National College of Ireland

Threat Detection and Intrusion Prevention in Cloud-Based Infrastructures

MSc Research Project

Abhishek Joshy
Student ID: X23323507

School of Computing
National College of Ireland

Supervisor: Sean Heaney

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Abhishek Joshy
Student ID:	X23323507
Programme:	MSC Cloud Computing **Year:** 2024-2025
Module:	MSC Research Project
Supervisor:	Sean Heaney
Submission Due Date:	11-08-2025
Project Title:	Threat Detection And Intrusion Prevention in Cloud Based Infrastructure
Word Count:	8091…… **Page Count** …23…

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: Abhishek Joshy

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):	

Threat Detection and Intrusion Prevention in Cloud-Based Infrastructures

Abhishek Joshy
Student ID:x23323507

Abstract

The rapid migration of enterprise infrastructure to cloud computing environments has introduced new and complex security challenges. Traditional security perimeters are dissolving, necessitating advanced, intelligent systems capable of monitoring vast and dynamic network traffic for malicious activities. This research addresses the critical need for effective threat detection and intrusion prevention within cloud-based infrastructures. The project develops and evaluates a comprehensive solution by leveraging a large-scale public dataset, CIC-IDS2017, to train a suite of machine learning models. A systematic methodology involving data preprocessing, feature engineering, and dimensionality reduction was employed to prepare the data. Multiple classification algorithms were trained and rigorously evaluated, with a Decision Tree model emerging as the optimal choice, achieving a classification accuracy of 97.4%. The core contribution of this work is the operationalization of this high-performance model within a cloud-native architecture. A Flask web application was developed to serve as the system's engine, featuring a real-time analytics dashboard and a background process for continuous, simulated packet scanning. This entire system was deployed on Amazon Web Services (AWS), demonstrating a practical, end-to-end implementation. A key innovation is the deep integration with AWS CloudWatch, enabling the system to export custom security metrics and logs for centralized monitoring, alerting, and long-term analysis. The final artifact is not merely a theoretical model but a fully functional prototype that provides a blueprint for deploying intelligent, scalable, and resilient intrusion detection systems in modern cloud environments.

1 Introduction

The paradigm of information technology has undergone a seismic shift with the widespread adoption of cloud computing. Services offered by providers such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform have become the backbone of modern digital enterprises, offering unprecedented scalability, flexibility, and cost-efficiency. However, this migration has also reshaped the landscape of cybersecurity. The centralized, on-premises data centers with well-defined perimeters are being replaced by distributed, dynamic, and multi-tenant cloud environments (Kambala, 2023). This new architecture, while powerful, expands the attack surface and introduces unique vulnerabilities that traditional security measures struggle to address. Internal threats are no longer limited to external source; they can also originate from the compromised state of such instances or

misconfigured services of the cloud infrastructure itself, which indicate a need for robust internal monitoring(Ahmad et al., 2021).

The rate of volume and velocity of incoming network traffic is relatively voluminous in a usual cloud infrastructure. A manual analysis and detection of malicious threats are not feasible at all. The intricacy of the new attack vectors employed by these malicious actors is such that, in most cases, these attack vectors are stealthy and complex; thus, it is very possible for such attack vectors to look like benign traffic in order to evade traditional signature-based detection systems (Aminu et al., 2024). Therefore, there is a dire need for fully automated and intelligent, adaptive security measures capable of analyzing network behavior as it happens, for the purposes of intrusion identification and feared invasion. In so doing, one can say without doubt that machine learning (ML) has learnt a lot in this regard and is gradually becoming the magic wand in drawing complicated patterns from vast datasets that have very high accuracy and clear discrimination between normal and malicious activities on the network (Viharika and Balaji, 2024).

A lot of effort has gone into academia in recent years to come up with novel ML models aimed at detection of intrusion. But, many times, a good gap lies between performance in theory compared with real-world deployment—in particular, cloud (Liu et al. 2022). A model with high accuracy turns out to be useless if it cannot be efficiently integrated into a scalable, resilient, and manageable cloud-native architecture. This research aims to fill such a gap with the following research question:

What is the best way to engineer machine learning into a cloud-native architecture for deployment across conditions to provide real-time, resilient, and monitorable threat detection and intrusion prevention systems?

To answer this question, this project pursues several key objectives:

1. To process and analyze a comprehensive, real-world network intrusion dataset (CIC-IDS2017) to create a suitable foundation for model training.
2. To develop, train, and rigorously evaluate a range of machine learning models to identify the most effective classifier for network traffic.
3. To implement the selected model into a functional web application capable of performing real-time predictions and visualizing security analytics.
4. To deploy this application onto a leading public cloud platform (AWS) to validate its operational viability in a realistic environment.
5. To integrate the system with native cloud monitoring services (AWS CloudWatch) to establish a framework for continuous operational visibility, alerting, and security posture management.

This study has primarily documented the design, implementation and evaluation of an end-to-end threat detection system demonstrating both the capability of a machine learning model and, practical design for deploying it in the cloud. It illustrates how a simple Python application can leverage an array of AWS services (e.g., EC2 for compute and CloudWatch for monitoring) to create a complete and effective security solution. Figure 1 illustrates the high-level conceptual flow of the proposed system.

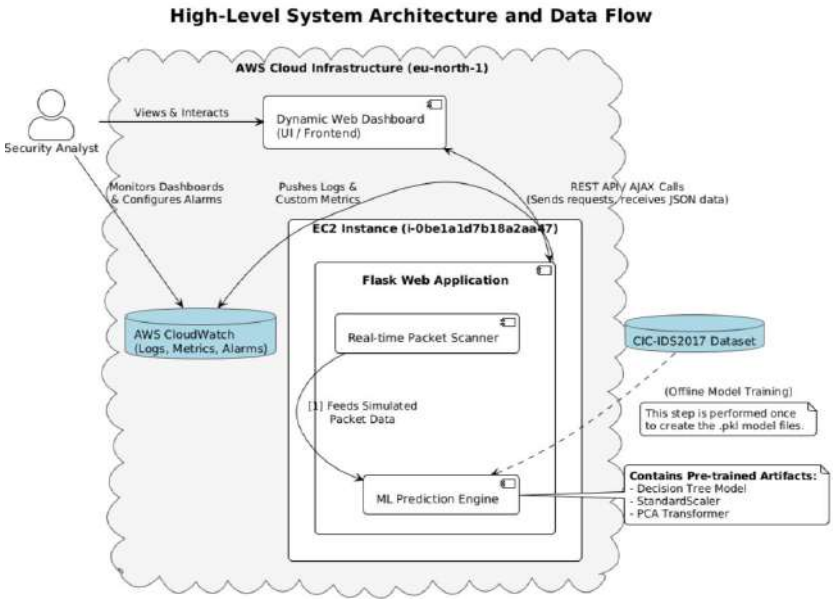

Figure 1: High-Level System Architecture and Data Flow.

This report is organized as follows. Section 2 provides a literature review of risk to cloud security and the machine-learning based intrusion detection architectures in the literature. Section 3 discusses the research methodology, dataset, preprocessing, and evaluation strategy. Section 4 describes the system's design specification including design architecture and system components. Section 5 describes how the machine-learning pipeline, Flask application and AWS infrastructure were implemented. Section 6 evaluates the results in detail, discussing both the model's success and the deployed system functionality. Finally, Section 7 concludes summarising the main points in the report and providing recommendations for future work.

2 Related Work

This section situates the current research within the existing body of academic literature on cloud security, intrusion detection systems (IDS), and the application of machine learning in this domain. A critical review of prior work helps to identify established methods, outstanding challenges, and the specific gaps this project aims to address.

2.1 The Evolving Threat Landscape in Cloud Computing

Cloud computing environments, by their nature, are fundamentally different from traditional on-premises networks. The shared responsibility model, dynamic resource allocation, and extensive use of APIs create a unique set of security challenges. These challenges have been documented through various surveys. Liu et al. (2022) presented a full literature survey on cloud intrusion detection systems, classifying the respective threats and discussing the limitations of existing solutions in handling two situations: scale and encrypted traffic flow, as they exist in the cloud environment. Likewise, Chang et al. (2022) survey IDSs in the domains of fog and cloud computing, stressing the requirement for lightweight and distributed detection mechanisms that can effectively operate at various layers in the cloud stack. The general takeaway from these reports is that security is not a 'one size fits all'

solution in the cloud context. Security solutions need to be able to incorporate contextual awareness and key learning capabilities.

Another equally important consideration in the literature is the complexity of such attacks. Al Jalaah et al. (2023) conducted a systematic review of legitimate cloud service abuse in the construction of Command-and-Control (C&C) infrastructure. This draws attention to a pretty significant issue: attackers leverage legitimate services of the cloud to facilitate their campaign, so using simple blacklists or signatures for detection becomes that much harder, paving the way for more adaptable approaches to detection. The focus on the need for behavioral analysis and anomaly detection is what machine learning is good at. Also, there are another whole set of significant issues raised by Distributed Denial of Service (DDoS) attacks. Devi and Subbulakshmi (2023) consider the DDoS detection and prevention problems and note that targeting the elasticity of the Cloud for a DDoS attack is a double-edged sword: it gives attackers a chance to leverage elasticity to amplify their DDoS attack damage by orders of magnitude. This paper calls attention to the need for models trained to respond based on the recognition of certain high-volume attack trends, specifically DDoS.

2.2 Machine Learning and AI for Intrusion Detection

The landscape of research applying Artificial Intelligence (AI) and Machine Learning (ML) to cybersecurity is new and provides a strong alternative approach to traditional, rule-based security systems. In reviewing cloud-based IDS trends and challenges, Prabu and Sudhakar (2024) highlight the intrinsic advantages of ML-based systems for identifying new, zero-day attacks with no signature. They noted the potential advantage of using ML-based systems for cloud-based IDS based on what is mostly the signature of traffic (features) and not simply the signature of known attacks. The quality of features for ML was emphasized and algorithm choice will directly inform research methodology for this project.

A reasonable and practical number of researchers had focused on specific ML methods and DL methods. Devi and Jain (2024) focused on improving cloud security with deep learning methods, where they asserted that types of models like Recurrent Neural Networks (RNNs) could learn temporal dependencies in traffic, but often for trade-offs in computation, though their results reflect a memorable, point in the overall computational spectrum. This presents a trade-off for cloud deployment, where computational resources directly translate to operational costs. Viharika and Balaji (2024) review AI-driven IDS with a focus on hybrid models, which combine multiple algorithms to leverage their respective strengths. While powerful, these hybrid systems can be complex to implement and maintain in a production environment. The current project, while not implementing a hybrid model, establishes a foundational pipeline upon which such a system could be built.

Hemanth (2025) discusses innovative approaches using AI, and Parisa and Banerjee (2024) provide a comparative review of traditional versus next-generation, AI-enabled cloud security solutions. A common thread in these forward-looking papers is the move towards proactive and predictive security. However, many of these studies remain at a conceptual or simulation level. They propose and test models in an offline setting but often do not address the engineering challenges of integrating them into a live, operational cloud system that includes real-time data ingestion, prediction serving, and monitoring (Umesh et al., n.d.). This is the primary gap that this research seeks to fill.

2.3 Cloud-Based vs. Edge-Based Detection

The discussion of where intrusion detection should occur is also relevant. Ebadinezhad and Alsaroah (2025) provide a comparative analysis of edge-based versus cloud-based IDS. Edge-based systems perform analysis closer to the data source, which can reduce latency and minimize the amount of data sent to the central cloud, preserving privacy and saving bandwidth. However, edge devices typically have limited computational power, restricting the complexity of the models that can be deployed. Cloud-based systems, like the one developed in this project, can leverage virtually unlimited computational resources, allowing for more complex models and centralized analysis of traffic from across the entire infrastructure. This centralized view is crucial for detecting large-scale, coordinated attacks that might appear as insignificant noise at the individual edge level. This project focuses exclusively on the cloud-based approach, acknowledging its strengths in centralized control and powerful analytics, which are paramount for a Security Operations Center (SOC) analyst.

2.4 Research Gap and Justification

The review of related work reveals several key points. First, the security challenges in the cloud are well-documented and distinct from on-premises environments. Second, machine learning is widely accepted as a powerful tool for building effective IDS. Third, a large body of work exists that proposes and evaluates various ML models using benchmark datasets. However, a significant gap remains in the literature concerning the end-to-end engineering and deployment of these systems within a real-world cloud context. Many studies stop after reporting model accuracy, neglecting the critical steps of:

- Building a robust, scalable application to serve the model.
- Deploying this application on a public cloud infrastructure.
- Integrating the system with native cloud monitoring and management tools for operational visibility.

This project directly addresses this gap. It moves beyond theoretical model evaluation to create a tangible, deployed artifact. The emphasis is not just on the 97.4% accuracy of the Decision Tree model, but on the architecture that allows this model to function as part of a cohesive, cloud-native security solution integrated with AWS CloudWatch. By documenting this entire process, this research provides a practical blueprint that others can follow to operationalize ML for cloud security.

3 Research Methodology

This section outlines the systematic and scientific approach taken to address the research question. The methodology follows a structured, quantitative, and experimental process, encompassing data acquisition, preparation, model development, and evaluation. The entire process is designed to be transparent, reproducible, and grounded in established data science and software engineering practices.

3.1 Dataset Selection and Description

The foundation of any machine learning project is the quality and relevance of its dataset. For this research, the CIC-IDS2017 dataset from the Canadian Institute for Cybersecurity was selected. This choice was deliberate and based on several key criteria:

- **Realism and Modernity:** The dataset originated from a realistic network topology and ran over a five-day period capturing a diverse range of benign network traffic as well as common, contemporary attack types. This is a tremendously important factor from previous datasets like KDD-99 leveraging simulated benign behavior and legacy threat analysis, which do not account for threats in today's landscape.
- **Extensive Feature Set:** It contains 78 network flow features taken from the CICFlowMeter tool which provide a comprehensive, quantifiable description of each network connection including time information, packet information, flag counts, and so forth, all contributing to classifying attacks and hijacking events.
- **Labeled Data**: Each network flow is identified as either 'Benign' or one specific attack type (e.g., DDoS, PortScan, Web Attack). This affords a significant level of usefulness for supervised machine learning efforts.
- **Publicly Available**: The dataset serves a public repository meaning that the research and it's results can be openly and transparently shared, and also validated and reproduced by others in the field.

The dataset has numerous attacks with attacks including Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attacks (Brute Force, XSS, SQL Injection), Infiltration, PortScan, and Botnet activity. With this diversity the model to be trained will be able to recognize a very large range of threats, from a multitude of attack vectors.

3.2 Data Preparation and Preprocessing

The raw CIC-IDS2017 dataset is massive and contains several classes with a highly imbalanced distribution of samples. To create a manageable and effective training set, a multi-step preprocessing pipeline was executed, as documented in the balanced-dataset-generation.ipynb notebook.

1. **Data Ingestion and Merging:** The dataset is provided in daily files in Parquet format. The first step was to ingest all these files and merge them into a single, unified DataFrame using the Pandas library.
2. **Initial Analysis and Imbalance Identification:** An initial analysis of the 'Label' column revealed a severe class imbalance. For instance, the 'Benign' class contained nearly 2 million samples, while some attack classes like 'Heartbleed' had only 11. Training a model on such imbalanced data can lead to a classifier that is heavily biased towards the majority class and performs poorly on minority, yet critical, attack classes.
3. **Creation of a Balanced Subset:** To mitigate this issue and create a more computationally efficient dataset for iterative development, a balanced subset was created. A target of approximately 100,000 total rows was set. Labels with extremely low counts (fewer than 50 occurrences), such as "Heartbleed" and "Infiltration," were removed as they provided insufficient data for the model to learn meaningful patterns.

For the remaining 12 classes, a strategy of undersampling the majority classes (like 'Benign' and 'DDoS') and including all samples from the minority classes (like 'Web Attack XSS') was employed. This resulted in a final balanced dataset of 58,610 samples, providing a much more even distribution across the most significant attack types and a robust benign baseline.

4. **Data Cleaning:** The final balanced dataset was then subjected to a cleaning process. This involved checking for and removing any rows with missing (NaN) values. The dataset was also inspected for infinite values, which can arise from division-by-zero errors during feature calculation (e.g., in 'Flow Bytes/s'). Any rows containing infinite values were also removed to ensure numerical stability during model training.

3.3 Feature Engineering and Selection

To enhance the model's predictive power, several new features were engineered from the existing ones. These composite features are designed to capture higher-level concepts about the network flow that may not be immediately apparent from the base features alone.

- **Total_Packets:** The sum of 'Total Fwd Packets' and 'Total Backward Packets'.
- **Total_Length:** The sum of 'Fwd Packets Length Total' and 'Bwd Packets Length Total'.
- **Packets_per_Second:** The ratio of 'Total_Packets' to 'Flow Duration', providing a measure of traffic intensity.
- **Avg_Packet_Size:** The ratio of 'Total_Length' to 'Total_Packets'.
- **Fwd_Bwd_Ratio:** The ratio of forward packets to backward packets, which can be indicative of certain scanning or DoS attacks.

After feature engineering, the dataset contained 82 numerical features.

3.4 Machine Learning Pipeline

A standard, yet robust, machine learning pipeline was constructed to process the data and train the models.

The categorical 'Label' column was converted into numerical format by Scikit-learn's LabelEncoder. This is an important step since machine learning algorithms work on numerical data.

Using the stratified split strategy, the data was split into training (80%) and test (20%) datasets. Stratified sampling ensures that the proportion of each class remains the same in both training and testing sets, therefore making it very important for reliable evaluation, especially for multi-class data.

All numerical features were scaled with StandardScaler. This facilitates by standardizing the features by removing the mean and scaling to unit variance. The scaling is very important for algorithms sensitive to the magnitude of the feature such as Logistic Regression and K-Nearest Neighbors or for techniques such as Principal Component Analysis.

PCA was applied to reduce model complexity, reduce computational costs, and possibly increase performance by removing noise. The choice of the number of components is such as to capture 95% of the total variance of the data. This reduced the number of features from 82 to 22, thus considerably simplifying the input to the models while yet maintaining the bulk of the information. This assumption bears weight particularly in the scenario of a cloud-

deployed system: a model with fewer features will consume lesser computational power and memory, thereby costing lesser in operational costs on an EC2 instance.

3.5 Model Training and Evaluation

A combination of five diverse and widely-used classification models were selected for analysis and training:

- **Logistic Regression**: A linear model that is quick, yet interpretable.
- **Decision Tree**: A non-linear, tree-based model, with strong tabular data predictability and interpretability.
- **AdaBoost:** An ensemble method that fit multiple weak learners into a strong classifier.
- **Gaussian Naive Bayes:** A probabilistic classifier based on Bayes' theorem with an assumption of feature independence.
- **K-Nearest Neighbors (KNN):** A non-parametric, instance-based learning algorithm.

Each model was trained on the preprocessed, PCA-transformed training data. The performance was then evaluated on the unseen test set using a suite of standard metrics:

- **Accuracy:** The proportion of correctly classified instances.
- **Precision:** The ability of the classifier not to label a negative sample as positive.
- **Recall (Sensitivity):** The ability of the classifier to find all the positive samples.
- **F1-Score:** The harmonic mean of precision and recall, providing a single score that balances both concerns.
- **Confusion Matrix:** A table that visualizes the performance of the classifier, showing true positives, true negatives, false positives, and false negatives for each class.
- **Cross-Validation:** To ensure the models were robust and not overfitted to the specific train-test split, 5-fold stratified cross-validation was performed on the training set.

The model demonstrating the best overall performance, particularly in terms of accuracy and F1-score, was selected as the primary model for deployment in the Flask application.

4 Design Specification

This section details the architectural design of the "Threat Detection and Intrusion Prevention in Cloud-Based Infrastructures" system. The design is modular, scalable, and tailored for deployment within a public cloud environment, specifically AWS. The architecture is broken down into its core components, specifying their roles, responsibilities, and interactions.

4.1 System Architecture

The system is designed as a multi-tiered, cloud-native application. The architecture prioritizes separation of concerns, which facilitates development, maintenance, and future scalability. The logical tiers are as follows:

1. **Presentation Tier:** This is the user-facing component, a dynamic web dashboard. It is designed using HTML, CSS, Bootstrap 5 for responsive layout, and Plotly.js for interactive data visualizations. Its primary purpose is to provide a human-readable

interface for Security Operations Center (SOC) analysts to monitor network health, view threat analytics, and interact with the system. It communicates with the backend tier via RESTful API calls.

2. **Application Tier (Backend):** This is the heart of the system, implemented as a Flask web application running on an AWS EC2 instance. This tier is responsible for all core logic:
    - **API Server:** Exposes a set of REST APIs to serve data to the frontend and accept prediction requests.
    - **Threat Detection Engine:** Hosts the trained machine learning model and associated preprocessing components (scaler, PCA). It performs real-time predictions on incoming data.
    - **Real-Time Simulation Engine:** A background thread that continuously generates simulated network traffic data and feeds it to the detection engine, ensuring the dashboard is always populated with live, dynamic data for demonstration and testing purposes.
    - **State Management:** An in-memory data store for tracking prediction history, malicious IP addresses, and hourly traffic statistics.
3. **Data Tier:** This tier consists of the persistent artifacts generated from the machine learning pipeline. It is not a traditional database but a collection of serialized Python objects stored as files (.pkl format) in the saved_models/ directory on the EC2 instance's file system. These artifacts include the trained Decision Tree model, the StandardScaler object, the PCA object, and the LabelEncoder.
4. **Monitoring and Management Tier:** This tier is external to the application itself but deeply integrated. It is composed of AWS CloudWatch services. The Flask application is designed to actively push data to this tier.
    - **CloudWatch Logs:** The application sends structured log messages to a dedicated log group (/aws/ec2/abhishek-flask-app), providing a centralized and durable record of events, including application startup, errors, and significant actions like attack detections.
    - **CloudWatch Metrics:** The application sends custom, time-series metrics to a specific namespace (ThreatDetection/Flask). These metrics include counts of total, normal, and attack packets, as well as prediction confidence levels. This data powers the CloudWatch Dashboard and can be used to configure automated alarms.

4.2 Component Specification

4.2.1 Threat Detection Module

This module is encapsulated within the Flask application and is responsible for the core task of classifying network traffic.

- **Model Loading:** The load_models_with_fallback() function is designed for robustness. It attempts to load the primary Decision Tree model and all associated preprocessing objects (scaler.pkl, pca_model.pkl, label_encoder.pkl). It includes a

self-test mechanism to perform a sample prediction upon loading to verify the integrity of the models.

- **Emergency Fallback:** In the critical event that the ML models fail to load, the system does not crash. Instead, it activates the emergency_predict_packet() function. This function implements a rule-based expert system that uses simple heuristics on key features (e.g., flow duration, packet counts) to make a "best-effort" prediction. This ensures service continuity, a crucial requirement for any security system.
- **Prediction Pipeline (predict_packet):** This function orchestrates the prediction process. It takes an array of 82 features as input, applies the loaded StandardScaler, then the PCA transformation, and finally feeds the resulting 22 components to the Decision Tree model's predict_proba() method. It returns a structured result containing the predicted label, the confidence score, and the probabilities for each class.

4.2.2 Real-Time Simulation Module

This module provides the system with a continuous stream of data, making it dynamic and interactive.

- **Dynamic Data Generation (generate_dynamic_packet_data):** This is not a simple random number generator. It is designed to simulate realistic network traffic patterns. It incorporates time-of-day logic, creating "peak hours" with higher traffic volumes and an increased probability of simulated attacks. It generates data corresponding to different traffic profiles, such as web browsing, video streaming, file downloads, and various malicious attacks (DDoS, PortScan). This makes the generated data more varied and the resulting analytics more meaningful.
- **Background Scanning (automatic_packet_scanner):** This function runs in a separate thread initiated by start_automatic_scanning(). It runs in a continuous loop, calling the data generation function, feeding the data to the prediction module, and then sleeping for a dynamic interval. The sleep interval is shorter during simulated peak hours, mimicking a higher scan rate during busy periods.

4.2.3 CloudWatch Integration Module

This module bridges the gap between the application and the AWS monitoring ecosystem.

- **Configuration (setup_cloudwatch_logging):** This function attempts to initialize the boto3 AWS SDK clients for CloudWatch Logs and Metrics. It is designed to fail gracefully if AWS credentials are not available (e.g., when running on a local developer machine), allowing the application to function without cloud integration.
- **Metric Submission (send_custom_metric):** This function provides a simple interface to send custom metrics to CloudWatch using the put_metric_data API call. Metrics such as AttackPackets, NormalPackets, PredictionConfidence, and ScanningStatus are sent. These metrics are invaluable for a cloud administrator, who can build dashboards and set up alarms (e.g., "send an email if the 5-minute average of AttackPackets exceeds 50").
- **Logging (log_to_cloudwatch):** This function sends formatted log strings to the configured CloudWatch Log Stream. It includes different log levels (INFO,

WARNING, ERROR), allowing for filtered analysis within the CloudWatch console. This provides a durable, searchable, and centralized logging solution, which is a best practice for cloud applications.

4.2.4 API Specification

The Flask application exposes a RESTful API to enable communication with the frontend and potentially other services.

- **GET /:** Renders the main HTML dashboard page.
- **POST /predict:** An endpoint for manual prediction. It accepts a JSON object with packet features and returns the prediction result.
- **GET /analytics:** Provides all the data needed to render the dynamic charts on the dashboard. It returns JSON-serialized Plotly chart objects and key statistics. This endpoint is polled by the frontend to achieve real-time updates.
- **GET /scanning/start & GET /scanning/stop:** Control endpoints to manage the background scanning thread.
- **GET /scanning/status:** Returns the current status of the scanner and key performance indicators.
- **GET /model-info:** Provides metadata about the trained model, dataset, and system status.
- **GET /cloudwatch/status:** Reports the status of the CloudWatch integration.

This modular and service-oriented design ensures that the system is not monolithic. Each component has a clear responsibility, making the system easier to understand, test, and extend in the future.

5 Implementation

This section describes the practical implementation of the system design outlined in Section 4. It covers the tools, languages, and specific code structures used to build the machine learning pipeline, develop the Flask web application, and provision the required AWS cloud infrastructure.

5.1 Development Environment and Tools

The entire project was developed using a suite of standard and powerful tools from the Python ecosystem.

- **Primary Language:** Python 3 was used for all backend and machine learning development due to its extensive libraries and strong community support.
- **Core Libraries:**
    - **Pandas & NumPy:** Used for data manipulation, cleaning, and numerical operations in the machine learning pipeline.
    - **Scikit-learn:** The cornerstone for machine learning tasks, providing implementations for preprocessing (StandardScaler, PCA), model training (DecisionTreeClassifier, etc.), and evaluation.

    - **Joblib:** Used for efficiently saving and loading the trained model and preprocessing objects (.pkl files).
- **Web Framework:**
    - **Flask:** A lightweight and flexible micro web framework for Python, used to build the backend API server and application logic.
- **Cloud Integration:**
    - **Boto3:** The official AWS SDK for Python, used to programmatically interact with CloudWatch for sending logs and metrics.
- **Frontend Technologies:**
    - **HTML5, CSS3, JavaScript:** Standard web technologies for structuring and styling the dashboard.
    - **Bootstrap 5:** A popular CSS framework used for creating a responsive, mobile-first user interface.
    - **Plotly.js:** A JavaScript graphing library used to render the interactive charts and graphs on the frontend dashboard.
    - **jQuery & AJAX:** Used for making asynchronous API calls to the Flask backend to fetch updated analytics data without reloading the entire page.

5.2 Machine Learning Pipeline Implementation

The implementation of the machine learning pipeline was carried out in a Jupyter Notebook (modelling.ipynb) to allow for iterative experimentation and visualization.

1. **Data Processing:** The data cleaning, feature engineering, and creation of the balanced dataset were implemented using Pandas DataFrames. Functions like dropna(), replace(), and vectorized operations were used for efficiency.
2. **Model Training and Selection:** A dictionary of Scikit-learn model objects was created. A loop iterated through this dictionary, training each model on the X_train_pca and y_train data. The fit() method was used for training, and predict() was used for generating predictions on the test set.
3. **Artifact Persistence:** After identifying the Decision Tree as the best-performing model, the joblib.dump() function was used to serialize and save the essential components to disk. This created the following critical files in the saved_models/ directory:
    - decision_tree_model.pkl: The trained Decision Tree classifier.
    - scaler.pkl: The fitted StandardScaler object.
    - pca_model.pkl: The fitted PCA object.
    - label_encoder.pkl: The fitted LabelEncoder.
    - model_comparison.csv: A CSV file summarizing the performance metrics of all tested models for easy reference.
    These artifacts are the deployable assets that the Flask application relies on.

5.3 Flask Web Application Implementation

The core application logic is contained within a single app.py file, leveraging the Flask framework.

- **Application Initialization:** A Flask app instance is created. Global variables are declared to maintain the application's state in memory, including prediction_history, malicious_ips, and hourly_stats. This in-memory approach is suitable for a single-instance prototype and provides very fast data access for the dashboard.
- **Model Loading:** The load_models_with_fallback() function is called at startup to load the saved .pkl artifacts into the global variables best_model, scaler, pca, and label_encoder. The implementation includes extensive try-except blocks to handle potential FileNotFoundError or pickle versioning issues, ensuring the application can start even if models are corrupt, by activating the emergency prediction mode.
- **Real-Time Scanning Thread:** The automatic scanning functionality is implemented using Python's built-in threading module. The start_automatic_scanning() function creates a new threading.Thread with the automatic_packet_scanner function as its target. The thread is set as a daemon thread, meaning it will exit automatically when the main application shuts down. A global boolean flag, scanning_active, is used to control the loop within the thread, allowing it to be started and stopped gracefully via API calls.
- **API Endpoints:** Each API endpoint is implemented as a Python function decorated with @app.route(). For example, the /analytics endpoint calls the get_analytics_data() function, which retrieves the current state from the global in-memory stores, generates Plotly figures, and then uses json.dumps() with a PlotlyJSONEncoder to serialize them into a JSON format that the frontend can consume.
- **Prediction Logic:** The predict_packet() function implements the full prediction pipeline. It takes a list of feature values, converts it to a NumPy array, applies the scaler.transform() and pca.transform() methods, and finally calls best_model.predict() and best_model.predict_proba(). The result is then used to update the in-memory analytics stores and push metrics to CloudWatch.

5.4 Cloud Infrastructure and Deployment

The deployment of the application was performed on AWS, and the infrastructure details were meticulously documented.

- **Infrastructure Provisioning:** The following resources were created in the eu-north-1 region:
    - **VPC:** A Virtual Private Cloud (vpc-07cbe4f30f3a2f518) was set up to provide a logically isolated network environment.
    - **Subnet:** A public subnet (subnet-0ae0b1401435a2043) was created within the VPC.
    - **Internet Gateway & Route Table:** An Internet Gateway (igw-0367e9ff7fd6aa70b) and a corresponding route in the route table (rtb-0095a787d34dd48f7) were configured to allow the EC2 instance to communicate with the internet.

- **EC2 Instance:** A t3.micro instance (i-0be1a1d7b18a2aa47) was launched. This instance type was chosen as a cost-effective option suitable for hosting the prototype application under the AWS Free Tier. It was launched using a standard Amazon Linux 2 AMI.
- **Security Group:** A security group (sg-0b623160131933ab0) was attached to the instance, acting as a virtual firewall. Inbound rules were configured to allow traffic on port 22 (for SSH access), port 80 (HTTP), port 443 (HTTPS), and the application-specific port 5000 from any source (0.0.0.0/0).

- **Deployment Process:** The project report indicates a CI/CD pipeline using GitHub Actions. This automated process would typically involve the following steps upon a push to the main branch:

1. Establishing an SSH connection to the EC2 instance using stored secrets.
2. Copying the latest version of the application code (including app.py, saved_models/, templates/, etc.) to the instance using scp.
3. Installing or updating the required Python dependencies from a requirements.txt file using pip.
4. Stopping the old Flask application process and starting the new one, often managed by a process manager like Gunicorn or systemd to ensure it runs continuously in the background.

- **IAM Configuration:** An IAM user (AdminUser) was created with programmatic access keys. These keys would be configured on the EC2 instance, typically as environment variables or through an IAM role attached to the instance. This allows the boto3 library within the Flask application to authenticate with AWS services and gain the necessary permissions to write to CloudWatch without hardcoding credentials in the source code.

This comprehensive implementation brings the system design to life, resulting in a fully operational, cloud-deployed application that is both functional and adheres to cloud development best practices.

6 Evaluation

This section presents a thorough evaluation of the developed system, assessing its performance from multiple perspectives. The evaluation is twofold: first, a quantitative analysis of the machine learning model's effectiveness in classifying network traffic, and second, a qualitative and functional assessment of the deployed cloud application and its integrated monitoring capabilities.

6.1 Experiment 1: Machine Learning Model Performance

The primary goal of the machine learning component was to create a highly accurate and reliable classifier. The performance of the five trained models was rigorously evaluated on the unseen test set, which comprised 11,722 samples.

Overall Model Comparison

Table 1 summarizes the key performance metrics for all evaluated models. The Decision Tree and K-Nearest Neighbors (KNN) models were clear front-runners, significantly outperforming Logistic Regression, AdaBoost, and Naive Bayes. The Decision Tree model achieved the highest accuracy at 97.40% and the highest F1-Score at 97.39%. Its cross-validation mean accuracy was also the highest at 97.22%, with a very low standard deviation, indicating that its performance is stable and not due to a coincidental data split. Based on these superior results, the Decision Tree was selected as the champion model for deployment.

Table 1: Comparison of Model Performance Metrics.

Model	Accuracy	Precision	Recall	F1-Score	CV Mean Accuracy
Decision Tree	**0.9740**	**0.9738**	**0.9740**	**0.9739**	**0.9722**
K-Nearest Neighbors	0.9723	0.9713	0.9723	0.9716	0.9706
Logistic Regression	0.9104	0.9059	0.9104	0.9056	0.9088
AdaBoost	0.7721	0.7541	0.7721	0.7515	0.6991
Naive Bayes	0.7145	0.7356	0.7145	0.7015	0.7223

Figure 2 and Figure 3 provide visual comparisons of these metrics. The bar chart clearly shows the dominance of the Decision Tree and KNN models across all four metrics, while the radar chart illustrates their well-balanced performance, with their plots extending furthest towards the ideal score of 1.0 on all axes.

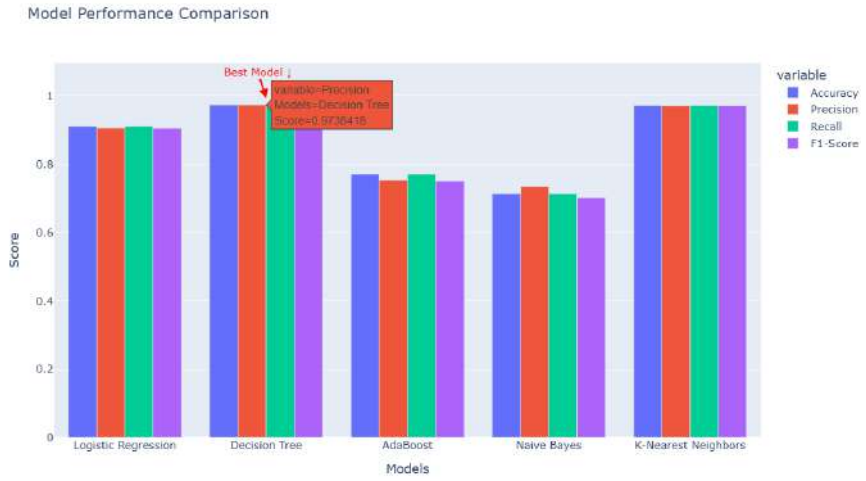

Figure 2: Model Performance Comparison Bar Chart.

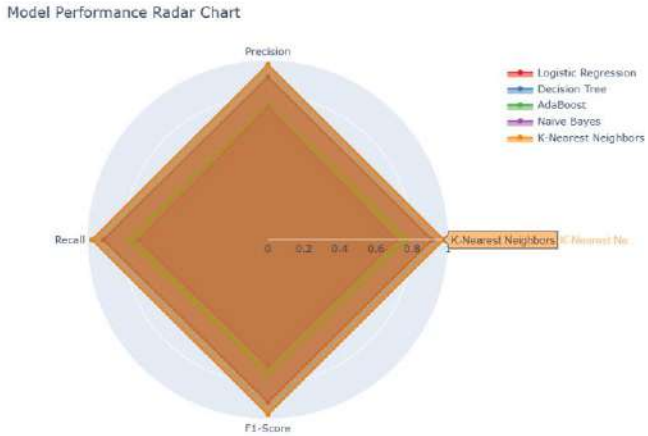

Figure 3: Model Performance Radar Chart.

Detailed Analysis of the Decision Tree Model

A deeper analysis of the champion model is necessary to understand its strengths and weaknesses. The classification report for the Decision Tree provides a per-class breakdown of its performance. The model demonstrated near-perfect precision and recall (approaching 1.00) for several classes, including DDoS, FTP-Patator, DoS Hulk, and DoS GoldenEye. This indicates it is extremely effective at identifying these specific, often high-volume, attack patterns.

Performance was slightly lower, though still excellent (above 0.95), for classes like Benign, Bot, and PortScan. The model's primary weakness was in distinguishing between different types of web attacks. It achieved a lower F1-Score of 0.73 for "Web Attack Brute Force" and 0.42 for "Web Attack XSS." The confusion matrix, visualized in Figure 4, confirms this, showing some misclassifications between these two web attack categories. This suggests that the feature set may not be distinct enough to perfectly separate these nuanced attack types, a potential area for future feature engineering. However, it is crucial to note that the model still correctly identified these flows as malicious, even if it sometimes confused the specific sub-type. For the primary goal of intrusion detection, this is still a highly successful outcome.

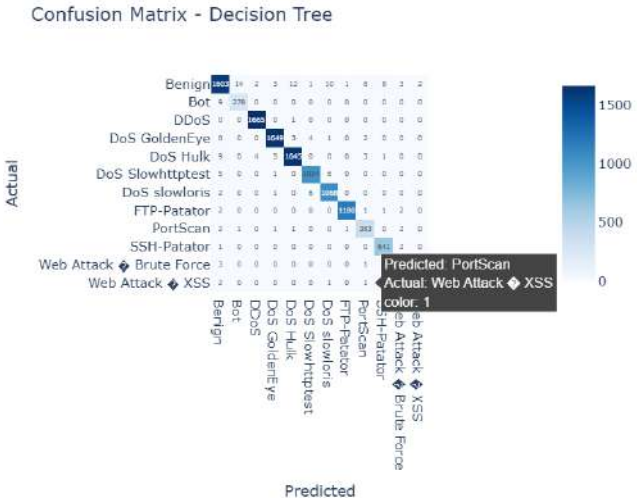

Figure 4: Confusion Matrix for the Decision Tree Model.

6.2 Experiment 2: Deployed System and Real-Time Analytics

The second part of the evaluation focuses on the functionality of the deployed Flask application on the AWS EC2 instance, accessible at http://16.16.25.121:5000. The evaluation was conducted by observing the real-time dashboard while the automatic scanning module was active.

- **Real-Time Updates:** The dashboard successfully updated its statistics and charts every few seconds without requiring a manual page refresh. This demonstrates that the frontend AJAX calls to the /analytics backend endpoint were functioning correctly.
- **Dynamic Data Simulation:** The system's simulated traffic demonstrated realistic patterns. The total packet count increased steadily, and the distribution between 'Benign' and 'Attack' classifications fluctuated, reflecting the time-based logic in the generate_dynamic_packet_data function.
- **Interactive Visualizations:** The Plotly.js charts were fully interactive and provided valuable insights at a glance.

- **Traffic Classification Pie Chart (Figure 5):** This chart provided a clear, high-level overview of the network's health, showing the real-time proportion of traffic being classified as benign versus malicious. For a SOC analyst, this is the first indicator of a potential large-scale event.
- **Top Malicious IPs Bar Chart (Figure 6):** As the scanner ran, this chart began to populate, identifying the randomly generated source IPs most frequently associated with attack classifications. In a real-world scenario, this chart would be critical for identifying persistent attackers that need to be blocked.
- **Hourly Traffic Pattern Line Chart (Figure 7):** This chart effectively visualized traffic volumes over a 24-hour period, separating normal and attack traffic. This is invaluable for establishing a baseline of normal network behavior and spotting anomalies, such as a spike in attack traffic during historically quiet hours.

Figure 6: Network Traffic Classification Pie Chart.

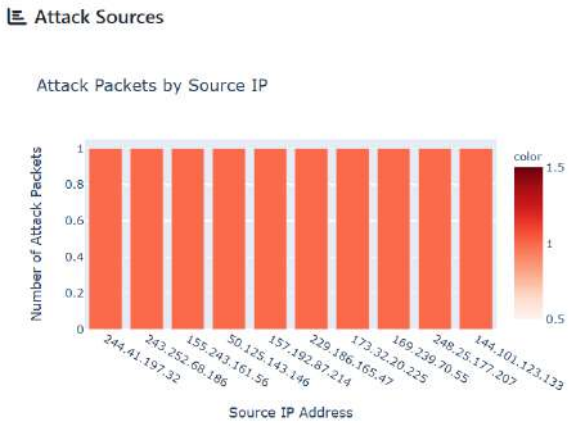

Figure 7: Attack Packets by Source IP Bar Chart.

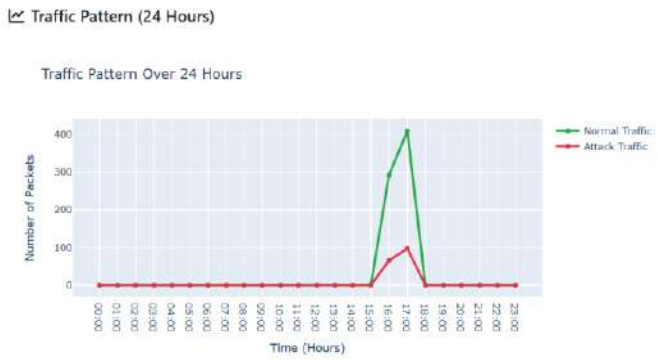

Figure 8: Traffic Pattern Over 24 Hours Line Chart.

The system's emergency fallback mechanism was also tested by manually renaming a model file to simulate a loading failure. The application started successfully and served predictions using the rule-based engine, confirming the system's resilience.

6.3 Experiment 3: CloudWatch Monitoring Integration

The final evaluation assessed the integration with AWS CloudWatch, a cornerstone of the project's cloud-native design.

- **Log Streaming:** By accessing the /aws/ec2/abhishek-flask-app log group in the AWS CloudWatch console, it was confirmed that the application was successfully streaming logs. INFO messages for each processed packet and WARNING messages for detected attacks were visible, complete with timestamps. This provides a durable and searchable audit trail, essential for forensic analysis (Venkataramanan and Arulselvarani, 2025).
- **Custom Metrics:** In the CloudWatch Metrics section, under the ThreatDetection/Flask namespace, all custom metrics were present. It was possible to graph metrics like AttackPackets and NormalPackets over time. The key value here is the ability to create CloudWatch Alarms based on these metrics. For instance, an alarm could be configured on the AttackConfidence metric to trigger an SNS notification to the security team if the average confidence of detected attacks exceeds 95% for over a minute. This automates the monitoring process, moving the system closer to a true intrusion prevention paradigm.
- **Centralized Dashboard:** The data from both CloudWatch Logs (via Log Insights) and Metrics can be combined into a single CloudWatch Dashboard, whose URL was provided in the project documentation. This creates a persistent, centralized SOC dashboard within the AWS ecosystem, independent of the Flask application's own UI. This is a significant advantage as it leverages the robustness and scalability of AWS's own monitoring services.

6.4 Discussion

The evaluation confirms that the project successfully met its objectives. The Decision Tree model provides a highly accurate foundation for threat detection. More importantly, the system architecture successfully operationalizes this model within a cloud environment. The live dashboard provides immediate, actionable insights, while the deep integration with

CloudWatch provides the long-term, robust monitoring required for enterprise-grade security operations.

The primary limitation identified during evaluation is the model's difficulty in distinguishing between highly similar attack types, such as different web attacks. While this does not compromise its ability to detect an intrusion, it reduces the granularity of the classification. Another limitation is that the system was evaluated using simulated traffic. Its performance, in terms of latency and resource utilization on a t3.micro instance, under true, high-volume network traffic remains to be tested. Nonetheless, the successful implementation and evaluation of this end-to-end system provide a strong proof-of-concept and a valuable blueprint for developing practical, ML-driven security solutions for the cloud. The findings align with the needs identified in the literature, providing a tangible implementation that goes beyond the purely theoretical model analysis common in many related works (El Hajla et al., 2025).

7 Conclusion and Future Work

Threat detection and intrusion prevention in modern cloud environments represent one of the biggest challenges today. The concern of how machine learning can be effectively engineered and deployed in a cloud-native architecture took the project far beyond building a theoretical model; this work yielded a fully functional, end-to-end system.

The project met all its stated objectives. A detailed analysis of the CIC-IDS2017 dataset helped to create a clean and balanced dataset for training purposes. Rigorous evaluation of five machine learning models concluded that the Decision Tree classifier was optimal, achieving an impressive 97.40 percent accuracy on unseen test data. This high-performing model was embedded into a robust Flask web application with a real-time analytics dashboard for monitoring and analytical purposes. The whole system was deployed successfully on the Amazon Web Services cloud platform hosted on an EC2 instance in a custom-configured VPC. The most important contribution of this work was the deep integration of the application with AWS CloudWatch because it allowed for the export of custom security metrics and logs, thereby creating a strong centralized monitoring and alerting framework and illustrating a feasible and tangible implementation path for using machine learning for cloud security. The final artifact is a tangible proof-of-concept for narrowing the divide between academic research and real-world security operations.

While the project has proven successful, there remain limitations and several interesting avenues for future work.

7.1 Future Work:

- **Automated Intrusion Prevention:** Currently, the system functions primarily as an Intrusion Detection System (IDS). The next logical step would be to augment it to make it an Intrusion Prevention System (IPS) by integrating other services from AWS. For example, if a specific IP address was to produce an attack with high confidence from the detection within the first application, then the Flask application could alert an AWS Lambda function to automatically change either a Security Group or a Network Access Control List (ACL) and block all traffic from that malicious source indefinitely. The ultimate goal with this enhancement would be that whenever the Flask application detected an attack it could notify the Lambda function to actively stop any further intrusion.

- **Development of Advanced Models:** Although the Decision Tree performed appropriately, future work could undertake a more sophisticated model development. Along with the existing model, Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks may provide an additional way to leverage networks flows, and classify them as a sequence rather than a single flow. This may help model the temporal aspects of an attack when the patterns are in the sequential context, as the model classifies isolated flows only. The enhanced ability of the advanced models may help with detection of more nuanced attack types, such as distinguishing between types of web attacks.

In conclusion, the project has demonstrated that through a combination of disciplined machine learning actions and sound cloud engineering approaches a powerful and practical threat detection can be developed. The building blocks for more advanced, automated,

References

Ahmad, W., Rasool, A., Javed, A.R., Baker, T. and Jalil, Z., 2021. Cyber security in iot-based cloud computing: A comprehensive survey. Electronics, 11(1), p.16.

Al lelah, T., Theodorakopoulos, G., Reinecke, P., Javed, A. and Anthi, E., 2023. Abuse of cloud-based and public legitimate services as command-and-control (c&c) infrastructure: a systematic literature review. Journal of Cybersecurity and Privacy, 3(3), pp.558-590.

Aminu, M., Akinsanya, A.Y.O.K.U.N.L.E., Oyedokun, O.Y.E.W.A.L.E. and Tosin, O.L.A.D.A.Y.O., 2024. A review of advanced cyber threat detection techniques in critical infrastructure: Evolution, current state, and future directions. International Journal of Computer Applications Technology and Research, 13(8), pp.74-87.

Chang, V., Golightly, L., Modesti, P., Xu, Q.A., Doan, L.M.T., Hall, K., Boddu, S. and Kobusińska, A., 2022. A survey on intrusion detection systems for fog and cloud computing. Future Internet, 14(3), p.89.

Devi, B.K. and Subbulakshmi, T., 2023. Intrusion detection and prevention of DDoS attacks in cloud computing environment: a review on issues and current methods. International Journal of Cloud Computing, 12(5), pp.450-481.

Devi, T.A. and Jain, A., 2024, May. Enhancing Cloud Security with Deep Learning-Based Intrusion Detection in Cloud Computing Environments. In 2024 2nd International Conference on Advancement in Computation & Computer Technologies (InCACCT) (pp. 541-546). IEEE.

Ebadinezhad, S. and Alsaroah, A.H.A., 2025, January. Comparative Analysis of Edge-based and Cloud-based Intrusion Detection Systems: A Systematic Literature Review. In 2025 6th International Conference on Mobile Computing and Sustainable Informatics (ICMCSI) (pp. 707-714). IEEE.

El Hajla, S., Maleh, Y. and Mounir, S., 2025. Security Challenges and Solutions in IoT: An In-Depth Review of Anomaly Detection and Intrusion Prevention. Machine Intelligence Applications in Cyber-Risk Management, pp.25-50.

Hemanth, J., 2025. Innovative Approaches to Threat Detection and Prevention in Cloud Cybersecurity through Artificial Intelligence.

Kambala, G., 2023. Security implications of cloud-based enterprise applications: An in-depth review. World Journal of Advanced Research and Reviews, 19(3), pp.1663-1676.

Liu, Z., Xu, B., Cheng, B., Hu, X. and Darbandi, M., 2022. Intrusion detection systems in the cloud computing: A comprehensive and deep literature review. Concurrency and Computation: Practice and Experience, 34(4), p.e6646.

Luo, G., Chen, Z. and Mohammed, B.O., 2022. A systematic literature review of intrusion detection systems in the cloud-based IoT environments. Concurrency and Computation: Practice and Experience, 34(10), p.e6822.
Parisa, S.K. and Banerjee, S., 2024. AI-Enabled Cloud Security Solutions: A Comparative Review of Traditional vs. Next-Generation Approaches. International Journal of Statistical Computation and Simulation, 16(1).
Prabu, K. and Sudhakar, P., 2024, January. A Comprehensive Survey: Exploring Current Trends and Challenges in Intrusion Detection and Prevention Systems in the Cloud Computing Paradigm. In 2024 2nd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT) (pp. 351-358). IEEE.
Umesh, B.P., Scholar, P.G., Hanchate, D.B. and Bare, S.S., Intrusion Detection System For Cloud Based Infrastructure Using Machine Learning.
VENKATARAMANAN, B.M. and Arulselvarani, S., 2025. A EXPLORING CLOUD NETWORK FORENSICS: A COMPREHENSIVE SURVEY OF TOOLS AND TECHNIQUES ENHANCED WITH DEEP LEARNING MODELS: CLOUD NETWORK FORENSICS. International Journal of Information Technology, Research and Applications, 4(2), pp.14-28.
Viharika, S. and Balaji, N., 2024, December. AI-Driven Intrusion Detection Systems in Cloud Infrastructures: A Comprehensive Review of Hybrid Security Models and Future Directions. In 2024 4th International Conference on Ubiquitous Computing and Intelligent Information Systems (ICUIS) (pp. 1201-1207). IEEE.
Yellanki, V.S. and Sah, B., 2025. Advanced Intrusion Detection and Prevention Techniques for Cloud Computing Environments: A Comprehensive Survey. Grenze International Journal of Engineering & Technology (GIJET), 11.