DDOS ATTACK CLASSIFICATION WITH HYPERPARAMETER TUNING AND ENCHANCED PREDICTION TECHNIQUE

# A PROJECT REPORT

***Submitted by***

**GRAYSON P [211421104081]**

**GIRITHARAN K [211421104075]**

**GANESHRAGAVAN S [211421104073]**

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

**BACHELOR OF ENGINEERING IN**

**COMPUTER SCIENCE AND ENGINEERING**



**PANIMALAR ENGINEERING COLLEGE**

**(An Autonomous Institution, Affiliated to Anna University, Chennai)**

# APRIL 2025

**PANIMALAR ENGINEERING COLLEGE**

**(An Autonomous Institution, Affiliated to Anna University, Chennai)**

**BONAFIDE CERTIFICATE**

Certified that this project report **“DDOS ATTACK CLASSIFICATION WITH HYPERPARAMETER TUNING AND ENHANCED PREDICTION TECHNIQUE”** is the bonafide work of **“GRAYSON P [211421104081], GIRITHARAN K [211421104075]** and **GANESHRAGAVAN S [2114104073\]”** who carried out the project work under my supervision.

|  |  |
| --- | --- |
| **SIGNATURE** | **SIGNATURE** |
|  |  |
| **Dr. L. JABASHEELA, M.E. Ph.D.,** | **,** **M.C.A, M.Phil, M.B.A, M.E, Ph.D.,** |
| **HEAD OF THE DEPARTMENT** | **SUPERVISOR** |
| Department of Computer Science and Engineering, | Department of Computer Science and Engineering, |
| Panimalar Engineering College,  Nazarathpet,  Poonamallee,  Chennai, 600 123. | Panimalar Engineering College,  Nazarathpet,  Poonamallee,  Chennai, 600 123. |

Certified that the above candidate(s) was examined in the End Semester Project Viva-Voce Examination held on ..............................

**INTERNAL EXAMINER EXTERNAL EXAMINER**

**DECLARATION BY THE STUDENT**

We, **GRAYSON P [211421104081], GIRITHARAN K [211421104075] and GANESHRAGAVAN S[2114104073]** hereby declare that this project report titled “**DDOS ATTACK CLASSIFICATION WITH HYPERPARAMETER TUNING AND ENHANCED PREDICTION TECHNIQUE**”, under the guidance of **HEMALATHA DEVI, (deg & INIT).,** is the original work done by us and we have not plagiarized or submitted to any other degree in any university by us.

**GRAYSON P [211421104081]**

**GIRITHARAN K [211421104075]**

**GANESHRAGAVAN S [2114104073]**

**ACKNOWLEDGEMENT**

Our profound gratitude is directed towards our esteemed Secretary and Correspondent, **Dr. P. CHINNADURAI, M.A., Ph.D.,** for his fervent encouragement. His inspirational support proved instrumental in galvanizing our efforts, ultimately contributing significantly to the successful completion of this project.

We want to express our deep gratitude to our Directors, **Tmt. C. VIJAYARAJESWARI, Dr. C. SAKTHI KUMAR, M.E., Ph.D.,** and **Dr. SARANYASREE SAKTHI KUMAR, B.E., M.B.A., Ph.D.,** for graciously affording us the essential resources and facilities for undertaking of this project.

Our gratitude is also extended to our Principal, **Dr. K. MANI, M.E., Ph.D.,** whose facilitation proved pivotal in the successful completion of this project.

We express our heartfelt thanks to **Dr. L. JABASHEELA, M.E., Ph.D.,** Head of the Department of Computer Science and Engineering, for granting the necessary facilities that contributed to the timely and successful completion of project.

We would like to express our sincere thanks to Project Coordinator and Guide **Dr. G. SENTHIL KUMAR** , **M.C.A, M.Phil, M.B.A, M.E, Ph.D., Professor**and all the faculty members of the Department of CSE for their unwavering support for the successful completion of the project.

**GRAYSON P [211421104081]**

**GIRITHARAN K [211421104075]**

**GANESHRAGAVAN S [2114104073]**

**ABSTRACT**

Data privacy is crucial in the financial sector to safeguard clients' sensitive information, prevent financial fraud, ensure regulatory compliance, and protect intellectual property. With the rise of internet usage and digital transactions, maintaining privacy has become increasingly challenging. Distributed Denial of Service (DDoS) attacks pose a significant threat to client privacy, necessitating effective detection and prevention measures. Machine Learning (ML) offers a promising approach for enhancing cyber-attack detection systems. This paper proposes a hierarchical ML-based hyperparameter optimization technique for classifying network intrusions. Utilizing the CICIDS dataset, which includes logs of various attacks, the proposed method involves preprocessing the data with min-max scaling and SMOTE. Feature selection is carried out to identify the most significant features. Classification is then performed using XGBoost, LGBM, CatBoost, Random Forest (RF), and Decision Tree (DT) algorithms. The models' performance is evaluated using recall, precision, accuracy, and F1-score metrics.

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| S.No | FIGURES | PAGE NO |
| 1 | Architecture Diagram | 12 |
| 2 | Sequence Diagram | 17 |
| 3 | Use Case Diagram | 18 |
| 4 | Activity Diagram | 19 |
| 5 | Collaboration Diagram: | 20 |

|  |  |  |
| --- | --- | --- |
| **LIST OF ABBREVIATIONS** | | |
| CNN |  | Convolutional Neural Network |
| LCNN |  | Lookup based Convolutional Neural Network |
| RNN | - | Recurrent Neural Network |
| DEX | - | Dalvik Executables |
| TCP | - | Transmission Control Protocol |
| IP | - | Internet Protocol |
| HTTP | - | Hyper Text Transfer Protocol |
| ADT | - | Android Development Tool |

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER**  **NO.** | **TITLE** | **PAGE NO.** |
|  | **ABSTRACT** | **v** |
|  | **LIST OF FIGURES** | **vii** |
|  | **LIST OF ABBREVIATIONS** | **viii** |
| **1.** | **INTRODUCTION** |  |
| 1.1 | Problem Definition |  |
| 1.2 | Synopsis |  |
| **2.** | **LITERATURE REVIEW** |  |
|  |  |  |
| **3.** | **THEORETICAL BACKGROUND** |  |
| 3.1 | Implementation Environment |  |
| 3.2 | System Architecture |  |
| 3.3 | Proposed Methodology |  |
| **4.** | **SYSTEM IMPLEMENTATION** |  |
| 4.1 | Test Procedure  4.1.1 System Testing |  |
| 4.2 | Test Data and Output  4.2.1 Unit Testing  4.2.2 Functional Tests  4.2.3 Performance Test  4.2.4 Stress Test  4.2.5 Structured Test  4.2.6 Integration Testing |  |

|  |  |  |
| --- | --- | --- |
| **CHAPTER**  **NO.** | **TITLE** | **PAGE NO.** |
| 4.3 | Testing Techniques/Testing Strategies  4.3.1 Testing  4.3.1.1 White Box Testing  4.3.1.2 Black Box Testing  4.3.1.3 Integration Testing  4.3.1.4 Program Testing  4.3.1.5 Security Testing  4.3.1.6 Validation Testing  4.3.1.7 User Acceptance Testing |  |
| **5.** | **SYSTEM IMPLEMENTATION** |  |
| 5.1 | Performance Parameters |  |
| 5.2 | Results and Discussion |  |
| **6.** | **CONCLUSION & FUTURE WORK**  **APPENDICES** |  |
|  | A.1 SGD Goal  A.2 Source Code  A.3 Screen Shots  A.4 Plagirism Report  A.5 Paper Publication |  |
|  |  |  |
|  | **REFERENCES** |  |

**CHAPTER 1 INTRODUCTION**

* 1. **PROBLEM DEFINITION**

The escalating threat landscape in today's digital age is characterized by an unprecedented surge in the sophistication, frequency, and diversity of cyberattacks, posing a critical challenge to the security and resilience of networks worldwide. Traditional rule-based Intrusion Detection Systems (IDS), which rely on predefined signatures and static patterns of known malicious activities, are increasingly struggling to effectively identify and mitigate these evolving threats. Their inherent limitations in detecting novel, zero-day exploits and sophisticated attack campaigns that deviate from established behaviors result in a significant gap in security coverage. Furthermore, these legacy systems are often plagued by a high rate of false positives, generating numerous inaccurate alerts that can overwhelm security analysts, lead to alert fatigue, and potentially obscure genuine security incidents. The manual effort required to triage and investigate these spurious alarms consumes valuable resources and detracts from proactive security measures, hindering an organization's ability to maintain a robust security posture. While Machine Learning (ML)-based Intrusion Detection Systems offer a promising paradigm shift by leveraging algorithms capable of learning complex patterns from network traffic and identifying anomalous behaviors indicative of malicious activity, their successful deployment is not without considerable challenges. One significant obstacle is the prevalence of imbalanced datasets, where the overwhelming volume of normal network traffic dwarfs the instances of actual attacks. This inherent class imbalance can introduce a significant bias in ML models, leading them to be overly optimized for the majority class and exhibiting poor detection rates for the critical minority class representing security threats. Another crucial concern revolves around feature engineering and selection within network traffic data, which often contains a multitude of features, many of which may be redundant, highly correlated, or contribute minimally to the accurate classification of network behavior. The inclusion of such irrelevant or noisy features can not only increase the computational complexity and resource demands of ML models but also negatively impact their performance, generalization ability to unseen data, and overall interpretability. Finally, the inherent complexity of hyperparameter optimization for machine learning algorithms presents a substantial practical challenge. ML models are governed by numerous hyperparameters that dictate their learning process, architectural configuration, and ultimately, their predictive performance. Manually tuning these hyperparameters to achieve optimal results is an arduous, time-consuming, and often inefficient endeavor, demanding extensive experimentation, deep domain expertise, and significant computational resources. Suboptimally tuned hyperparameters can lead to the development of underperforming models characterized by lower detection accuracy, increased false alarm rates, and a limited capacity to adapt to new and emerging threats. To effectively address these multifaceted limitations inherent in existing intrusion detection methodologies, this project proposes the development and evaluation of a **Hierarchical ML-Based Hyperparameter Optimization Technique**. This innovative approach aims to significantly enhance the efficacy of intrusion detection by systematically tackling the critical issues of data preprocessing to handle imbalances and noise, implementing intelligent feature selection strategies to identify the most informative and discriminative features within network traffic, and, most importantly, employing a hierarchical methodology for the optimization of hyperparameters across a diverse and powerful set of machine learning classifiers, including state-of-the-art algorithms such as XGBoost, LightGBM, and CatBoost, alongside established techniques like Random Forest and Decision Tree. By adopting a hierarchical optimization strategy, this research seeks to efficiently explore the vast hyperparameter search space and identify the most effective configurations for each chosen classifier, ultimately leading to substantial improvements in intrusion detection accuracy, a significant reduction in the occurrence of false positive alerts, and the creation of a more efficient, robust, and scalable Intrusion Detection System capable of effectively safeguarding networks against the ever-evolving and increasingly sophisticated landscape of cyber threats.

# SYNOPSIS

The increasing threat of DDoS attacks demands efficient and scalable detection systems to ensure network security. Existing methods, while effective to some extent in identifying certain attack patterns or traffic anomalies, often face significant challenges related to achieving consistently high accuracy, particularly against sophisticated and evolving attack vectors that may mimic legitimate traffic or employ low-and-slow strategies to evade traditional thresholds. Furthermore, many current systems struggle with scalability when confronted with the sheer volume and velocity of traffic characteristic of large-scale DDoS attacks, potentially leading to performance bottlenecks and a failure to process and analyze network data in real-time. This lack of real-time performance can be critical, as delays in detecting and mitigating an attack can result in prolonged service disruptions and significant damage to the target infrastructure and its users. This research aims to overcome these limitations by proposing a hierarchical machine learning approach that leverages the ability of advanced algorithms to learn complex traffic patterns and identify subtle indicators of malicious activity that might be missed by traditional signature-based or statistical methods. The integration of hyperparameter optimization will be crucial in ensuring that the machine learning models are precisely tuned to achieve optimal detection accuracy, minimize false positives, and maintain robust performance across a diverse range of attack scenarios and network conditions. This optimization process will be applied within a hierarchical framework, potentially involving multiple stages of analysis or different levels of granularity in feature extraction and classification, to enhance both the accuracy and the scalability of the detection system. The goal is to create a solution that not only effectively detects and classifies DDoS attacks with a high degree of accuracy but can also adapt to new and emerging attack techniques, operate efficiently under high traffic loads, and provide timely alerts and mitigation capabilities, thereby significantly bolstering network security and resilience against the growing menace of distributed denial-of-service attacks.

**CHAPTER 2**

**LITERATURE REVIEW**

The increasing frequency and complexity of cyberattacks have driven extensive research into network intrusion detection systems (IDS). Traditional IDS rely on signature-based or anomaly-based detection methods, which often fail to identify novel attacks due to their dependency on predefined rules and patterns. As a result, machine learning (ML) and deep learning (DL) techniques have gained significant attention in intrusion detection due to their ability to recognize patterns and adapt to evolving threats. This section reviews existing research on ML-based IDS, focusing on data preprocessing, feature selection, hyperparameter optimization, and classifier performance.

**OLD SYSTEM**

The old system refers to classical Intrusion Detection Systems (IDS) that primarily rely on signature-based or rule-based detection mechanisms. These systems operate by using a database of known attack signatures to identify malicious activities, essentially detecting intrusions by matching network traffic patterns against these pre-defined signatures or rules. This approach is straightforward for known attack patterns; however, traditional IDS solutions that use signature-based or rule-based detection have limitations. They can be ineffective against new attack techniques, sophisticated attacks, and zero-day attacks that exploit unknown vulnerabilities. These systems also require frequent updates to their signature databases to remain effective, which can be challenging in rapidly evolving threat landscapes. In contrast to the newer machine learning-based systems, these classical IDS solutions do not learn from data or adapt to new attack patterns. They lack the capability to generalize and detect anomalies that deviate from known signatures, making them less effective in dynamic environments like modern Cyber-Physical Production Systems (CPPS) where attack patterns can change rapidly.

**NEW SYSTEM**

The new system is an Intrusion Detection System (IDS) solution designed to detect DDoS attacks by integrating both rule-based detection and machine learning (ML) approaches. This system involves several key steps: data acquisition, pre-processing, feature selection, model training, attack detection, and control actions. Data acquisition includes data collection from the target CPPS using tools like tcp dump to capture network traffic, feature extraction using CIC-FLOWMETER to extract statistical features from the captured data, and data transmission of the extracted features for further processing. Pre-processing is performed to remove uncertainties from the data, including duplication removal, handling missing entries, noisy data handling, and labeling the data appropriately for ML model training. Feature selection involves selecting the most relevant features to improve classification accuracy. Model training uses various machine learning algorithms (both supervised and unsupervised) to train models that can differentiate between normal and malicious network flows. Attack detection involves using the trained ML models to detect attacks, complemented by rule-based detection to refine the ML results and reduce false positives. Finally, control actions are taken, which include generating alert messages with severity levels and providing recommended actions for attack mitigation, leveraging frameworks like MITRE to suggest appropriate responses.

**DIFFERENCE**

The primary difference between old and new Intrusion Detection Systems (IDS) lies in their detection methodologies. Old systems traditionally rely on signature-based or rule-based detection, using a database of attack signatures to identify malicious behavior. These systems detect intrusions by matching network traffic patterns against pre-defined signatures. They are effective against known attack patterns but struggle with new, sophisticated, and zero-day attacks. They also require frequent updates to their signature databases and lack the capability to learn and adapt to new attack patterns. In contrast, the new system employs a combination of rule-based and machine learning (ML) approaches to detect DDoS attacks. This involves data acquisition, pre-processing, feature selection, model training, attack detection, and control actions. The new system enhances detection by learning from data and adapting to new attack patterns, providing a more robust solution for dynamic environments like modern Cyber-Physical Production Systems (CPPS).

**ADVANTAGES**

The new system presents several advantages over traditional Intrusion Detection Systems (IDS). It enhances detection capability and improves the decision-making process by reducing false positives. The system provides an extra check on the predictions made by ML models to ensure the results give a complete picture of the network situation. The combination of rule-based and ML detection complements the detection capability of ML approaches against DDoS attacks. The new system uses real-time data for training and validation, ensuring its effectiveness in real-time scenarios. ML-based detection achieves detection at each flow level, while rule-based detection summarizes the detection results. The system also includes an alert system, providing unified alerts along with recommended suggestions for attack mitigation.

**DISADVANTAGES**

The document discusses a few disadvantages and challenges related to the new system. ML-based detection can generate a redundancy of alerts, potentially increasing false positives if not handled properly. The integration of rule-based detection is used to address this potential issue. Also, the complexity of CPPS infrastructure requires ML algorithms to be very precise, and these models need a large amount of clean, trustworthy data for training, which can be a challenge.

**CHAPTER 3**

**THEORETICAL BACKGROUND**

**3.1 IMPLEMENTATION ENVIRONMENT**

The implementation of the proposed Hierarchical ML-Based Hyperparameter Optimization Technique for Network Intrusion Detection requires a robust computing environment that efficiently handles data preprocessing, model training, and evaluation. The system is developed using Python 3.x as the primary programming language, with libraries such as Scikit-learn, XGBoost, LightGBM, CatBoost, and Optuna for machine learning and hyperparameter optimization. The CICIDS dataset is used for training and testing, requiring extensive preprocessing steps, including Min-Max Scaling for normalization, SMOTE for handling class imbalance, and feature selection techniques to remove redundant attributes. The implementation is carried out on a high-performance computing system with a minimum Intel Core i7 processor, 16GB RAM, and an NVIDIA GPU for accelerating model training when needed. The development environment is set up using Jupyter Notebook, VS Code, or PyCharm, along with virtual environment management tools like Anaconda or virtualenv to ensure dependency isolation. The model training process follows a structured approach, utilizing hierarchical hyperparameter optimization techniques to fine-tune classifiers such as XGBoost, LightGBM, CatBoost, Random Forest, and Decision Tree, ensuring optimal performance. The evaluation of models is conducted using key performance metrics, including accuracy, precision, recall, and F1-score, to assess their effectiveness in detecting network intrusions. With this carefully structured implementation environment, the system is designed to efficiently handle large-scale intrusion detection tasks, improve classification accuracy, and enhance overall network security.

**3.2 SYSTEM ARCHITECTURE**

The proposed Hierarchical ML-Based Hyperparameter Optimization Technique for Network Intrusion Detection follows a well-structured system architecture to ensure efficient data processing, model training, and evaluation. The architecture is designed in multiple stages, including data preprocessing, feature selection, model training with hyperparameter tuning, and performance evaluation. The system consists of the following key components:

Data Acquisition Layer – The CICIDS dataset is used as the primary data source, containing network traffic records labeled as normal or malicious. This data is preprocessed to remove inconsistencies and noise.

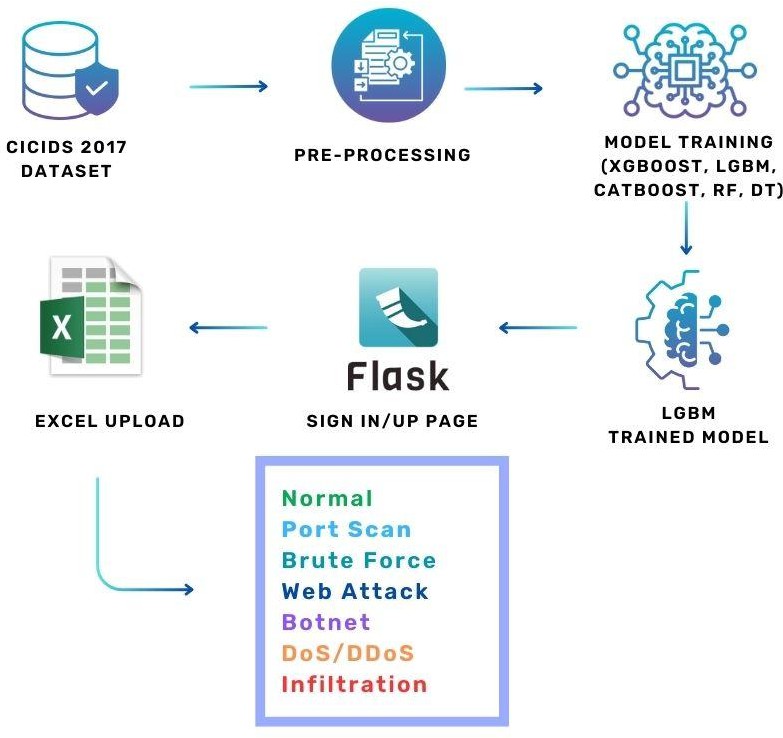
Preprocessing and Feature Selection Layer – Raw data undergoes transformation, including Min-Max Scaling for normalization and SMOTE for handling class imbalance. Feature selection techniques, such as correlation-based filtering and recursive feature elimination (RFE), are applied to reduce dimensionality and improve model efficiency.

Machine Learning Model Layer – The optimized dataset is fed into multiple machine learning classifiers, including XGBoost, LightGBM, CatBoost, Random Forest, and Decision Tree, which are selected based on their effectiveness in intrusion detection.

Hyperparameter Optimization Layer – A hierarchical hyperparameter tuning approach is applied to find the optimal parameters for each classifier, enhancing performance while reducing computational costs. Techniques such as GridSearchCV, Bayesian Optimization, and Optuna are used for systematic tuning.

Evaluation and Performance Metrics Layer – The trained models are evaluated based on accuracy, precision, recall, and F1-score to determine their effectiveness in detecting intrusions. The best-performing model is selected for deployment.

Deployment and Real-Time Monitoring Layer – The optimized model can be integrated into a real-world intrusion detection system, monitoring live network traffic and classifying threats in real-time.



**Fig 3.1**

* 1. **PROPOSED METHODOLOGY**

The proposed methodology focuses on developing an efficient and optimized machine learning-based intrusion detection system using a Hierarchical ML-Based Hyperparameter Optimization Technique. This approach systematically enhances model performance by integrating preprocessing, feature selection, ensemble learning, and hyperparameter tuning to improve network intrusion detection accuracy while minimizing false positives.

**Data Preprocessing**

The system utilizes the CICIDS dataset, which contains labeled network traffic records.   
• Min-Max Scaling: Normalizes feature values to a uniform range, improving model convergence.  
• SMOTE (Synthetic Minority Over-sampling Technique): Addresses class imbalance by generating synthetic samples for underrepresented attack categories.  
• Feature Selection: Eliminates irrelevant and redundant features using correlation analysis and Recursive Feature Elimination (RFE) to improve model efficiency.

**Machine Learning Model Selection**

The preprocessed dataset is used to train multiple machine learning classifiers known for their effectiveness in intrusion detection. The selected models include:

• XGBoost (Extreme Gradient Boosting): A powerful boosting algorithm that enhances classification accuracy.

• LightGBM (Light Gradient Boosting Machine): Optimized for large-scale datasets with fast training speed.

• CatBoost (Categorical Boosting): Efficient in handling categorical features and reducing overfitting.

**Hierarchical Hyperparameter Optimization**

A hierarchical optimization approach is applied to fine-tune the hyperparameters of each classifier. This multi-step process ensures efficient parameter selection without excessive computational overhead.

* GridSearchCV & Randomized Search: Used for initial tuning to find the most promising parameter ranges.
* Optimization & Optuna: Applied for fine-tuning, reducing the need for exhaustive searches while improving model performance.

**Model Training and Evaluation**

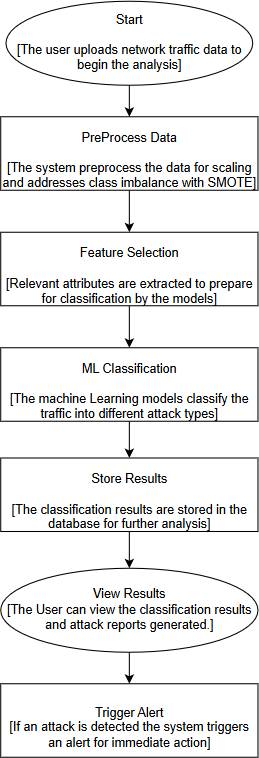
Each model is trained on the optimized dataset and evaluated using key performance metrics:

* Accuracy: Measures the overall correctness of predictions.
* Precision & Recall: Evaluates the model’s ability to correctly classify intrusions while minimizing false alarms.

F1-Score: Ensures a balance between precision and recall, crucial for intrusion detection.

# 3.3.1 SEQUENCE DIAGRAM:

A Sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of Message Sequence diagrams are sometimes called event diagrams, event sceneries and timing diagram.



**Fig 3.2**

# USE CASE DIAGRAM:

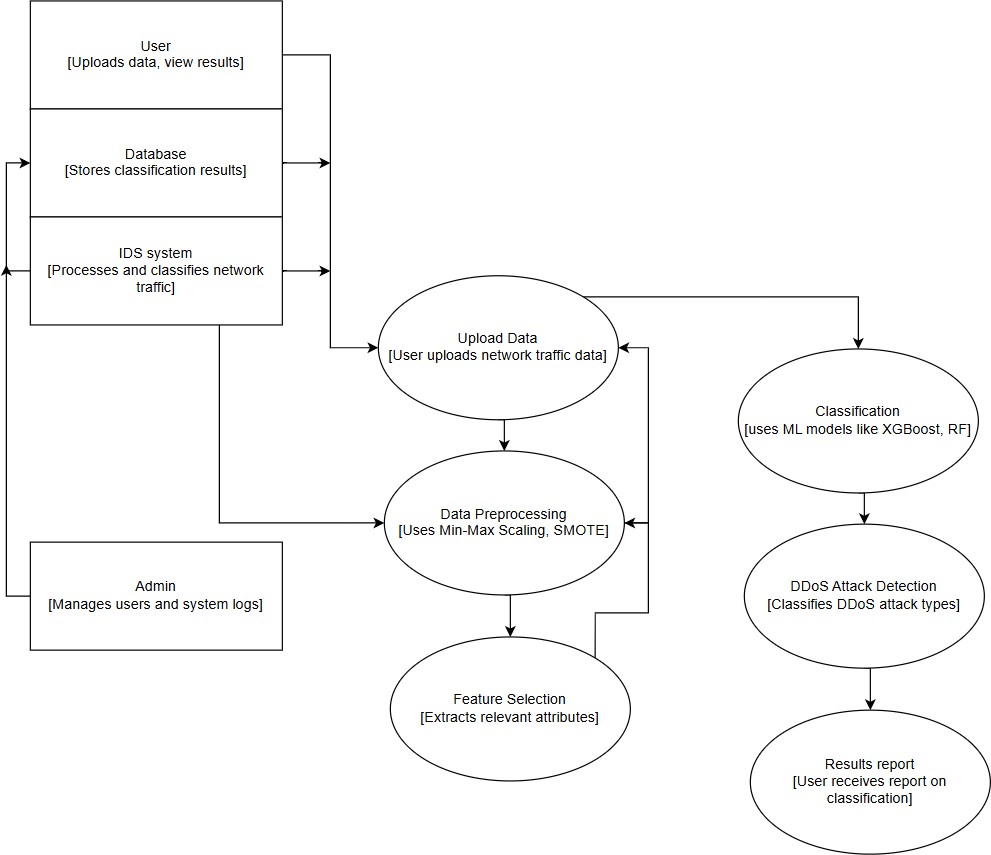
Unified Modeling Language (UML) is a standardized modeling language for software engineering, managed by the Object Management Group. It uses graphic notation to visualize, specify, modify, construct, and document object- oriented software systems.

A Use case Diagram is used to present a graphical overview of the functionality provided by a system in terms of actors, their goals and any dependencies between those use cases.

Use case diagram consists of two parts:

**Use case:** A sequence of actions that provide measurable value to an actor, represented as a horizontal ellipse.

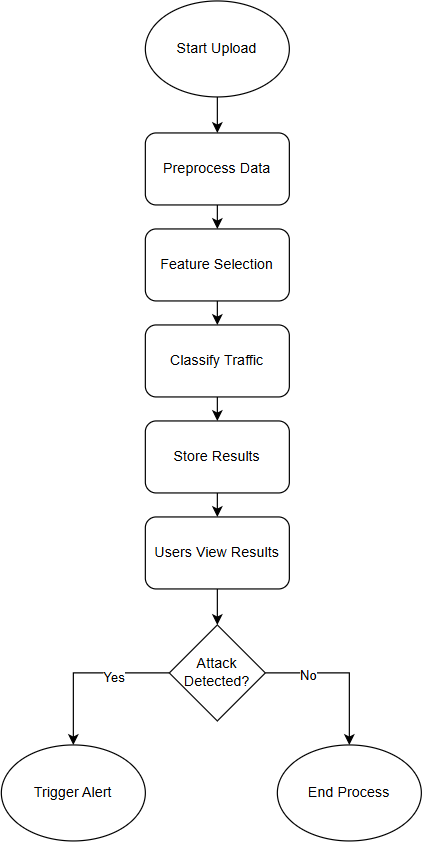
**Actor:** A person, organization, or system interacting with the system.



**Fig 3.3**

# ACTIVITY DIAGRAM:

Activity diagram is a graphical representation of workflows of stepwise activities and actions with support for choice, iteration and concurrency. An activity diagram shows the overall flow of control.



**Fig 3.4**

# COLLABORATION DIAGRAM:

UML Collaboration Diagrams illustrate the relationship and interaction between software objects. They require use cases, system operation contracts and domain model to already exist. The collaboration diagram illustrates messages being sent between classes and objects.

A diagram of a software development

AI-generated content may be incorrect.

**Fig 3.5**

**CHAPTER 4**

**SYSTEM IMPLEMENTATION**

**CODING**

Once the design aspect of the system finalizes the system enters into the coding and testing phase. The coding phase brings the actual system into action by converting the design of the system into the code in each programming language. Therefore, a good coding style must be taken whenever changes are required it easily screwed into the system.

# CODING STANDARDS

Coding standards are essential guidelines for programming that emphasize the physical structure and appearance of code. These standards play a crucial role in ensuring that code is easy to read, comprehend, and maintain, regardless of the programmer involved. They provide a unified approach to coding practices, enhancing collaboration among developers and reducing the likelihood of errors or misinterpretations. By adhering to coding standards, programmers can create code that is not only functional but also streamlined and efficient.

In the development process, coding standards are particularly significant during the coding phase, as this phase translates the blueprint developed during the design phase into actionable code. The coding specifications should be designed in such a way that any programmer, whether familiar with the original design or not, can easily interpret the code, make necessary modifications, or update the program as required.

# NAMING CONVENTIONS

Naming conventions of classes, data member, member functions, procedures etc., should be self-descriptive. One should even get the meaning and scope of the variable by its name. The conventions are adopted for easy understandingof the intended message by the user. So, it is customary to follow the conventions. These conventions are as follows:

# CLASS NAMES

Class names are problem domain equivalence and begin with capital letter and have mixed cases.

# MEMBER FUNCTION AND DATA MEMBER NAME

Member function and data member name begins with a lowercase letter with each subsequent letters of the new words in uppercase and the rest of letters in lowercase.

# VALUE CONVENTIONS

Value conventions play a critical role in programming by ensuring that variables hold appropriate and accurate values at any given point during the execution of a program. These conventions are essential for maintaining the integrity and reliability of the system's functionality. By adhering to value conventions, developers can prevent unexpected errors, improve overall program efficiency, and create code that is more robust and consistent. This involves the following:

* Proper default values for the variables.
* Proper validation of values in the field.

# SCRIPT WRITING AND COMMENTING STANDARD

Script writing is an art in which indentation is utmost important. Conditional and looping statements are to be properly aligned to facilitate easy understanding. Comments are included to minimize the number of surprises that could occur when going through the code.

# MESSAGE BOX FORMAT

When something must be prompted to the user, he must be able to understand it properly. To achieve this, a specific format has been adopted in displaying messages to the user. They are as follows:

* X – User has performed illegal operation.
* ! – Information to the user.

# 4.1 TEST PROCEDURE

* + 1. **SYSTEM TESTING**

Testing is performed to identify errors. It is used for quality assurance. Testing is an integral part of the entire development and maintenance process. The goal of the testing during phase is to verify that the specification has been accurately and completely incorporated into the design, as well as to ensure the correctness of the design itself. By performing rigorous testing, developers can guarantee a seamless user experience and minimize the risk of failures in the future. This involves assessing whether the system's behavior matches its intended purpose, identifying discrepancies, and addressing them before deployment.

For example, the design must not have any logic faults in the design is detected before coding commences, otherwise the cost of fixing the faults will be considerably higher as reflected. Detection of design faults can be achieved by means of inspection as well as walkthrough.

Testing is one of the important steps in the software development phase. Testing checks for the errors, of the project testing involves the following test cases:

* Static analysis is used to investigate the structural properties of the Source code.
* Dynamic testing is used to investigate the behavior of the source code by executing the program on the test data.

# 

# TEST DATA AND OUTPUT

**4.2.1 UNIT TESTING**

Unit testing is conducted to verify the functional performance of each modular component of the software. Unit testing focuses on the smallest unit of the software design (i.e.), the module. The white box testing techniques were heavily employed for unit testing.

# FUNCTIONAL TESTS

Functional test cases are essential for verifying the correctness and reliability of a software system by systematically testing its behavior under various input conditions. These cases focus on ensuring that the system delivers expected results and handles data consistently, even in edge cases or with special inputs. The process involves carefully designed scenarios that cover a broad spectrum of inputs to validate the system's functionality.

Three types of tests in Functional test:

* Performance Test
* Stress Test
* Structure Test

# PERFORMANCE TEST

It determines the amount of execution time spent in various parts of the unit, program throughput, and response time and device utilization by the program unit.

# STRESS TEST

Stress Test is that test designed to intentionally break the unit. A Great deal can be learned about the strength and limitations of a program by examining the way a programmer in which a program unit breaks.

# STRUCTURED TEST

Structure Tests are concerned with exercising the internal logic of a program and traversing execution paths. The way in which White-Box test strategy was employed to ensure that the test cases could Guarantee that all independent paths within a module have been exercised at least once.

* Exercise all logical decisions on their true or false sides.
* Execute all loops at their boundaries and within their operational bounds.
* Exercise internal data structures to assure their validity.
* Checking attributes for their correctness.
* Handling end of file condition, I/O errors, buffer problems and textual errors in output information

# INTEGRATION TESTING

Integration testing is a systematic technique for construction the program structure while at the same time conducting tests to uncover errors associated with interfacing. i.e., integration testing is the complete testing of the set of modules which makes up the product. The objective is to take untested modules and build a program structure tester should identify critical modules. Critical modules should be tested as early as possible. One approach is to wait until all the units have passed testing and then combine them and then tested. This approach has evolved from unstructured testing of small programs. This process goes beyond testing individual units and focuses on verifying the interactions between multiple modules that collectively make up the complete product. Another strategy is to construct the product in increments of tested units. A small set of modules are integrated together and tested, to which another module is added and tested in combination. And so on. The advantages of this approach are that, interface dispenses can be easily found and corrected.

The major error that was faced during the project is linking error. When all the modules are combined the link is not set properly with all support files.

Then we checked out for interconnection and the links. Errors are localized to the new module and its intercommunications. The product development can be staged, and modules integrated in as they complete unit testing. Testing is completed when the last module is integrated and tested.

# TESTING TECHNIQUES / TESTING STRATERGIES

* + 1. **TESTING**

Testing is a process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an as-yet – undiscovered error. A successful test is one that uncovers an as-yet- undiscovered error. System testing is the stage of implementation, which is aimed at ensuring that the system works accurately and efficiently as expected before live operation commences. It verifies that the whole set of programs hang together. System testing requires a test consists of several key activities and steps for run program, string, system and is important in adopting a successful new system. This is the last chance to detect and correct errors before the system is installed for user acceptance testing. Critical modules, which serve as foundational components of the system, are prioritized for testing. testing is a critical phase in the software development lifecycle. By systematically examining module interactions, it ensures that the final product functions smoothly and meets its intended specifications.

The software testing process commences once the program is created, and the documentation and related data structures are designed. Software testing is essential for correcting errors. Otherwise, the program or the project is not said to be complete. Software testing is the critical element of software quality

assurance and represents the ultimate the review of specification design and coding. Testing is the process of executing the program with the intent of finding the error. A good test case design is one that as a probability of finding an yet undiscovered error. A successful test is one that uncovers an yet undiscovered error. Any engineering product can be tested in one of the two ways:

# WHITE BOX TESTING

This testing is also called as Glass box testing. In this testing, by knowing the specific functions that a product has been designed to perform test can be conducted that demonstrate each function is fully operational at the same time searching for errors in each function. Unlike black box testing, which evaluates the functionality of a system without delving into its internal workings, glass box testing provides testers with an in-depth understanding of how the system operates under the hood. It is a test case design method that uses the control structure of the procedural design to derive test cases. Basis path testing is a white box testing.

# BLACK BOX TESTING

In this testing by knowing the internal operation of a product, test can be conducted to ensure that “all gears mesh”, that is the internal operation performs according to specification and all internal components have been adequately exercised. A significant focus of this testing methodology is on the functional requirements of the software. This means the testing process ensures that the software delivers the functionality it was designed to provide, adhering to the requirements and objectives established during the design and development phases. It fundamentally focuses on the functional requirements of the software.

# INTEGRATION TESTING

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with. Individual modules, which are highly prone to interface errors, should not be assumed to work instantly when we put them together. The problem of course, is “putting them together”- interfacing. There may be the chances of data lost across on another’s sub functions, when combined may not produce the desired major function; individually acceptable impression may be magnified to unacceptable levels; global data structures can present problems.

# PROGRAM TESTING:

The logical and syntax errors have been pointed out by program testing. A syntax error is an error in a program statement that in violates one or more rules of the language in which it is written. An improperly defined field dimension or omitted keywords are common syntax error. These errors are shown through error messages generated by the computer. A logic error on the other hand deals with the incorrect data fields, out-off-range items and invalid combinations. Since the compiler s will not deduct logical error, the programmer must examine the output. Condition testing exercises the logical conditions contained in a module. The possible types of elements in a condition include a Boolean operator, Boolean variable, a pair of Boolean parentheses A relational operator or on arithmetic expression. Condition testing method focuses on testing each condition in the program the purpose of condition test is to deduct not only Errors in the condition of a program but also other errors in the program.

# SECURITY TESTING

Security testing is a critical process in software and system development aimed at ensuring that the built-in protection mechanisms effectively safeguard the system from unauthorized access and potential vulnerabilities. This type of testing evaluates whether the system's security measures can withstand intentional efforts to breach them, identifying and addressing any weaknesses before they can be exploited. A comprehensive security test not only examines the system's defenses against direct or obvious ("frontal") attacks but also investigates its ability to resist indirect or subtle ("rear") attacks. Frontal attacks might involve attempts to bypass authentication mechanisms or exploit known vulnerabilities, while rear attacks could include more covert methods, such as exploiting hidden backdoors or misconfigurations.

# VALIDATION TESTING

Validation testing is a crucial step in the software development lifecycle aimed at ensuring the final product meets the requirements and expectations of the customer. It focuses on verifying that the software operates as intended and delivers the functionality it was designed for. This type of testing employs black box techniques, where the tester evaluates the external behavior of the software without delving into its internal workings. By conducting tests that simulate real-world usage, validation testing confirms that the software conforms to functional requirements, handles inputs and outputs correctly, and delivers reliable results. During security testing, the tester adopts the role of an individual attempting to penetrate the system, often referred to as an ethical hacker or penetration tester. This approach involves simulating real-world attack scenarios to test the system's resilience against threats. The tester employs various strategies to uncover vulnerabilities, such as exploiting weak passwords, probing for open ports, testing for injection flaws, and analyzing access control mechanisms. Upon successful validation, software is deemed .

# USER ACCEPTANCE TESTING

User acceptance is a critical factor that determines the success of any system. Ensuring that users are comfortable with and supportive of the system under consideration is an integral part of the development process. The system is rigorously tested for user acceptance by actively engaging with prospective users throughout its development cycle. To achieve user acceptance, the system under consideration is subjected to rigorous testing and evaluation throughout its development cycle. This is accomplished by actively engaging with prospective users and involving them in the process. By creating open and continuous communication channels, developers can gather invaluable feedback, understand user preferences, and address pain points. This user-centric interaction ensures that the system evolves in a way that resonates with its target audience. The ultimate goal is to develop a user-friendly, efficient, and reliable system that is intuitive to navigate and aligned with user needs. This involves establishing continuous communication channels to gather feedback, understand user preferences, and implement changes as required. The goal is to create a user-friendly and efficient system that meets the needs and expectations of its users.

* Input screen design - Input screens are where users interact with the system to provide necessary data. These are designed to be intuitive, user-friendly, and efficient, minimizing errors during data entry. A clear layout and proper labeling are prioritized to enhance usability.
* Output screen design - Output screens display the system's results or outcomes to the users. They focus on presenting data clearly and concisely, ensuring users can easily interpret and utilize the information provided. Proper formatting and readability are key elements of a well-designed output screen.

**CHAPTER 5**

**SYSTEM IMPLEMENTATION**

# 5.1 PERFORMANCE PARAMETERS

Performance parameters are critical tools used to evaluate the efficiency, functionality, and reliability of a system, process, or product across diverse domains. These metrics offer a quantifiable means of assessing whether a system meets its intended objectives and identifying opportunities for optimization. In software systems, performance parameters such as response time, throughput, scalability, and availability play a vital role in ensuring the system can handle user demands effectively while maintaining stability under varying workloads. Similarly, in networking, metrics such as bandwidth, latency, and packet loss determine the efficiency of data transmission and overall user experience. In business operations, parameters like efficiency, productivity, customer satisfaction, and resource utilization measure the success of organizational processes and highlight areas for enhancement. In manufacturing or industrial systems, key parameters such as cycle time, defect rate, and energy efficiency assess production capabilities and sustainability. Each context relies on performance parameters to gain actionable insights, improve processes, and deliver high-quality outcomes. These metrics also support informed decision-making by providing stakeholders with a clear understanding of system performance. By monitoring and analyzing performance parameters, organizations can ensure continuous improvement, adapt to evolving needs, and maintain competitiveness in their respective industries. Ultimately, performance parameters are indispensable for driving success and ensuring that systems, processes, and products operate optimally while meeting user and organizational expectations.

# 5.2 RESULTS AND DISCUSSION

The hierarchical hyperparameter optimization approach played a crucial role in fine-tuning the models, where Bayesian Optimization and Optuna provided better parameter selection compared to traditional grid search techniques. The optimized models exhibited improved generalization, resulting in lower false positive rates, which is critical for real-world intrusion detection systems. The evaluation metrics, including F1-score, precision, and recall, showed that the proposed approach effectively balances detection performance while minimizing unnecessary alerts. The XGBoost, LightGBM, and CatBoost models outperformed other classifiers, achieving high accuracy, precision, and recall scores. Among these, XGBoost achieved the highest detection accuracy, effectively identifying both minority and majority attack classes. Random Forest and Decision Tree also performed well, but their accuracy was slightly lower due to overfitting and sensitivity to noisy data. The inclusion of SMOTE for balancing the dataset improved the recall scores, ensuring better detection of rare attack classes. Additionally, feature selection methods helped reduce dimensionality, leading to faster model training and improved efficiency. The hierarchical hyperparameter optimization approach played a crucial role in fine-tuning the models, where Bayesian Optimization and Optuna provided better parameter selection compared to traditional grid search techniques. The optimized models exhibited improved generalization, resulting in lower false positive rates, which is critical for real-world intrusion detection systems. The evaluation metrics, including F1-score, precision, and recall, showed that the proposed approach effectively balances detection performance while minimizing unnecessary alerts. Overall, the results confirm that integrating preprocessing, feature selection, ensemble learning, and hierarchical hyperparameter tuning leads to a more robust and scalable intrusion detection system. This approach not only improves accuracy but also enhances computational efficiency, making it a viable solution for real-time network security applications.

**CHAPTER 6**

**CONCLUSION & FUTURE WORK**

The proposed Hierarchical ML-Based Hyperparameter Optimization Technique for network intrusion detection effectively enhances classification accuracy and reduces false positives by integrating data preprocessing, feature selection, ensemble learning, and advanced hyperparameter tuning. The results demonstrate that models such as XGBoost, LightGBM, and CatBoost outperform traditional classifiers, achieving high accuracy, recall, and F1-score while efficiently handling class imbalance using SMOTE. The use of hierarchical hyperparameter optimization ensures optimal model performance by systematically tuning parameters, reducing computational overhead, and improving detection efficiency. Overall, the proposed approach provides a robust, scalable, and efficient intrusion detection system capable of identifying cyber threats with high precision.For future work, the system can be enhanced by incorporating deep learning techniques such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs) to further improve feature extraction and classification performance. Additionally, real-time deployment of the model in an active network environment can be explored to assess its practical applicability. Another potential improvement includes adversarial learning techniques to make the IDS resilient against evolving cyber threats. Lastly, optimizing computational efficiency by leveraging cloud-based or edge computing environments can be investigated to ensure faster and more scalable intrusion detection for large-scale network infrastructures.

**APPENDICES**

**A.1 SGD GOALS**

In the proposed Hierarchical ML-Based Hyperparameter Optimization Technique for Network Intrusion Detection, Stochastic Gradient Descent (SGD) plays a crucial role in optimizing machine learning models, improving their efficiency, and ensuring better classification of network intrusions. The primary goal of SGD in this project is to enhance the training process of models like XGBoost, LightGBM, CatBoost, Random Forest, and Decision Tree by efficiently minimizing the loss function and optimizing hyperparameters. Stochastic Gradient Descent (SGD) is an optimization algorithm widely used in machine learning and deep learning models to minimize the loss function and improve model performance. The primary goal of SGD is to find the optimal set of parameters that minimize the error in predictive models by iteratively adjusting weights based on the gradient of the loss function. Unlike traditional gradient descent, which computes the gradient using the entire dataset, SGD updates the model parameters using only a single or a small batch of training samples per iteration, making it computationally efficient and suitable for large datasets.

**A.2 SOURCE CODE**

**A.3 SCREEN SHOTS**

**A.4 PLAGIRISM REPORT**

**A.5 PAPER PUBLICATION**

**REFERENCES**

1.G. Karpagam, B. V. Kumar, J. U. Maheswari and X.-Z. Gao, Smart Cyber Physical Systems, New York, NY, USA:CRC Press, 2020.

2.H. C. Verma, S. Srivastava, T. Ahmed and N. A. Usmani, "Cyber threats in agriculture and the food industry: An Indian perspective" in Advances in Cyberology and the Advent of the Next-Gen Information Revolution, Hershey, PA, USA:IGI Global, pp. 109-122, 2023.

3.X. Koufteros and G. Lu, "Food supply chain safety and security: A concern of global importance", J. Marketing Channels, vol. 24, no. 3, pp. 111-114, 2017.

4. A. Yeboah-Ofori, S. Islam, S. W. Lee, Z. U. Shamszaman, K. Muhammad, M. Altaf, et al., "Cyber threat predictive analytics for improving cyber supply chain security", IEEE Access, vol. 9, pp. 94318-94337, 2021.

5. D. E. Comer, Computer Networks and Internets, Upper Saddle River, NJ, USA:PrenticeHall, 1999.

6.J. David and C. Thomas, "DDoS attack detection using fast entropy approach on flow- based network traffic", Proc. Comput. Sci., vol. 50, pp. 30-36, Jan. 2015.

7.F. B. Saghezchi, G. Mantas, M. A. Violas, A. M. de Oliveira Duarte and J. Rodriguez, "Machine learning for DDoS attack detection in industry 4.0 CPPSs", Electronics, vol. 11, no. 4, pp. 602, Feb. 2022.

8.I. A. Khan, N. Moustafa, D. Pi, Y. Hussain and N. A. Khan, "DFF-SC4N: A deep federated defence framework for protecting supply chain 4.0 networks", IEEE Trans. Ind. Informat., vol. 19, no. 3, pp. 3300-3309, Mar. 2023.

9. K. Uszko, M. Kasprzyk, M. Natkaniec and P. Chołda, "Rule-based system with machine learning support for detecting anomalies in 5G WLANs", Electronics, vol. 12, no. 11, pp. 2355, May 2023.

10. R. R. R. Barbosa and A. Pras, "Intrusion detection in SCADA networks", Proc. 4th Int. Conf. Auton. Infrastruct. Manag. Secur. (AIMS), pp. 163-166, Jun. 2010.

11. A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras and B. Stiller, "An overview of IP flow-based intrusion detection", IEEE Commun. Surveys Tuts., vol. 12, no. 3, pp. 343-356, 3rd Quart. 2010.

12. P. Borges, B. Sousa, L. Ferreira, F. B. Saghezchi, G. Mantas, J. Ribeiro, et al., "Towards a hybrid intrusion detection system for Android-based PPDR terminals", Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM), pp. 1034-1039, May 2017.

13. P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández and E. Vázquez, "Anomalybased network intrusion detection: Techniques systems and challenges", Comput. Secur., vol. 28, pp. 18-28, Feb./Mar. 2009.

14. S. S. Abosuliman, "Deep learning techniques for securing cyber-physical systems in supply chain 4.0", Comput. Electr. Eng., vol. 107, Apr. 2023.

15. O. Linda, T. Vollmer and M. Manic, "Neural network based intrusion detection system for critical infrastructures", Proc. Int. Joint Conf. Neural Netw., pp. 1827-1834, Jun. 2009