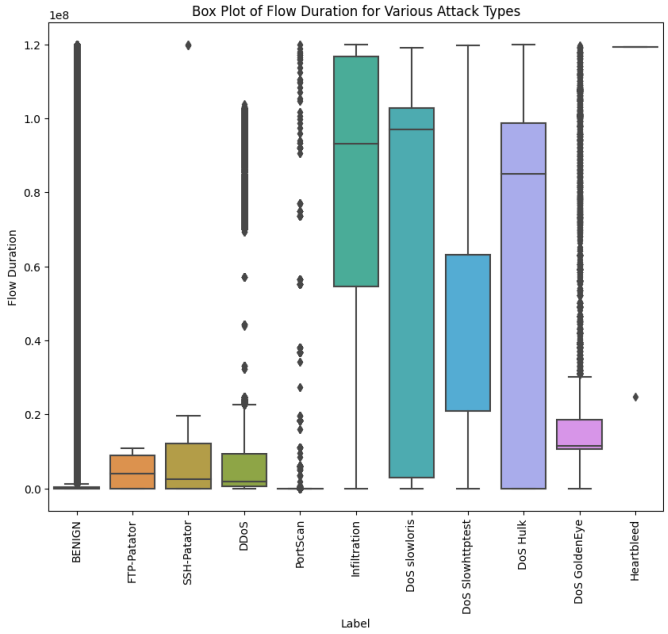

Fig. 19. Box Plot of Flow Duration for Various Attack Types

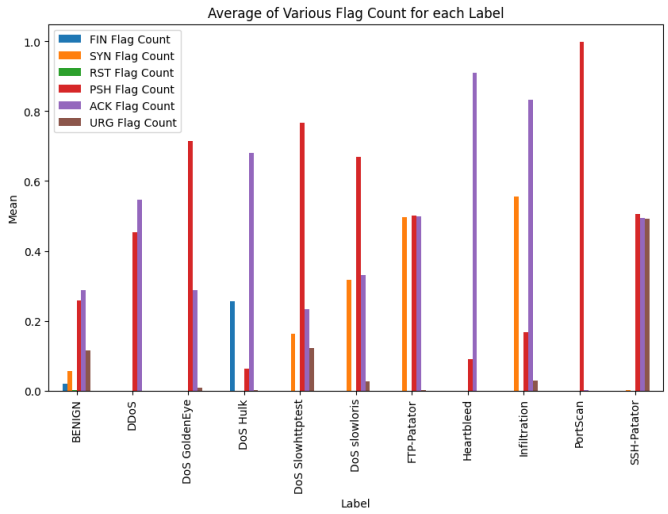

Fig. 20. Average of Various Flag Count for each Label

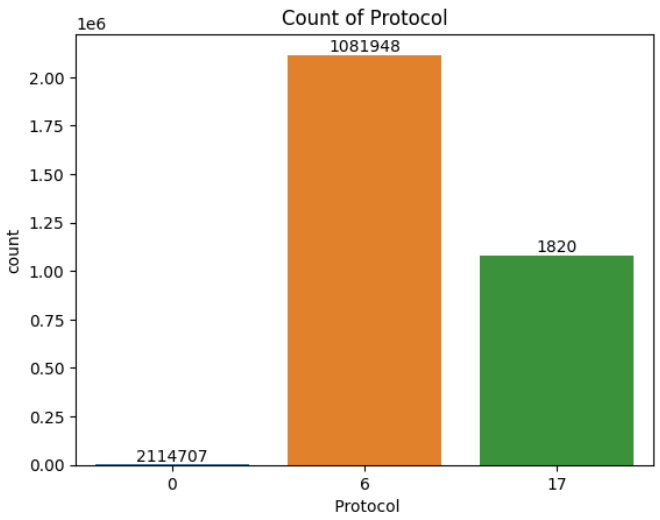

Fig. 21. Count of Protocol

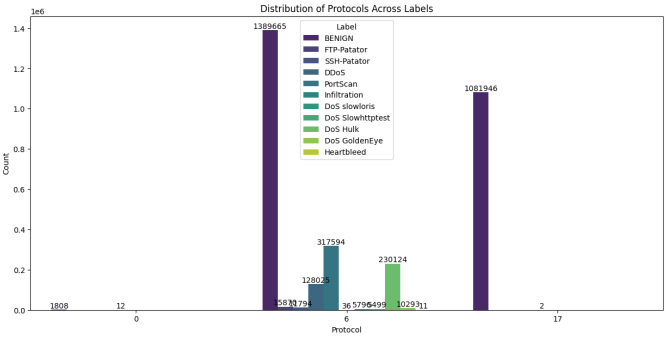

Fig. 22. Count of Label across various Protocol

counts, but not as pronounced as the aforementioned attacks. The graph highlights how different attack types manipulate these flags differently to create malicious traffic, and this behavior helps in distinguishing between attack types and benign traffic.

From the Fig. 21 we can infer that the Protocol 6 (likely TCP) dominates the dataset, reflecting its widespread use in network traffic. This is consistent with TCP's role in ensuring reliable communication for many common applications such as web browsing, file transfers, and email. Protocol 17 (likely UDP) is the second most frequent protocol in the dataset. UDP is typically used in applications where low latency and speed are prioritized over reliability, such as video streaming, gaming, and VoIP.In contrast, Protocol 0 is rarely observed in the dataset, suggesting it corresponds to specific or uncommon network events. Its infrequent use may indicate specialized traffic or non-standard network communications, which are not typically seen in regular network activity.

From the Fig. 22 we can infer that the 'BENIGN' label predominantly utilizes Protocol 6, with significantly fewer instances of other protocols such as Protocol 17 or Protocol 0. This indicates that benign network traffic primarily consists of communication using TCP, while other protocols are less frequently observed in normal traffic.On the other hand, attack types like 'PortScan' and 'DoS Hulk' exhibit a more diverse distribution across protocols. This suggests that certain attack types are more closely associated with specific protocol usage patterns. For instance, 'PortScan' may involve multiple protocols to probe various network services, while 'DoS Hulk' could show a stronger correlation with a particular protocol, reflecting the nature of the attack and its impact on the network.

Based on the Fig. 23 we can infer that the relationships between features like Flow Duration, Total Fwd Packets, and Total Backward Packets reveal distinct clusters, potentially aiding in label classification. Attack labels such as 'DoS Hulk' or 'Port Scan' tend to show dispersed values, distinguishing them from BENIGN traffic, which is more clustered. Feature Relationships: The diagonal histograms reveal that features

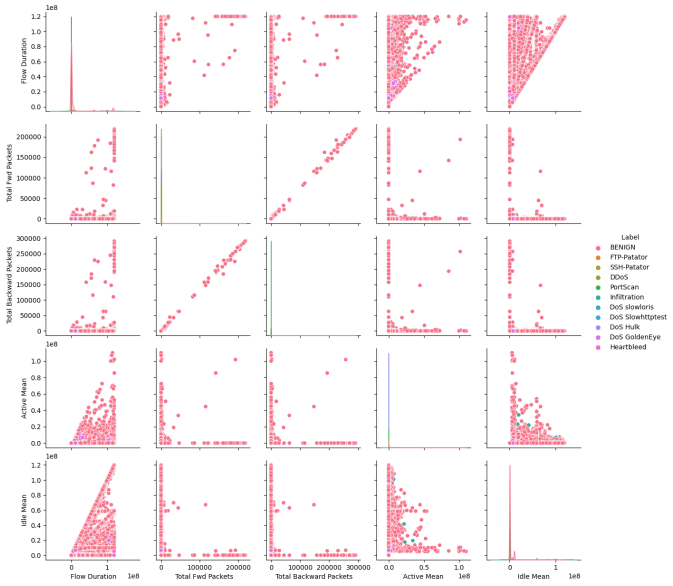

Fig. 23. Pair Plot on Attributes such as Flow Duration, Total Fwd Packets, Total Bwd Packets, Active and Idle Mean

like Flow Duration and Active Mean are heavily skewed, showing a concentration of values near the lower end .Label Separation: Certain attack types, such as 'DoS Hulk' are distinguishable in specific feature pairings (e.g., Flow Duration vs. Total Fwd Packets), showing potential for effective classification.

V. Future Work

1. Advanced Dimensionality Reduction Techniques

Compare with other dimensionality reduction methods, including t-SNE, UMAP, or LDA, in terms of effectiveness to PCA[20] for both visualization and classification performance. Look into nonlinear PCA[20] or kernel based methods that can handle complex interactions between features.

2. Model Optimization

Carry out hyperparameter tuning by either grid search or Bayesian optimization in order to further improve the performance of the XGBoost model[13]. Try different ensemble methods, such as Stacking or Boosting, with PCA[20] versus non-PCA models to see which has better accuracy.

3. Scalability Analysis

This will test the performance of both XGBoost[13] with and without PCA[20] on larger datasets, focusing on their scalability in terms of computation time and memory. Integrate distributed frameworks such as Apache Spark or Dask for datasets larger than current hardware limitations.

4. Real-Time Analysis

Deploy the model in an actual live-streaming environment using Apache Kafka or Flink, etc., to perform real-time network traffic data analysis for intrusion detection. Build a real-time dashboard to monitor predictions and visualize critical metrics dynamically.

5. Comparison with Other Models

Observe performances of deep models (Auto-encoder, LSTMs,.) used as dimensionality reducers or as state-of-the art classifiers. Compare tree-based models against other classifiers, like SVM[16], Neural Networks[22], or k-NN, in order to decide upon the best approach for any given scenario.

6. Improved Visualization Techniques

Create more interactive visualizations in Power BI[21] or Tableau that better illustrate the impact of PCA[20] on different metrics. Use 3D or multidimensional plots to show the relationships between the PCA[20] components and classification outcomes.

7. Tensor-Based Modeling for Complex Network Patterns

Future work can advance the research in the application of tensors in better describing multi-relational and high-dimensional network traffic data. Techniques such as the Tensor Levenberg-Marquardt Algorithm (TLMA) [26] and the Tensor Extended Kalman Filter (TEKF) [25] enable the description of input, output, and state variables in terms of generic tensor structures. They are suited for identifying nonlinear temporal evolution and complex relations that happen in real-world traffic and intrusion attack scenarios. These models can be combined to significantly enhance predictive accuracy, scalability, and robustness in network anomaly detection systems.

VI. Paper Highlights

- **Superior Performance of XGBoost:** Among the five machine learning models evaluated, XGBoost[13] emerged as the most accurate and effective in detecting cyber attacks, demonstrating its robustness in handling complex patterns and feature interactions.
- **Feature Importance Analysis:** Random Forest[12]tended to assign importance to a wide range of features, potentially leading to overfitting. In contrast, XGBoost[13] and Decision Tree[11] focused on a smaller, more critical subset of features, enhancing their interpretability and efficiency. Logistic Regression[14], while offering interpretable coefficients, struggled to clearly identify distinct features due to its linear assumptions, making it less suitable for the dataset's complexities.
- **Impact of PCA on Model Efficiency:** Implementing Principal Component Analysis (PCA)[20] led to a negligible reduction in accuracy but significantly improved computational efficiency and reduced training time. This trade-off highlights the practicality of dimensionality reduction techniques in handling large datasets without compromising performance.
- **Insights into Attack Detection:** It was found that different types of cyber attacks manipulate flags differently in the network. This will be a very good indicator that will help in finding malicious traffic from benign traffic, featuring engineering and domain specific knowledge become important in detection.

VII. Conclusion

This paper emphasizes how important data visualization and preprocessing have become in value addition to network traffic analysis in cybersecurity. Our group processed, analyzed,

and visualized the network traffic of the CICIDS 2017[1] dataset based on this, showing visualized patterns in network traffic and hence providing profound insight into cybersecurity threats. To improve model performance, cleaning was done, some features were transformed, and appropriate feature selection was performed. It used dimensional reduction, generally by Principal Component Analysis (PCA)[20], to enable computational efficiency enhancement with limited loss of information and no real reduction in predictive accuracy. Of the five models tested herein, such as XGBoost[13], Random Forest[12], Decision Tree[11], Logistic Regression[14], and Gaussian Naive Bayes[15], XGBoost[13] presented the best in terms of high intrusion detection rate. We have also compared the same with the work of other authors. We also made an interactive platform in Python and PowerBI to simulate real-time data for analyzing attack types, feature importance, and performance metrics. Our visualizations depicted how different cyberattacks manipulated network flags and this distinction can be used to detect malicious traffic. Our findings pointedly highlight how data analytics and visualization are very paramount in cybersecurity infrastructure development and intrusion detection.

ACKNOWLEDGMENT

We would like to extend our deep appreciation to San José State University for this opportunity and all the resources given to us throughout this project. We are grateful for the project guidance by Dr. Shih Yu Chang and to Sourab Saklecha for the resources he shared that helped us go deeper with our data visualizations. Finally, thanks to the Canadian Institute of Cybersecurity, which made this project viable with its openly available database.

REFERENCES

[1] 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*, 2018, pp. 108–116.

[2] Y. Wang, J. Liu, J. Zhang, and W. Zhang, "PCA-based dimensionality reduction for network intrusion detection," *IEEE Access*, vol. 8, pp. 123091–123104, 2020, doi: 10.1109/ACCESS.2020.3016783.

[3] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive dataset for network intrusion detection systems," in *Proc. Mil. Commun. Inf. Syst. Conf.*, 2015, pp. 1–6.

[4] D. Kurniabudi, D. Stiawan, M. Y. B. Idris, A. M. Bamhdi, and R. Budiarto, "CICIDS-2017 dataset feature analysis with information gain for anomaly detection," *IEEE Access*, vol. 8, pp. 132911–132921, 2020, doi: 10.1109/ACCESS.2020.3009843.

[5] S. Chen, Q. Li, and Y. Wang, "A comprehensive visualization framework for intrusion detection using Tableau," *Comput. Secur.*, vol. 107, p. 102335, 2021, doi: 10.1016/j.cose.2020.102335.

[6] F. Ullah, S. Habib, A. Khan, and S. Khan, "Enhancing interpretability of intrusion detection systems using Power BI dashboards," *Expert Syst. Appl.*, vol. 184, p. 115635, 2021, doi: 10.1016/j.eswa.2020.115635.

[7] P. Mondal and J. Sanchez, "Real-time intrusion detection using machine learning in Docker environments," *J. Netw. Comput. Appl.*, vol. 184, p. 102773, 2021, doi: 10.1016/j.jnca.2021.102773.

[8] Y. Zhou, G. Cheng, S. Jiang, and M. Dai, "Building an efficient intrusion detection system based on feature selection and ensemble classifier," *Comput. Netw.*, vol. 174, p. 107247, 2020, doi: 10.1016/j.comnet.2020.107247.

[9] W. Elmasry, A. Akbulut, and A. H. Zaim, "Evolving deep learning architectures for network intrusion detection using a double PSO meta-heuristic," *Comput. Netw.*, vol. 168, p. 107042, 2020, doi: 10.1016/j.comnet.2019.107042.

[10] R. Vinayakumar et al., "Deep learning approach for intelligent intrusion detection systems," *IEEE Access*, vol. 7, pp. 41525–41550, 2019, doi: 10.1109/ACCESS.2019.2895334.

[11] J. R. Quinlan, "Learning decision tree classifiers," *ACM Comput. Surv.*, vol. 28, no. 1, pp. 71–72, 1996.

[12] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, pp. 5–32, 2001.

[13] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, 2016, pp. 785–794.

[14] M. P. LaValley, "Logistic regression," *Circulation*, vol. 117, no. 18, pp. 2395–2399, 2008.

[15] I. Rish, "An empirical study of the naive Bayes classifier," in *IJCAI 2001 Workshop Empir. Methods Artif. Intell.*, vol. 3, no. 22, pp. 41–46, Seattle, WA, USA, 2001.

[16] Y. Zhang, "Support vector machine classification algorithm and its application," in *Inf. Comput. Appl.: 3rd Int. Conf. ICICA 2012*, Chengde, China, Sept. 14–16, 2012, Proc. Part II, Springer, 2012, pp. 179–186.

[17] M. A. Ferrag, L. Shu, and Q. Li, "Convolutional neural network-based intrusion detection system for IoT networks," *Comput. Netw.*, vol. 181, p. 107420, 2022.

[18] Z. Zhang, S. Yu, and W. Wang, "Hybrid LSTM and Random Forest model for intrusion detection," *Future Gener. Comput. Syst.*, vol. 116, pp. 128–139, 2022.

[19] V. V. Subramaniam, S. Santhosh, and P. K. Nair, "1D-CNN for intrusion detection in cybersecurity," *J. Cybersecurity*, vol. 15, no. 4, pp. 243–251, 2021.

[20] D. J. Olive, "Principal component analysis," in *Robust Multivariate Analysis*, Springer, pp. 189–217, 2017.

[21] A. Ferrari and M. Russo, *Introducing Microsoft Power BI*. Microsoft Press, 2016.

[22] Y.-C. Wu and J.-W. Feng, "Development and application of artificial neural network," *Wireless Personal Communications*, vol. 102, pp. 1645–1656, 2018.

[23] S. Gamage and J. Samarabandu, "Deep learning methods in network intrusion detection: A survey and an objective comparison," *J. Netw. Comput. Appl.*, vol. 169, p. 102767, 2020.

[24] A. Yulianto, P. Sukarno, and N. A. Suwastika, "Improving Adaboost-based intrusion detection system (IDS) performance on CIC IDS 2017 dataset," in *J. Phys.: Conf. Ser.*, vol. 1192, p. 012018, IOP Publishing, 2019.

[25] S. Y. Chang, H. -C. Wu and Y. -C. Kao, "Tensor Extended Kalman Filter and its Application to Traffic Prediction," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 12, pp. 13813-13829, Dec. 2023, doi: 10.1109/TITS.2023.3299557.

[26] S. Y. Chang, H. -C. Wu, Y. -C. Kuan and Y. Wu, "Tensor Levenberg-Marquardt Algorithm for Multi-Relational Traffic Prediction," *IEEE Trans. Veh. Technol.*, vol. 72, no. 9, pp. 11275-11290, Sept. 2023, doi: 10.1109/TVT.2023.3270037.