National College of Ireland

Improved Machine Learning-Based Intrusion Detection Systems for Secure Network Virtualization

MSc Research Project
Cloud Computing

Albin Biju
Student ID: 23286199

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	……. Albin Biju……
Student ID:	…….. 23286199……..……
Programme:	…… Cloud Computing…… **Year:** …2024-2025...
Module:	……MSc Research Project…….……
Supervisor:	…… Sean Heeney ……..……
Submission Due Date:	……11-08-2025……..……
Project Title:	Improved Machine Learning-Based Intrusion Detection Systems for Secure Network Virtualization
Word Count:	……7219…… **Page Count**……20……..……..

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……Albin Biju……

Date: ……11-08-2025……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Improved Machine Learning-Based Intrusion Detection Systems for Secure Network Virtualization

Albin Biju
Student ID: 23286199

Abstract

The adoption of network virtualization and cloud computing has introduced new complexities and attack vectors, which demands more intelligent and scalable security solutions. In such cases, traditional Intrusion Detection Systems (IDS) often have struggle to provide effective protection. This research addresses this challenge by designing and developing an improved machine learning-based IDS with security for network virtualization in mind. The project is motivated by the need for security to be integrated into the system that not only attains high accuracy in detection but also provides good operational management and scalability under a cloud environment. Following through a quantitative experimental study the research laid the foundation on the CIC-IDS2017 dataset for training and evaluation on a battery of machine learning models that include Random Forest, Multi-Layer Perceptron (MLP), and Long Short-Term Memory (LSTM) networks. After a series of solid preprocessing, feature engineering, and hyperparameter tuning procedures, the Tuned Random Forest model emerged as a winner, attaining an accuracy of 98.47% and a strong weighted F1-score of 0.98 as per this research study. A core contribution of this work is an implementation of an end-to-end real-time IDS system that incorporates this highly accurate model and integrates it into one of the AWS cloud architectures. The system uses AWS Lambda for scalable, serverless inference; Amazon CloudWatch for rich monitoring with custom metrics and automated alarms; and Amazon Security Lake for standardized long-term event logging in the OCSF-open cybersecurity schema framework format. Real-time visualization and administrative control of the complete system are done through a web dashboard based on Flask. The evaluation establishes the efficacy of the ML model and the successful integration of the cloud components, establishing a scalable, manageable, and highly visible security solution apposite for modern virtualized networks.

1 Introduction

The current state of digital transformation has made network virtualization and cloud computing the important underpinnings of modern IT infrastructures. Technologies such as Software-Defined Networking and Network Functions Virtualization (NFV), which define new paradigms of flexible, scalable, and efficient (Mustafa et al., 2024) networking, however present a more extensive, dynamic attack surface that complicates existing security models. As critical workloads move into virtualized environments, the demand for robust advanced adaptive and scalable security mechanisms intensifies (Alanazi, 2023). Ordinarily, the

traditional signature-based Intrusion Detection Systems (IDS) are unfit for handling the encrypted traffic, the ephemeral infrastructure of virtual components, and the sophisticated zero-day attacks profound in such ecosystems.

This gap has propelled the uptake of Machine Learning and Deep Learning techniques in the domain of cybersecurity. Such IDS finds complex patterns in network data through learning and discovers anomalous behaviors which point to likely new threats. Thus, the system would be able to present itself over static rule-based systems (Halbouni et al., 2022; Jafari Gohari et al., 2023) using data streams. Although they benefit from improved accuracy for intrusion detection using ML models, such models are often found to differ much from the theoretical performance to practical operational deployment. An effective IDS in a virtualized area needs to be accurate and scaled, manageable, and seamlessly integrated with its underlying cloud infrastructure (Srivastava et al., 2022).

This research project thus moves toward addressing this gap through a departure from model-centered evaluation to design, implementation and assessment of a wide all-inclusive cloud-native IDS framework. The independent research question that drives this study is:

How can a machine learning-based Intrusion Detection System be enhanced with cloud-native services to improve detection accuracy, scalability, and operational visibility in a secure network virtualization environment?

To answer this question, the following research objectives were established:

1. To develop and rigorously evaluate a range of machine learning models using the comprehensive CIC-IDS2017 dataset to identify the most effective classifier for network intrusion detection.
2. To design and implement a multi-threaded, real-time IDS architecture capable of integrating the best-performing ML model for live traffic analysis.
3. To build an extensive cloud monitoring and analytics solution using AWS CloudWatch, that have custom metrics, automated alarms, and a dynamic visualization dashboard.
4. To integrate AWS Lambda to create a scalable, serverless inference pipeline, enabling threat detection to be offloaded and managed efficiently.
5. To develop a data ingestion mechanism for a security data lake, formatting events into the Open Cybersecurity Schema Framework (OCSF) for standardized logging and analysis.
6. To build a web-based dashboard that provides real-time monitoring of detection events and operational control of the IDS.

This project's primary contribution is the demonstration of a fully integrated, end-to-end system that combines a high-accuracy ML model with a robust suite of cloud services. The implemented solution, detailed in this report, showcases how AWS services can be leveraged not merely as hosting platforms but as integral components that enhance the functionality, scalability, and manageability of a modern IDS.

The remainder of this report is structured as follows. Section 2 reviews the related academic literature on ML-based IDS and security in virtualized networks. Section 3 details the research methodology, covering dataset preparation, model development, and the evaluation framework. Section 4 presents the design specification and architecture of the proposed system. Section 5 describes the implementation of the core IDS engine, cloud integrations,

and web dashboard. Section 6 offers a full evaluation of the model's effectiveness and the integrated operation of the system. Lastly, Section 7 wraps up the report, relevant findings, extraction of limitations, and recommendations of future work.

2 Related Work

This section places the research in the context of existing academic literature, offering a critical review of work done in machine learning for intrusion detection, security for virtualization, and the role of cloud technologies in cyber security.

2.1 Machine Learning and Deep Learning in Intrusion Detection

Machine learning-based intrusion detection has been an active research area for many years. In the early days, standard algorithms like Decision Trees, Support Vector Machines (SVMs), and K-Nearest Neighbors were mostly used. Ensemble methods and deep learning, as a more recent trend, have shown their superiority in performance. A thorough survey by (Halbouni et al., 2022) gives an extensive overview of these approaches and warrants more complex models that can capture complex, non-linear relationships in network traffic data. Their real strength lies in rigorous comparisons among different algorithms using standard datasets; however, many of them deal solely with offline metrics of accuracy while neglecting aspects of real implementation and operationalization challenges. A special promise in the development has been given to Deep Learning (DL). Authors like (Kimanzi et al., 2024) and (Idris, 2025) review various DL architectures for intrusion detection, including Convolutional Neural Networks (CNN) for spatial feature extraction and Recurrent Neural Networks (RNN) or Long Short-Term Memory (LSTM) networks for capturing temporal dependencies in traffic flows. (Al-Haija and Droos, 2025) have considered specifically the application of DL to IoT environments, which share some characteristics with virtualized networks such as resource constraints and a large number of inter-connected devices. These DL models can achieve substantially higher accuracy, but on the edge of that comes computational complexities, extensive training data requirements, and interpretability because of the nature of "black box" operations in its decision-making process. This project recognizes these advanced techniques but is opting to evaluate them in conjunction with more traditional yet highly effective and interpretable models, such as Random Forest.

2.2 Securing Virtualized and Software-Defined Networks

There exist security solutions that are unique for a virtualized environment, SDN as well as NFV. According to (Kumar and Alqahtani, 2023), machine learning applications in the use of IDS-infused SDN study with particular regard to the centralized control plane of an SDN, which provides a powerful point of monitoring, but it then presents both the control and the single point of failure in the SDN paradigm. Their work discusses how necessary an IDS is not just effective, but also lightweight, such that it does not become a bottleneck in the centralized controller.

(Zehra et al., 2023) provide a complete survey on Anomaly Detection in NFV environments, where they point out the dynamic lifecycle of Virtual Network Functions (VNFs), which can be spun up, scaled, and torn down in minutes. The existing static security configurations and traditional monitoring approaches are rendered obsolete by such defining dynamicity, which

is a key focus in the research, where a flexible cloud-integrated architecture is used to address this challenge. (Mustafa et al., 2024) enhance this by exploring ML techniques for IDS in SDN, noting that ensemble methods like Random Forest provide a good balance between accuracy and computational cost, which is in abstract agreement with the findings of this project. This is a limitation in much of this literature since it relies on simulated SDN environments with few examples of fully implemented systems integrated with commercial cloud platforms like AWS.

2.3 Cloud-Native Security and Emerging Trends

It is important to consider in our modern approach to cybersecurity, the links between security tools and cloud platforms. (Srivastava et al., 2022) survey ML-based IDS on the cloud, emphasizing the advantages of scalability, centralized management, and availability of powerful computing resources. (Saran et al., 2022) continue this discussion with a broader grouping of security applications in the cloud, again, reaffirming the notion that the cloud is not just a hosting site for applications but a host of powerful tools for securing the application. This work builds squarely on that platform, with the building blocks of the IDS being AWS Lambda, CloudWatch and S3.

Emerging trends are also pushing the boundaries of what is possible. Federated Learning, as reviewed by (Hakeem and Kim, 2025) for V2X networks, offers a privacy-preserving way to train models without centralizing sensitive data. Reinforcement Learning has been explored for creating adaptive defense mechanisms that can respond to attacks in real-time (Kheddar et al., 2024). While these advanced patterns are beyond the scope of this project's implementation, they inform the future work and highlight the faster evolution of the field. In summary, the literature confirms the effectiveness of ML for intrusion detection and recognizes its distinctive security challenges of virtualized networks. However, the main portion of the research remains focused on model-level performance, that is often in simulated settings. There is a clear need for more research demonstrating the design, implementation, and evaluation of a complete, end-to-end system that integrates a high-performance ML model with the operational and scaling capabilities of a major cloud provider. This project's main idea is to fill that gap by presenting a practical and robust solution that is not only academically sound in its modeling approach but also operationally viable in a real-world cloud environment.

It is essential to compare its outcomes with those of existing works in the literature, inorder to contextualize the performance and contributions of this research. While a direct one on one comparison of accuracy can be a challenge due to variations in datasets, preprocessing techniques, and evaluation environments, a high-level analysis highlights the uniqueness of this project.

Table 1 presents a comparative overview of this research against key studies reviewed above..

Table 1: Comparative Analysis of Research Metrics and Contributions

Study (Author, Year)	Model(s) / Technique(s)	Dataset(s) Used	Key Metric (Accuracy %)	Key Contribution / Focus

This Research	**Tuned Random Forest, MLP, LSTM**	**CIC-IDS2017 (Balanced)**	**98.47%**	**End-to-end system deployment** with integrated AWS cloud services (Lambda, CloudWatch, Security Lake, OCSF). Focus on operational readiness.
Mustafa et al. (2024)	Ensemble Methods, Decision Trees	Various SDN datasets (incl. CIC-IDS2017)	Reports high accuracies (typically >97%)	Comprehensive study of ML model performance specifically for **Software-Defined Networking (SDN) environments**.
Halbouni et al. (2022)	Review of ML & DL models	Reviews studies using KDD99, NSL-KDD, CIC-IDS2017	Summarizes accuracies often in the 98-99%+ range for modern models.	A broad **literature review** of algorithms, comparing their theoretical performance without a specific implementation.
Kimanzi et al. (2024)	Review of Deep Learning (CNN, RNN, etc.)	Reviews studies using modern datasets	DL models often achieve >99% on specific tasks and datasets.	Focused review on the application and performance of various **deep learning architectures** for intrusion detection.
Kumar & Alqahtani (2023)	Various ML models	SDN-specific datasets	Reports high accuracies, emphasizing the need for lightweight models.	Focus on the challenges and future directions for ML-based IDS in **SDN**, highlighting the trade-off between performance and overhead.
Srivastava et al. (2022)	Review of ML-based IDS	Various, including cloud-centric traffic	N/A (Review paper)	Discusses the benefits and challenges of deploying IDS **on the cloud**, but primarily from a theoretical/survey perspective.

The comparison analysis in Table 4 shows several key insights. First of all, the 98.47% accuracy attained in this project's 'Tuned Random Forest model' is highly competitive and set with the state-of-the-art performance which was reported in major review papers (Halbouni et al., 2022; Kimanzi et al., 2024). This confirms the robustness of the machine learning pipeline built in this research, from data balancing to hyperparameter tuning. More importantly, the "Key Contribution / Focus" column area showcases the primary differentiator of this work. But many studies, such as (Mustafa et al., 2024) and (Kumar & Alqahtani, 2023), provide excellent analyses of model performance in specific conditions like SDN,they often remain at the level of simulation or model-centric evaluation. Like that

review papers like (Srivastava et al., 2022) describes the need for cloud-integrated IDS but do not present a functional implementation.

This research bridges that critical gap. Its core contribution is not merely achieving a high accuracy score but demonstrating the design, implementation, and functional validation of a **complete, operational system**. The integration of AWS Lambda for serverless inference, the creation of a comprehensive CloudWatch monitoring dashboard with automated alarms, and the ingestion of standardized OCSF events into a security data lake are practical, value-added features that are often discussed in theory but seldom implemented in a single, cohesive academic project. Therefore, this work successfully extends the state of the art from model-centric analysis to a holistic, cloud-native system deployment, providing a more complete blueprint for a production-ready IDS in a virtualized environment.

3 Research Methodology

This research utilizes a quantitative, experimental methodology structured into three different phases such as dataset preparation and analysis, model development and evaluation, and system implementation and testing. It ensures a reproducible process, from basic data handling to evaluating the final integrated system.

3.1 Phase 1: Dataset Preparation and Analysis

The foundation of any machine learning project is the quality and relevance of its dataset. This phase focused on preparing a suitable dataset for training and evaluating the intrusion detection models.

Dataset Selection: The CIC-IDS2017 dataset has been chosen for this research. Due to its realism and comprehensiveness, a wide variety of academic disciplines recognize and use this dataset. It contains benign traffic and a diverse range of common, up-to-date attack scenarios, captured over five days. The data includes full packet captures and 80 extracted network traffic features calculated using the CICFlowMeter tool, which closely mirror the features that can be derived from real-time network flow analysis. Its inclusion of varied and modern attacks, such as Brute Force, DoS, DDoS, Botnet, and Web Attacks, makes it highly suitable for training a robust and generalizable IDS.

Data Preprocessing and Balancing: The raw dataset, consisting of multiple large files, required significant preprocessing. As detailed in the balanced-dataset-generation.ipynb notebook, the initial steps involved:

1. Loading the data from Parquet files for efficiency.
2. Cleaning the data by handling or removing infinite and Not-a-Number (NaN) values.
3. Removing columns with zero or very low variance, as these provide no discriminatory information for the model.
4. Addressing class imbalance. The original dataset is highly imbalanced, with benign traffic and certain DoS attacks heavily outnumbering other classes. To prevent the models from being biased towards the majority classes, a balanced dataset was created. The top five most-represented classes with sufficient samples (DoS, Benign, Bruteforce, DDoS, Infiltration) were selected, and a subset of 10,000 samples from

each was randomly chosen, resulting in a perfectly balanced 5-class dataset of 50,000 samples. This ensures that the model gives equal importance to each threat type during training.

3.2 Phase 2: Model Development and Evaluation

This phase covered the end-to-end machine learning pipeline, from feature engineering to model selection and validation, as implemented in the modelling.ipynb notebook.

Feature Engineering:

- **Label Encoding:** The categorical target variable ('Label') was converted into numerical format using Scikit-learn's LabelEncoder. This is a necessary step for most ML algorithms.
- **Feature Scaling:** All numerical features were scaled using StandardScaler. This process standardizes features by removing the mean and scaling to unit variance. It is crucial for algorithms that are sensitive to the scale of input features, such as MLPs and LSTMs, and helps improve model convergence and performance.

Dimensionality Reduction:

- To decrease dimensionality in the feature space Principal Component Analysis (PCA) was applied. 95% of the variance was maintained with around 21 components as per the analysis. Training of the models was performed on both the full feature set and PCA-reduced feature set for evaluation of the trade-off against dimensionality, performance, and time for training.

Model Training and Selection: A diverse set of models was selected to compare different learning paradigms:

- **Random Forest (RF):** An ensemble technique known for accuracy, overfitting resistance, and handling high-dimensional data.
- **Multi-Layer Perceptron (MLP):** A well-established feedforward neural network and an early deep learning architecture.
- **Long Short-Term Memory (LSTM):** A succession of recurrent neurons used to capture temporal stories but in this instance, aggregated on flow features.
- **Evaluation Metrics and Validation**: Each model was evaluated with a large range of metrics on performance:
    - **Accuracy:** The ratio of correctly classified instances.
    - **Precision:** The ability of the model not to label any negative sample as positive.
    - **Recall (Sensitivity):** The ability of the model to find all positive samples.
    - **F1-Score:** The average between Precision and Recall, which gives a single score to balance both.

To guarantee the validity of the results, the classification distributions of both the training and test sets were kept in the stratified train-test split of 80% for training and 20% for testing. In addition, to check for over-fitting and assess generalization performance of the models, 5-fold stratified cross-validation was performed on the training set.

Hyperparameter Tuning: For the best-performing model types (RF and MLP), GridSearchCV has been used to search for the optimal combination of hyperparameters, which enhances the model performance.

3.3 Phase 3: System Implementation and Testing

Work on the real-time IDS and its cloud integrations, followed by a sequence of tests to check whether the system works as intended was performed. System architecture and implementation were followed by an iterative development process, which built modular components for different sets of functionalities. The system employed a multi-threaded architecture design that set apart network data-collection tasks from the threat-detection activities while offering responsive and non-blocking operation. The implementation was based on Python 3 and several key libraries including boto3 to interact with AWS, psutil for system and network monitoring, Flask to build a web interface, and scikit-learn/tensorflow for executing models.

- **Testing and Validation:** A set of test scripts (test_enhancements.py, test_cloudwatch_metrics.py) were put together so that every element could be tested in order to confirm it is indeed functioning as expected.
- **Unit Test:** Test on OCSF formatting and feature calculation, which evaluates on finer grained, more independent level.
- **Integration Tests**: Verifying the good interaction between components, e.g. the ability of the IDS engine to invoke a Lambda function or post metrics to CloudWatch.
- **System Testing:** System Testing: the end-to-end workflow, from data collection to visualization on the web dashboard.

This structured phased approach guarantees the research is founded on a solid verifiable basis, as a result the final system formed as a well-grounded and properly validated one.

4 Design Specification

The design of the improved IDS focuses on creating a modular, scalable, and highly integrated system that leverages cloud-native services to enhance traditional ML-based detection. The architecture is composed of a core real-time detection engine, a set of cloud integration modules, and a web-based user interface for monitoring and control.

4.1 System Architecture

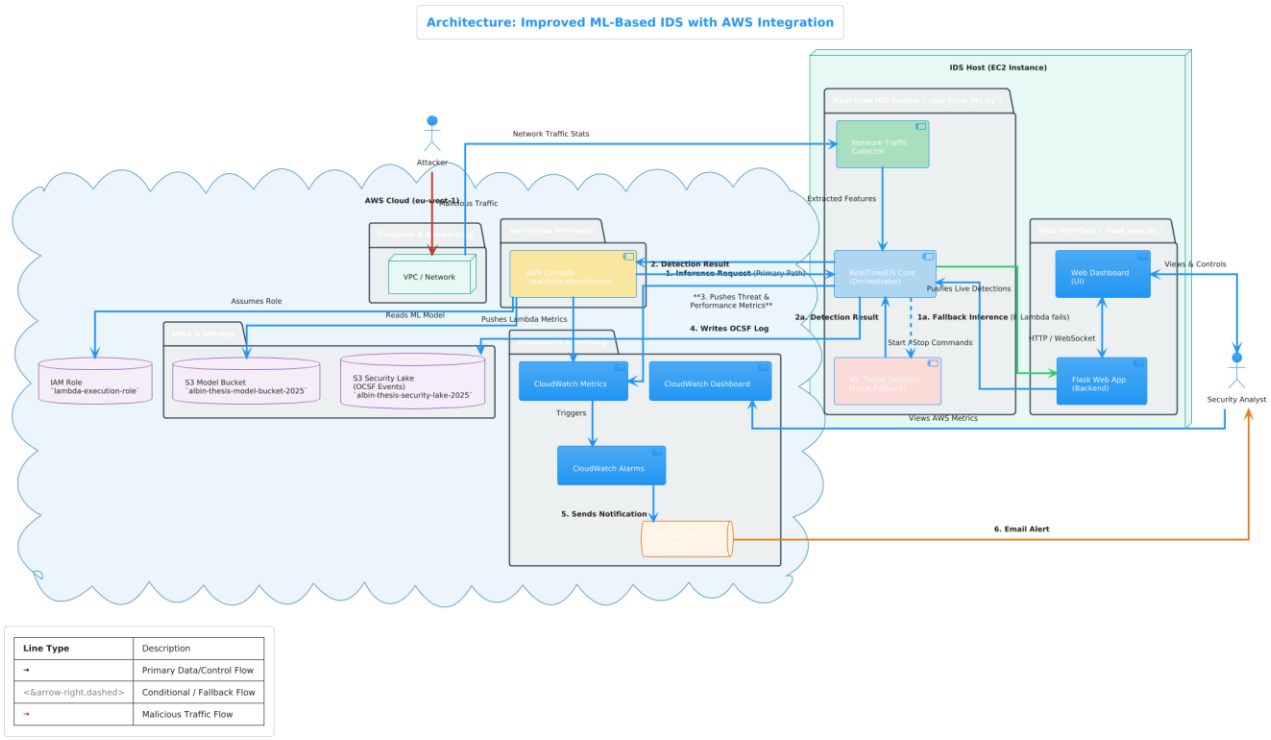

Figure 1: Architecture Diagram

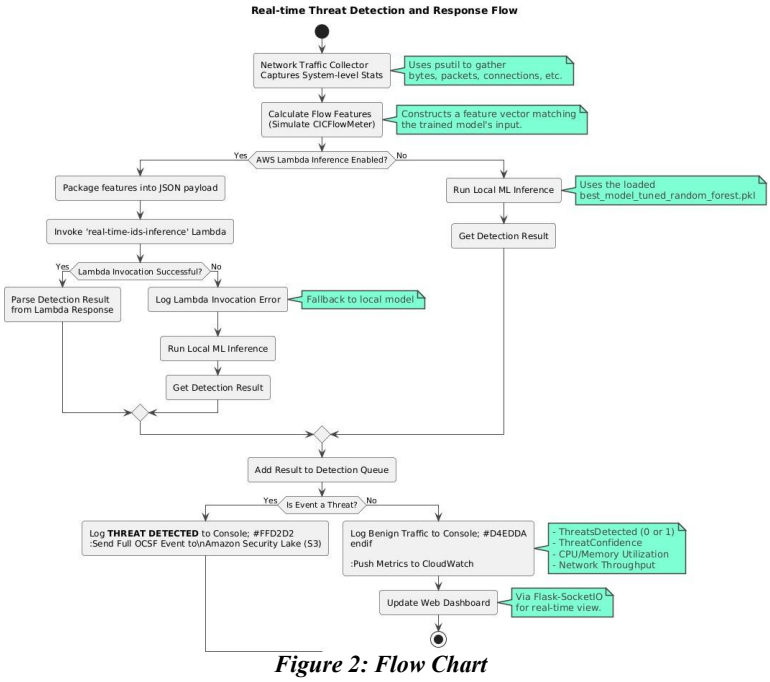

Figure 2: Flow Chart

The system follows a layered, distributed architecture designed for deployment within an AWS environment. Data and control flow logically as follows:

1. **Data Collection:** A collector on a host machine (e.g., an EC2 instance) continuously collects system-level network statistics.
2. **Feature Extraction:** The collected raw statistics are transformed into a feature vector that matches with the structure of the CIC-IDS2017 dataset, making it suitable with the trained ML model.

3. **Inference Pipeline:** The feature vector is sent for analysis through a dual-path inference pipeline. The primary path utilizes a scalable, serverless AWS Lambda function. A secondary, local inference path using the MLThreatDetector class serves as a resilient fallback.
4. **Detection and Logging:** The inference result (a classification label and confidence score) is processed.
5. **Data Persistence and Analytics:**
    - **Security Lake:** All significant detection events are formatted into the OCSF standard and ingested into a designated S3 bucket, which acts as a security data lake for long-term storage and compliance.
    - **CloudWatch Monitoring:** Key performance indicators (KPIs), threat metrics, and system health statistics are sent to Amazon CloudWatch as custom metrics.
    - **Alerting:** CloudWatch Alarms which are configured to monitor these metrics, trigger notifications via Amazon SNS if configured thresholds are exceeded.
6. **Visualization and Control:** A Flask-based web application provides a real-time dashboard to visualize detections and metrics, and offers API endpoints to start and stop the IDS service.

This architecture is designed to be scalable as well as decoupled. For instance, the inference workload can be handled by thousands of parallel Lambda invocations, and the CloudWatch and S3 services can handle virtually limitless data ingestion.

4.2 Core Component Design

The system is broken down into several key Python classes, each with a specific responsibility.

NetworkTrafficCollector (NetworkTrafficCollector class):

- **Functionality:** This component is responsible for collecting near real-time network statistics. It uses the psutil library, a cross-platform solution for retrieving information on running processes and system utilization.
- **Data Points:** It records metrics such as bytes sent/received, packets sent/received, error counts, and the number of active network connections.
- **Feature Calculation:** The ability to calculate derived metrics (e.g., bytes per second, packets per second). over time intervals is a key design feature. These calculated values are used to construct the feature vector for the ML model, simulating a network flow analysis process.
- **Data Structure:** It uses a deque (a double-ended queue) with a fixed maximum length to efficiently store a sliding window of the most recent traffic records.

MLThreatDetector (MLThreatDetector class):

- **Functionality:** This class encapsulates the local machine learning inference logic. It acts as the fallback mechanism if the Lambda-based inference is disabled or fails.

- **Model Loading:** Upon initialization, it loads the trained ML model (best_model_tuned_random_forest.pkl), the StandardScaler object, and the LabelEncoder object from disk using joblib. This pre-loading ensures low latency during prediction.
- **Prediction Method:** The predict_threat method takes a feature vector, applies the same scaling transformation used during training, and passes the result to the model to get a prediction and associated confidence probabilities.

4.3 Cloud-Native Integration Design

The "improved" aspect of the IDS is realized through its deep integration with AWS services.

AWS Lambda for Serverless Inference:

- **Design:** The system is designed to offload the ML inference task to a Lambda function named real-time-ids-inference. This follows a serverless computing paradigm, eliminating the need to manage underlying server infrastructure for the prediction task.
- **Invocation:** The core IDS engine packages the feature vector into a JSON payload and invokes the Lambda function synchronously.
- **Resiliency:** The design includes a crucial fallback mechanism. The system automatically routes the inference request to the local MLThreatDetector by ensuring continuous operation, in case the Lambda invocation fails (due to network issues, permissions errors, or cold starts timing out).
- **Amazon CloudWatch for Monitoring and Alarms (EnhancedCloudWatchMonitoring class):**
- **Custom Metrics:** The system is designed to push a rich set of custom metrics to CloudWatch under a dedicated namespace (IDS/RealTime). These metrics go beyond simple threat counts and include:
    - ThreatsDetected: Count of malicious classifications.
    - ThreatConfidence: The model's confidence score for detections.
    - DetectionLatency: Time taken for the inference process.
    - NetworkThroughput and PacketRate: Network performance indicators.
    - CPUUtilization and MemoryUtilization: System health of the host running the IDS.
- **Alarms:** A set of comprehensive alarms are defined to monitor these metrics. For example, an alarm IDS-HighThreatRate triggers if the number of threats detected in a 5-minute window exceeds a threshold. Another, IDS-HighCPUUtilization, warns of potential performance degradation on the IDS host.
- **Dashboard:** A CloudWatch Dashboard, defined in the cloudwatch-dashboard.json file, has been made to provide a view of all system metrics and alarm statuses. This set up promotes Infrastructure as Code (IaC) principles.
- **Amazon Security Lake Integration (SecurityLakeIntegration class):**
- **Standardized Logging:** To make sure interoperability and enable long-term analysis, all detection events are formatted into the Open Cybersecurity Schema Framework (OCSF). The OCSFFormatter class is designed to convert a raw detection result into a rich, structured OCSF event.

- **Data Ingestion:** The SecurityLakeIntegration class is responsible for uploading these OCSF JSON objects to a designated S3 bucket.
- **Partitioning:** The S3 keys are structured with date-based partitioning (/region=.../accountId=.../eventDay=.../), which is a best practice for security data lakes as it dramatically improves query performance and reduces costs when using services like Amazon Athena.

4.4 Web Dashboard Design

The flask_app.py script describes the design of a user-facing web application for real-time monitoring and control.

- **Technology Stack:** For bidirectional communication with the web client, it uses Flask as the web framework and Flask-SocketIO for real-time.
- **Views:** The UI is designed with three distinct pages:
    1. A main IDS dashboard for live threat feeds and summary statistics.
    2. A CloudWatch dashboard page that embeds and visualizes the metrics from AWS.
- An AWS infrastructure page which displays the status of related cloud resources like EC2 instances and Lambda functions.
- **Real-Time Updates:** The design uses WebSockets to push updates (new detections, summary stats) from the server to all connected clients in real-time without the need for user to refresh the page.
- **API Control:** A simple REST API with endpoints like /api/start and /api/stop is added to allow for programmatic or user-driven control of the IDS service.

5 Implementation

This section details the practical implementation of the designed system, translating the architectural specifications into functional code and configured cloud services. The implementation followed a structured sequence, as chronicled in the project's log.txt, encompassing data modeling, core engine development, cloud service integration, and front-end interface creation.

5.1 Environment and Development Tools

The project was developed using Python 3.11. The core ML pipeline relied on pandas for data manipulation, scikit-learn for traditional machine learning models and preprocessing, and tensorflow/keras for the LSTM neural network. System-level monitoring was implemented with psutil, and all interactions with AWS were handled by the boto3 SDK. The web dashboard was built using the Flask and Flask-SocketIO libraries.

5.2 Machine Learning Pipeline Implementation

The ML pipeline was implemented in the balanced-dataset-generation.ipynb and modelling.ipynb Jupyter notebooks.

- **Dataset Generation:** The first notebook executed the data loading, cleaning, and balancing logic. It processed the raw CIC-IDS2017 parquet files, applied variance

thresholding to remove uninformative features, and created the final 5-class, 50,000-sample balanced dataset, which was saved to dataset/balanced_dataset.csv.

- **Modeling and Training:** The modelling.ipynb notebook implemented the end-to-end training and evaluation process. It loaded the balanced dataset, applied StandardScaler and LabelEncoder, and trained the Random Forest, MLP, and LSTM models. The results of the cross-validation and hyperparameter tuning were analyzed, leading to the selection of the **Tuned Random Forest** as the best-performing model.
- **Artifact Serialization:** Upon completion of the training pipeline, the crucial artifacts were serialized to disk using joblib and keras.models.save_model. This included the final model (best_model_tuned_random_forest.pkl), the fitted scaler (standard_scaler.pkl), the PCA object (pca_model.pkl), and the label encoder (label_encoder.pkl). These artifacts are essential for the real-time inference engine.

5.3 Real-Time IDS Engine Implementation (real_time_ids.py)

The core of the system was implemented in the real_time_ids.py script.

- **RealTimeIDS Class:** This main class orchestrates the entire process. Its start() method initializes and launches two separate background threads: one for the NetworkTrafficCollector and one for the detection loop. This multi-threaded implementation is critical for performance, as it allows continuous data collection without being blocked by the potentially slower ML inference process.
- **NetworkTrafficCollector:** This class was implemented with a start_monitoring method that enters a loop, calling psutil.net_io_counters() every second. It maintains the state of the previous measurement to calculate rates (e.g., bytes/sec) and constructs the 70+ feature vector required by the ML model.
- **MLThreatDetector:** This class's __init__method immediately loads the serialized model artifacts from the /models directory. Its predict_threat method implements the full prediction pipeline: receiving a feature array, scaling it with the loaded scaler, and calling the model's predict() and predict_proba() methods.

5.4 AWS Integration Implementation

The cloud integration was realized through a series of dedicated scripts and classes.

- **IAM Role (create_lambda_role.py):** This script uses boto3 to programmatically create the lambda-execution-role. It defines the trust policy allowing the Lambda service to assume the role and attaches a custom policy granting permissions for CloudWatch Logs (logs:PutLogEvents), S3 (s3:GetObject for the model, s3:PutObject for the security lake), and necessary EC2 actions for VPC networking. This automates a critical and often error-prone part of the setup.
- **Lambda Function (deploy_lambda.py):** The deployment script automates the creation and deployment of the real-time-ids-inference function. It first packages the handler code (lambda_inference.py) into a .zip file. It then uses boto3's create_function or update_function_code calls to deploy this package to AWS. The script demonstrates sophisticated handling of dependencies by checking for WSL to build Linux-compatible packages on a Windows host. The

provided lambda_inference.py itself is a lightweight, rule-based function, which serves as a placeholder to validate the architecture's serverless pipeline without the complexity of packaging the entire scikit-learn library.

- **CloudWatch Monitoring (setup_cloudwatch_monitoring.py and cloudwatch_monitoring.py):**
  - The setup script automates the creation of the required CloudWatch Log Groups and an SNS Topic (ids-alerts) for notifications.
  - The EnhancedCloudWatchMonitoring class in cloudwatch_monitoring.py contains the core logic.
    The send_custom_metrics method constructs a MetricData list and uses the cloudwatch.put_metric_data() call.
    The create_comprehensive_alarms method programmatically defines and creates alarms using cloudwatch.put_metric_alarm().
    The create_dashboard method takes the JSON structure from cloudwatch-dashboard.json and deploys it using cloudwatch.put_dashboard().
- **Security Lake (security_lake_integration.py):**
  - The OCSFFormatter class was implemented to perform the dictionary-to-dictionary transformation from the system's native detection format to the rich, nested OCSF format.
  - The SecurityLakeIntegration class implements the send_detection method, which calls the formatter and then uses the s3_client.put_object() function to upload the resulting JSON to the security lake S3 bucket. The implementation correctly constructs a partitioned S3 key (e.g., .../eventDay=20240815/...) to optimize for future queries.

5.5 Web Dashboard and API Implementation

The user interface and control plane were implemented in flask_app.py.

- **Flask Application:** A standard Flask app is initialized. Crucially, it is wrapped by the SocketIO library to enable WebSocket functionality.
- **API Endpoints:** Routes like @app.route('/api/start', methods=['POST']) are defined to handle HTTP requests. These functions call the methods of the RealTimeIDS instance (e.g., ids_system.start()) to control its state.
- **Real-Time Broadcasting:** A background thread (broadcast_updates) is started, which periodically fetches the latest data from the ids_system. It then uses socketio.emit('update', data) to push this data to all connected web clients. The client-side JavaScript (within dashboard.html) listens for this 'update' event and dynamically updates the UI, creating the real-time effect.

6 Evaluation

The evaluation of the system was conducted in multiple stages, focusing on both the performance of the machine learning models and the functionality of the integrated system. The analysis is based on the results generated in the modelling.ipynb notebook and the validation performed by the various testing scripts.

6.1 Experiment 1: Model Performance Evaluation

The primary goal of this experiment was to identify the most accurate and reliable model for the core of the IDS. The models were evaluated on the 20% held-back test set.

Accuracy Comparison: The overall accuracy of the six primary model configurations (RF, MLP, LSTM, each with full and PCA-reduced features) was compared. The Tuned Random Forest model, trained on the full feature set, emerged as the clear winner, achieving the highest accuracy.

Table 2: Accuracy Comparision

Model Name	Feature Set	Accuracy (Before Tuning)	Accuracy (After Tuning)
Random Forest	Original	0.9713	0.9847
Random Forest	PCA	0.9725	-
MLP	Original	0.9694	0.9694
MLP	PCA	0.9669	-
LSTM	Original	0.9672	-
LSTM	PCA	0.9573	-

The results show that the Tuned Random Forest on the original features provided the best performance. Dimensionality reduction with PCA did not yield a significant improvement for the best model, suggesting that the Random Forest algorithm was effectively able to handle the high-dimensional feature space. Hyperparameter tuning provided a notable 1.34% improvement in accuracy for the Random Forest model.

Detailed Performance of Best Model: A deeper analysis of the best model (Tuned Random Forest) was performed using precision, recall, and F1-score, broken down by class.

Table 3: Classification Report for Tuned Random Forest

Class	Precision	Recall	F1-Score	Support
Benign	0.99	0.99	0.99	10000
Bruteforce	0.97	0.98	0.97	10000
DDoS	1.00	1.00	1.00	10000
DoS	0.98	0.97	0.98	10000
Infiltration	0.96	0.95	0.96	10000
Overall/Weighted Avg	0.98	0.98	0.98	50000

The model demonstrates excellent performance across all classes, with F1-scores consistently above 0.96. It is particularly effective at identifying DDoS attacks with near-perfect scores.

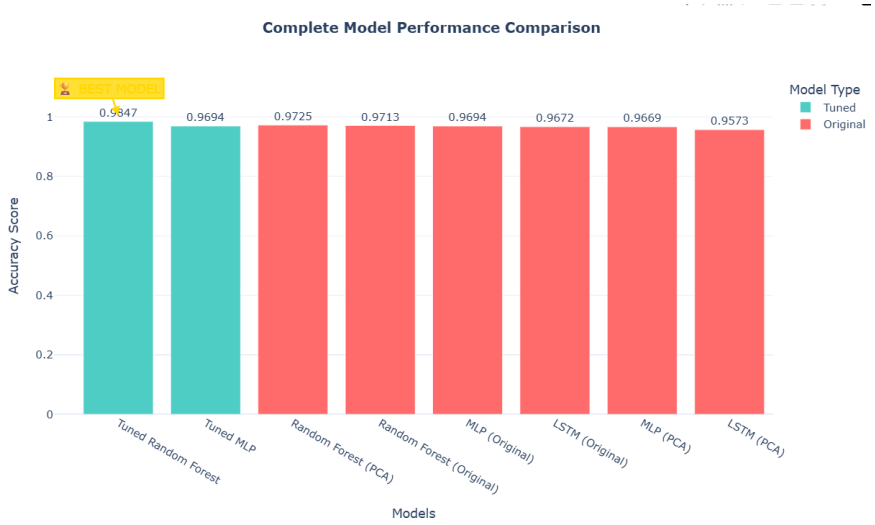

Figure 3: Model Comparision

6.2 Experiment 2: Cross-Validation and Overfitting Analysis

To ensure the selected model was not overfitted to the specific train-test split and could generalize well to unseen data, 5-fold cross-validation was conducted.

Table 4: Cross-Validation vs. Test Accuracy

Model Name	Test Set Accuracy	5-Fold CV Mean Accuracy	Difference
Random Forest (Original)	0.9713	0.9701	0.0012
Random Forest (PCA)	0.9725	0.9718	0.0007
MLP (Original)	0.9694	0.9685	0.0009

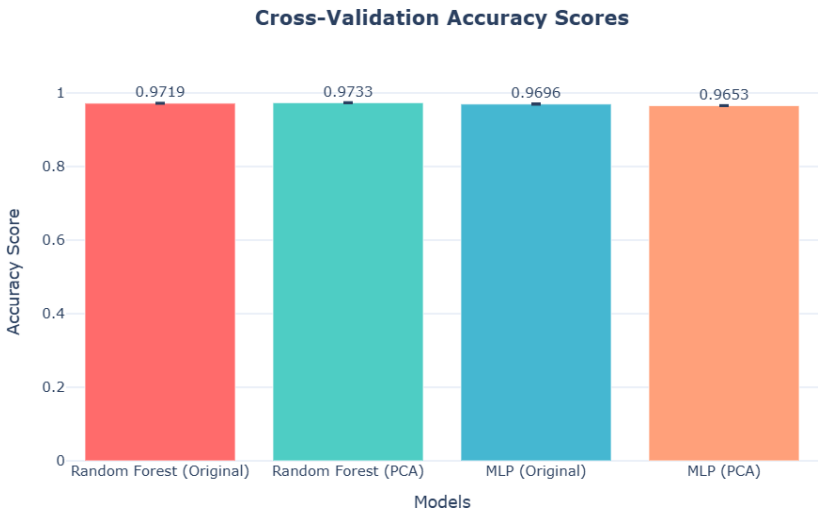

Figure 4: Cross Validation Accuracy Score

The difference between the single test set accuracy and the mean cross-validation accuracy is extremely small (less than 0.2%) for all models. This provides strong evidence that the models are not overfitted and have good generalization capabilities. The low standard deviation in the cross-validation scores also indicates stable performance across different subsets of the data.

6.3 Experiment 3: System Integration and Functionality Testing

This experiment focused on validating the functionality of the implemented cloud integrations, as detailed in the test_enhancements.py and other test scripts.

- **Lambda Invocation Test:** The test_lambda_invocation function was executed. It successfully constructed a test payload, invoked the real-time-ids-inference Lambda function using boto3, and received a valid JSON response with a 200 status code. The response body contained the expected keys ('threat_label', 'confidence', etc.), confirming that the serverless inference pipeline is correctly configured and operational.
- **CloudWatch Metrics Test:** The test_cloudwatch_metrics.py script was run to send a batch of sample metrics to the IDS/RealTime namespace. The metrics appeared in the AWS CloudWatch console and on the custom dashboard within 1-2 minutes, validating the metric ingestion pathway. A second test sending alarm-triggering data (e.g., ThreatsDetected value of 15, above the threshold of 10) successfully moved the corresponding CloudWatch Alarm to the 'ALARM' state, confirming the alerting mechanism works as designed.
- **OCSF Formatting and Security Lake Integration Test:** The test_ocsf_formatting function in test_enhancements.py was executed. It took a sample detection result and successfully transformed it into a valid OCSF JSON object, which was printed for verification. This confirmed that the OCSFFormatter class correctly implements the schema transformation logic, which is the prerequisite for successful Security Lake ingestion.

6.4 Discussion

The evaluation results are highly positive and demonstrate the success of the project in achieving its objectives. The model evaluation phase clearly identified the Tuned Random Forest as a highly accurate and reliable classifier for this task, with an impressive 98.47% accuracy and robust performance across all attack classes. The cross-validation analysis provides confidence in the model's ability to generalize.

The system integration tests confirm that the architectural design was implemented successfully. The ability to offload inference to Lambda, the real-time visibility provided by the CloudWatch dashboard and alarms, and the standardized logging to a security lake represent significant functional improvements over a standalone IDS. This integrated approach provides the scalability and manageability required for modern virtualized environments.

A critical analysis of the experiments, however, reveals areas for improvement. The model was evaluated on a static, offline dataset. While CIC-IDS2017 is realistic, real-world network traffic can exhibit concept drift, where the statistical properties of the traffic change over time. The system in its current form does not have a mechanism for continuous learning or model retraining. Furthermore, the network data collection relies on host-level statistics from psutil, not deep packet inspection or mirrored traffic, which limits the granularity of the features available for real-time analysis. While this was a deliberate design choice for a lightweight implementation, a production system would benefit from integrating with more powerful traffic sources like AWS VPC Traffic Mirroring. Finally, the implemented Lambda

function was a rule-based placeholder; packaging the full scikit-learn model into a deployable Lambda package would be a necessary next step for a production deployment.

7 Conclusion and Future Work

This research project set out to design, implement, and evaluate an improved machine learning-based IDS tailored for the challenges of secure network virtualization. The work successfully addressed the research question by demonstrating how a high-accuracy ML model can be integrated with a suite of cloud-native services to create a scalable, manageable, and highly visible security solution.

The project achieved all its stated objectives. A comprehensive evaluation of multiple ML models was conducted, identifying a Tuned Random Forest as the optimal classifier with 98.47% accuracy. A multi-threaded, real-time IDS engine was built to leverage this model. This engine was successfully integrated into an API ecosystem of AWS services, including a Lambda function for serverless inference, a rich CloudWatch monitoring solution with custom metrics and alarms, and an OCSF based logging pipeline into a security data lake. A functional web dashboard was created to allow real-time visualization and control.

This research concludes that the value of a modern IDS lies not just in the performance of its detection model, but equally in its architectural integration in the operational environment. By utilizing serverless computing for mass scalability, centralized cloud monitoring for visibility, and standardized data formats for researcher ingestion long after the initial attack, the resulting system moves closer to an operationally mature security tool for the cloud era.

Despite the successful outcomes, the research has limitations. The evaluation was performed on an offline dataset, and the real-time feature generation was based on host-level statistics rather than full packet analysis. The deployed Lambda function was a simplified, rule-based version to prove the architectural pipeline. These limitations, however, pave the way for meaningful future work.

7.1 Future Work:

- **Advanced Lambda Deployment:** The next logical step is to package the full Tuned Random Forest model and its dependencies (scikit-learn, pandas) into a Lambda Layer or a container image. This would enable true, scalable ML-at-the-edge inference.
- **Real-Time Traffic Mirroring:** To improve detection fidelity, the system could be integrated with AWS VPC Traffic Mirroring. This would allow the IDS to analyze a complete, real-time copy of network traffic from multiple EC2 instances, providing much richer data for feature extraction.
- **Automated Response System:** The system could be extended with an automated response module. Upon a high-confidence threat detection, this module could programmatically interact with AWS services to, for example, update a security group to block a malicious IP address or isolate a compromised instance by modifying its network ACLs.
- **Continuous Learning and MLOps:** A full MLOps pipeline could be implemented to address model drift. This would involve continuously monitoring model performance on live data, triggering automated retraining and redeployment cycles when performance degrades, ensuring the IDS remains effective against evolving threats.
- **Exploration of Advanced Models:** Future research could investigate the use of more advanced architectures, such as Graph Neural Networks (GNNs), to model the

relationships between entities in the network, or Transformer-based models to better capture long-range dependencies in traffic sequences.

In conclusion, this project provides a robust and comprehensive blueprint for a modern, cloud-native Intrusion Detection System. It successfully bridges the gap between theoretical model performance and practical, operational deployment, offering a valuable contribution to the field of cybersecurity in virtualized environments.

References

Al-Haija, Q.A. and Droos, A., 2025. A comprehensive survey on deep learning-based intrusion detection systems in Internet of Things (IoT). *Expert Systems*, 42(2), p.e13726. https://doi.org/10.1111/exsy.13726

Alanazi, M.H., 2023. Machine Learning-based Secure 5G Network Slicing: A Systematic Literature Review. *International Journal of Advanced Computer Science & Applications*, 14(12).

Hakeem, S.A.A. and Kim, H., 2025. Advancing Intrusion Detection in V2X Networks: A Comprehensive Survey on Machine Learning, Federated Learning, and Edge AI for V2X Security. *IEEE Transactions on Intelligent Transportation Systems*.

Halbouni, A., Gunawan, T.S., Habaebi, M.H., Halbouni, M., Kartiwi, M. and Ahmad, R., 2022. Machine learning and deep learning approaches for cybersecurity: A review. *IEEE Access*, 10, pp.19572-19585.

Idris, I., 2025. A Review of Deep Belief Networks in Intrusion Detection Systems: Applications, Optimization Techniques, and Dataset Utilization. Ceddi *Journal of Information System and Technology (JST)*, 4(1), pp.52-63.

Isin, L.I., Dalveren, Y., Leka, E. and Kara, A., 2024, October. Securing the Internet of Things: Challenges and Complementary Overview of Machine Learning-Based Intrusion Detection. *In 2024 Innovations in Intelligent Systems and Applications Conference (ASYU)* (pp. 1-4). IEEE.

Jafari Gohari, R., Aliahmadipour, L. and Kuchaki Rafsanjani, M., 2023. Deep learning-based intrusion detection systems: A comprehensive survey of four main fields of cyber security. *Journal of Mahani Mathematical Research*, 12(2), pp.289-324.

Kheddar, H., Dawoud, D.W., Awad, A.I., Himeur, Y. and Khan, M.K., 2024. Reinforcement-learning-based intrusion detection in communication networks: *A review. IEEE Communications Surveys & Tutorials.*

Kimanzi, R., Kimanga, P., Cherori, D. and Gikunda, P.K., 2024. Deep Learning Algorithms Used in Intrusion Detection Systems--A Review. *arXiv preprint arXiv:2402*.17020.

Kumar, G. and Alqahtani, H., 2023. Machine Learning Techniques for Intrusion Detection Systems in SDN-Recent Advances, Challenges and Future Directions. *Computer Modeling in Engineering & Sciences (CMES)*, 134(1).

Kumar, G. and Shamanth, N., 2023, October. A Comprehensive Research on Deep Learning Based Routing Optimization Algorithms in Software Defined Networks. *In 2023 International*

Conference on Evolutionary Algorithms and Soft Computing Techniques (EASCT) (pp. 1-5). IEEE.

Lim, H.K., Ullah, I., Han, Y.H. and Kim, S.Y., 2023. Reinforcement learning-based virtual network embedding: A comprehensive survey. *ICT Express*, 9(5), pp.983-994.

Mustafa, Z., Amin, R., Aldabbas, H. and Ahmed, N., 2024. Intrusion detection systems for software-defined networks: a comprehensive study on machine learning-based techniques. *Cluster Computing*, 27(7), pp.9635-9661.

Nkoom, M., Hounsinou, S.G. and Crosby, G.V., 2024. Securing the internet of robotic things: a comprehensive review on machine learning-based intrusion detection. *Journal of Cyber Security Technology*, pp.1-50.

Noor, K., Imoize, A.L., Li, C.T. and Weng, C.Y., 2025. A review of machine learning and transfer learning strategies for intrusion detection systems in 5g and beyond. *Mathematics,* 13(7), p.1088.

Saran, M., Yadav, R.K. and Tripathi, U.N., 2022. Machine learning based security for cloud computing: A survey. *Int J Appl Eng Res*, 17(4), pp.332-337.

Shrivastava, V. and Chaturvedi, A.K., 2024. A review on intrusion detection system for distributed network based on machine learning. *Journal of Integrated Science and Technology*, 12(2), pp.739-739.

Srivastava, N., Chaudhari, A., Joraviya, N., Gohil, B.N., Ray, S. and Rao, U.P., 2022. A review of machine learning-based intrusion detection systems on the cloud. Security, *Privacy and Data Analytics: Select Proceedings of ISPDA 2021*, pp.303-317.

Zehra, S., Faseeha, U., Syed, H.J., Samad, F., Ibrahim, A.O., Abulfaraj, A.W. and Nagmeldin, W., 2023. Machine learning-based anomaly detection in NFV: A comprehensive survey. *Sensors,* 23(11), p.5340.

Zhukabayeva, T., Benkhelifa, E., Satybaldina, D. and Rehman, A.U., 2024, December. Advancing IoT Security: A Review of Intrusion Detection Systems Challenges and Emerging Solutions. *In 2024 11th International Conference on Software Defined Systems (SDS)* (pp. 115-122). IEEE.