A

Project Report

On

Real-time anomaly detection in IoT networks

Submitted in partial fulfillment of the requirements

for the award of the degree of

Master of Technology

in

Information Technology

by

Prachee Singh (MSE2023006)

Sakshi Agrawal (MSE2023004)

III Semester

Under the Supervision of

Dr. Sonali Agarwal

Department of Information Technology

Indian Institute of Information Technology, Allahabad

Devghat, Jhalwa, Prayagraj- 211015 (UP) INDIA

November, 2024

ABSTRACT

The exponential growth in the deployment of Internet of Things (IoT) devices has led to a surge in cyber threats, necessitating the development of efficient and scalable solutions for real-time anomaly detection. This project addresses the challenges associated with securing IoT networks by leveraging a combination of the Hadoop-Spark ecosystem and Apache Kafka for real-time data streaming. The primary objective is to detect anomalous behavior within network traffic to mitigate potential cyber-attacks.

The project utilizes machine learning models, specifically **Random Forest**, **Logistic Regression**, and **XGBoost**, to classify network events as either normal or attacks. After rigorous evaluation, the Random Forest model was selected as the optimal choice due to its impressive performance metrics: **99.63% accuracy**, **99.63% precision**, **99.63% recall**, and **99.63% F1-score**. These results demonstrate the model's robustness and reliability in identifying anomalous network activities with minimal false positives and negatives.

The integrated solution includes real-time data ingestion using Kafka, processing and classification using Spark, and a Flask-based dashboard for visualizing detection results. This scalable architecture ensures that the system can efficiently handle large volumes of IoT data in real time. Future enhancements will focus on optimizing the model with neural networks and deploying the system on cloud platforms for enhanced scalability and performance. This project serves as a comprehensive solution for securing IoT networks by providing real-time, high-accuracy anomaly detection.

Keywords: IoT Networks, Real-time Anomaly Detection, Hadoop-Spark, Apache Kafka, Random Forest, Machine Learning, Network Security, Data Streaming, Scalability, Cyber Threats

INTRODUCTION

Problem Statement

The exponential growth of IoT devices necessitates efficient and scalable real-time anomaly detection systems to ensure network security. This project aims to develop such a system using the Hadoop-Kafka framework, capable of flagging anomalous and non-anomalous events in real-time.

The rapid proliferation of Internet of Things (IoT) devices has transformed industries and everyday life by enabling interconnected systems that improve efficiency, automation, and convenience. However, the exponential increase in these devices has also created new vulnerabilities in network security, making IoT networks prime targets for cyber-attacks. Traditional network security systems often struggle to keep up with the massive data generated by IoT devices, leading to delayed threat detection and potential breaches.

In IoT networks, the detection of anomalies—unusual patterns in network traffic—can be a critical indicator of potential security threats. These threats include Denial of Service (DoS) attacks, data exfiltration, reconnaissance attempts, and other malicious activities. However, the sheer volume and velocity of data generated by IoT networks make real-time anomaly detection a challenging task. This project addresses these challenges by developing a scalable, efficient, and real-time anomaly detection system using a combination of the Hadoop-Spark ecosystem and Apache Kafka.

Background and Motivation

With billions of IoT devices in operation, securing these networks has become more urgent than ever. IoT devices often have limited computational power and are deployed in environments where they are vulnerable to attacks. The consequences of a security breach in IoT networks can be severe, ranging from financial losses to compromising critical infrastructure, especially in sectors like healthcare, manufacturing, and smart cities.

Traditional intrusion detection systems are not designed to handle the high-speed, high-volume data streams characteristic of IoT networks. These systems often rely on manual analysis, which is time-consuming and not scalable. Additionally, they are prone to high false-positive rates, making them unreliable for real-time threat detection. To address these limitations, there is a growing need for automated, real-time anomaly detection systems that can efficiently analyze large volumes of streaming data to identify and mitigate threats as they occur.

Project Overview

This project aims to develop a robust, scalable solution for real-time anomaly detection in IoT networks by leveraging big data technologies such as Hadoop, Spark, and Kafka. By integrating these technologies, we can efficiently process large volumes of network data to detect anomalies in real-time, ensuring timely response to potential cyber threats.

The system is designed to handle large-scale datasets using distributed computing frameworks. It uses Apache Kafka for real-time data streaming and ingestion, while Hadoop's HDFS (Hadoop Distributed File System) provides scalable storage. Spark, a high-performance data processing engine, enables the system to classify network traffic in real time using machine learning algorithms.

Objectives

The project is driven by the following key objectives:

1. **Develop a Scalable Anomaly Detection Model**: Build a system that can handle large-scale IoT datasets efficiently, leveraging distributed computing frameworks.
2. **Compare Machine Learning Models**: Evaluate the performance of different machine learning models (Random Forest, Logistic Regression, and XGBoost) to identify the most effective algorithm for anomaly detection.
3. **Monitor Real-Time Detection**: Implement a real-time dashboard for continuous monitoring of network traffic, allowing for quick identification and response to anomalies.

Significance of the Study

The successful implementation of this system can significantly enhance the security of IoT networks by providing real-time insights into network traffic behavior. The solution is not only scalable but also adaptable to various IoT environments, making it suitable for deployment across multiple sectors. By leveraging distributed computing frameworks, the project addresses the limitations of existing systems and offers a practical solution for real-time anomaly detection, reducing the time to detect and respond to potential threats.

This project, therefore, contributes to the field of cybersecurity by proposing a scalable, real-time approach for anomaly detection that is both efficient and effective in securing IoT networks. Future work will focus on further optimizing the system using deep learning models and deploying it in cloud and edge environments to achieve even greater scalability and performance.

LITERATURE SURVEY

Idrissi (2021)
Idrissi proposed a Convolutional Neural Network (CNN)-based intrusion detection system named BotIDS, specifically designed to detect botnet attacks in IoT networks. The study utilized the Bot-IoT dataset to train the CNN model, which was then compared against other deep learning models like Recurrent Neural Networks (RNN), Long Short-Term Memory networks (LSTM), and Gated Recurrent Units (GRU). The CNN model outperformed these alternatives, achieving a validation accuracy of **99.94%** with a prediction time of just **0.34 milliseconds**, highlighting its efficiency in real-time detection scenarios. The system was implemented using Keras and TensorFlow frameworks, making it highly effective for multi-class classification tasks in IoT security.

Hadj-Attou (2024)
This study aimed to tackle the issue of class imbalance in the Bot-IoT dataset, which often hampers the effectiveness of intrusion detection systems. The proposed solution involved the use of a big data framework leveraging Apache Spark and Decision Trees, with a focus on weighted average ensemble methods. By addressing the imbalance problem, this approach significantly improved the detection rates of minority attack classes, which are typically harder to identify. The framework demonstrated superior performance over traditional methods, making it a robust solution for large-scale IoT networks prone to skewed data distributions.

Motylinski (2022)
Motylinski's research focused on accelerating machine learning models for detecting botnet attacks in IoT environments using GPU technology. The study included extensive preprocessing and feature selection on the IoT-Bot dataset, followed by the application of GPU-accelerated algorithms like Random Forest (RF), K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Logistic Regression (LR). The results were remarkable, achieving over **99%** in accuracy, precision, recall, and F1-score, along with significantly reduced training and prediction times due to GPU acceleration. The use of RAPIDS libraries (cuDF, cuML) in conjunction with Python provided a scalable solution for real-time IoT security applications.

Alheeti (2021)
Alheeti developed an Intrusion Detection System (IDS) using deep learning models to secure IoT networks from various cyber-attacks. The study employed both normal and fuzzified datasets for evaluation, using deep neural networks to detect intrusions. The IDS achieved **99.30%** accuracy with the normal dataset and **99.42%** accuracy with the fuzzified dataset, demonstrating its robustness in handling data variations. This work underscores the effectiveness of deep learning techniques in enhancing the resilience of IoT networks against sophisticated cyber threats.

Popoola (2021)
In an effort to improve botnet detection in smart home environments, Popoola proposed the use of a Stacked Recurrent Neural Network (SRNN). The model was trained on the Bot-IoT dataset to learn hierarchical representations of imbalanced network traffic. The SRNN model was compared with traditional RNNs and proved to

be more effective, achieving higher classification accuracy and better handling of class imbalances. By leveraging the Adam optimizer, this method provided a robust solution for real-time anomaly detection in smart home IoT systems.

Abushwereb (2022)
Abushwereb's study focused on the development of an IoT intrusion detection system using Apache Spark to efficiently handle large-scale datasets. The Bot-IoT dataset was used to evaluate the performance of different machine learning models, particularly Random Forest and Decision Tree algorithms, for both binary and multi-class classification tasks. The system achieved a **99.7%** F1-score for binary classification and an **88.5%** F1-score for subclass classification, showcasing its capability to accurately detect a wide range of IoT network threats. The use of Apache Spark's MLlib enabled scalable model training, making the system suitable for real-time deployment.

This detailed literature survey encapsulates the objectives, methodologies, and results from recent studies on intrusion detection using machine learning and big data frameworks, emphasizing their relevance to your current project on real-time anomaly detection in IoT networks.

METHODOLOGY

Phase 1: Data Collection

The first step in our methodology involved acquiring the **Bot-IoT dataset** from Kaggle. This dataset was chosen because it includes a diverse range of network traffic records, simulating both normal operations and various types of cyber-attacks, such as Denial of Service (DoS) and reconnaissance activities. The dataset consists of over **70 million records**, making it ideal for testing the scalability of big data processing frameworks like Hadoop and Spark. The comprehensive nature of the dataset ensured that our anomaly detection models were exposed to a wide range of attack patterns, thereby enhancing their generalization capabilities.

Phase 2: Data Preprocessing

Given the large volume of data, efficient preprocessing was critical to ensure the smooth functioning of subsequent phases. The dataset was initially cleaned to remove any missing or corrupt entries that could potentially bias the model training process. We standardized features to ensure consistency, as raw network data often includes attributes with varying scales (e.g., packet size, duration, and byte counts). Techniques such as **one-hot encoding** were used for categorical variables, and **feature scaling** was applied to numerical features to bring them to a comparable range. Additionally, redundant features were removed to reduce dimensionality and improve model efficiency.

Phase 3: Feature Engineering

Feature engineering played a pivotal role in optimizing model performance. By carefully selecting and extracting key attributes from the dataset, we enhanced the model's ability to differentiate between normal and anomalous network traffic. Features such as **protocol type, packet counts, flags, bytes, and duration** were found to be particularly indicative of attack patterns. We also created new features based on domain knowledge to capture complex relationships within the data, thereby improving model accuracy. Feature importance analysis, especially using Random Forest, helped us identify the most impactful variables, which were then prioritized during model training.

Phase 4: Algorithm Selection

To identify the best-performing model for real-time anomaly detection, we experimented with several machine learning algorithms, including **Logistic Regression, Random Forest, and XGBoost**. Each of these models was trained and evaluated on the preprocessed dataset to compare their performance. Logistic Regression was used as a baseline due to its simplicity, while Random Forest and XGBoost were chosen for their robustness and ability to handle large, complex datasets. We evaluated these models based on metrics like **accuracy, precision, recall, and F1-score** to determine their suitability for our use case. Ultimately, Random Forest emerged as the optimal choice due to its high accuracy and balanced performance across all evaluation metrics.

Phase 5: Real-time Data Processing

A key objective of the project was to enable real-time anomaly detection, which required the integration of **Apache Kafka** for streaming data ingestion. Kafka was employed to capture live network traffic data and stream it into the **Hadoop-Spark ecosystem** for processing. The choice of Kafka was driven by its ability to handle high-throughput data streams with low latency, making it ideal for real-time applications. Once the data was ingested, **Spark** was used to apply the pre-trained machine learning model to classify incoming network traffic as normal or anomalous. The distributed nature of Spark ensured that the system could scale efficiently as the volume of data increased.

Phase 6: Dashboard Development

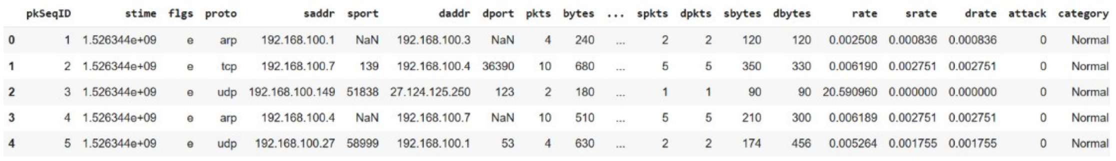
To provide an intuitive interface for monitoring real-time detection results, we developed a dashboard using **Flask**, a lightweight Python web framework. The dashboard displays key metrics and classifications, allowing users to monitor network traffic in real-time. The interface includes a table view where network attributes, such as **sensor ID, flags, protocol type, and prediction status (Normal/Attack)**, are displayed. The dashboard is updated dynamically, providing immediate feedback on detected anomalies. The real-time visualizations and alerts help administrators quickly identify and respond to potential threats, thereby enhancing the security of IoT networks.

Conclusion of Methodology

By combining the power of distributed computing frameworks like Hadoop and Spark with real-time data streaming through Kafka and a Flask-based dashboard for visualization, our methodology ensures that the system is both scalable and efficient. This end-to-end pipeline effectively addresses the challenges of handling large IoT datasets while providing accurate and timely anomaly detection, making it a comprehensive solution for securing IoT networks.

Dataset (Bot_IoT)

The Bot_IoT dataset plays a crucial role in our project by providing a realistic simulation of IoT network traffic, capturing both normal operations and various types of cyber-attacks. The dataset, sourced from Kaggle, is designed to aid in the development and evaluation of intrusion detection systems in IoT environments. It is structured to include an extensive range of network traffic data that reflects real-world IoT deployments. Below is a detailed breakdown of the key features, attributes, and challenges associated with this dataset.



	pkSeqID	stime	flgs	proto	saddr	sport	daddr	dport	pkts	bytes	...	spkts	dpkts	sbytes	dbytes	rate	srate	drate	attack	category
0	1	1.526344e+09	e	arp	192.168.100.1	NaN	192.168.100.3	NaN	4	240	...	2	2	120	120	0.002508	0.000836	0.000836	0	Normal
1	2	1.526344e+09	e	tcp	192.168.100.7	139	192.168.100.4	36390	10	680	...	5	5	350	330	0.006190	0.002751	0.002751	0	Normal
2	3	1.526344e+09	e	udp	192.168.100.149	51838	27.124.125.250	123	2	180	...	1	1	90	90	20.590960	0.000000	0.000000	0	Normal
3	4	1.526344e+09	e	arp	192.168.100.4	NaN	192.168.100.7	NaN	10	510	...	5	5	210	300	0.006189	0.002751	0.002751	0	Normal
4	5	1.526344e+09	e	udp	192.168.100.27	58999	192.168.100.1	53	4	630	...	2	2	174	456	0.005264	0.001755	0.001755	0	Normal

a. Dataset Overview

- **Source**: The Bot_IoT dataset is publicly available on Kaggle, making it an accessible resource for researchers working on IoT network security.
- **Description**: It is a comprehensive dataset that simulates network traffic data, including both benign (normal) activities and malicious attacks. The dataset is generated using IoT devices and network configurations to mimic real-world IoT environments.
- **Size**: With over **70 million records** in CSV format, this dataset is exceptionally large, posing challenges in terms of storage, processing, and efficient handling. The sheer volume of data makes it ideal for evaluating the scalability of big data frameworks like Hadoop and Spark.

b. Key Features and Attributes

The Bot_IoT dataset includes a diverse set of features that are essential for training machine learning models to distinguish between normal and malicious network traffic. Here are some of the most significant attributes:

1. **Source IP Address**: The IP address of the device that initiated the connection. This feature can help identify potential attackers based on suspicious IP ranges.
2. **Destination IP Address**: The IP address of the target device, which helps in tracking which devices are being targeted in the network.
3. **Source Port**: The port number used by the source device to establish the connection, which can indicate the type of application or service being used.
4. **Destination Port**: The target port on the destination device. Certain ports are commonly associated with specific types of attacks, such as DoS or reconnaissance.
5. **Protocol Type**: Specifies the network protocol used, such as TCP, UDP, or ICMP. Certain protocols may be more susceptible to specific types of attacks.
6. **Packet Count**: The total number of packets sent during a connection. Unusually high packet counts could indicate flooding attacks, such as DoS.

7. **Bytes Sent/Received**: The total number of bytes transmitted in each direction during the session. Large data transfers can be indicative of data exfiltration attempts.
8. **Duration**: The total time (in seconds) for which the connection lasted. Abnormally long or short connection durations can be indicative of certain types of attacks.
9. **Flags**: Network flags such as SYN, ACK, FIN, etc., which are used to control data flow in TCP connections. Anomalous flag patterns can indicate potential attacks like SYN floods.
10. **Packet Size**: The average size of packets sent during the connection. Deviations in expected packet sizes may signal malicious activities.
11. **Flow Count**: Represents the number of network flows between the source and destination IP addresses within a given time frame.
12. **Attack Label**: The dataset includes a label indicating whether a given network event is normal or belongs to one of several attack categories (e.g., DoS, DDoS, reconnaissance, keylogging, data theft).
13. **Class**: A higher-level classification that differentiates between normal traffic and various types of malicious attacks. (TARGET VARIABLE)

c. Types of Attacks Covered

The dataset is designed to simulate a wide range of attack scenarios, which are crucial for training and testing anomaly detection systems. Some of the common attack types included are:

- **Denial of Service (DoS)**: Attacks that flood a network to exhaust its resources, making it unavailable to legitimate users.
- **Reconnaissance**: Includes scanning activities aimed at gathering information about network configurations, which attackers can use to plan more severe attacks.

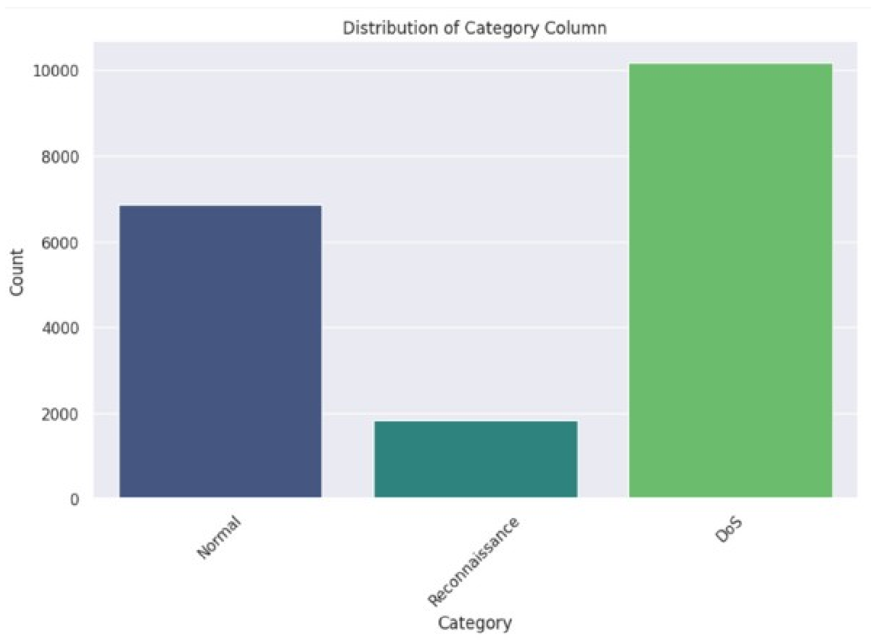


Fig: Distribution of Category

d. Challenges in Working with the Bot_IoT Dataset

The large size and complexity of the Bot_IoT dataset present several challenges:

- **Data Volume**: Handling over 70 million records requires efficient data storage and processing capabilities, which is why we opted to use the Hadoop Distributed File System (HDFS) for scalable storage.
- **Class Imbalance**: The dataset contains significantly more normal instances than attack instances, which can bias machine learning models towards predicting normal traffic. Techniques like oversampling, undersampling, and weighted loss functions were used to address this issue.
- **High-dimensional Data**: The dataset includes a wide range of features, many of which are correlated. Feature selection techniques were applied to reduce dimensionality while retaining the most informative attributes.
- **Real-time Processing**: The need to detect anomalies in real time required integrating data streaming tools like Apache Kafka with the Hadoop-Spark framework to ensure low-latency processing.

e. Importance of Feature Engineering

Based on the comprehensive analysis of the Bot_IoT dataset, we strategically selected a subset of features that were most relevant for training our anomaly detection model. This selection was driven by the need to balance model accuracy, computational efficiency, and real-time performance.

We have used a total of 11 features for model training: "flgs", "proto", "pkts", "bytes", "dur", "mean", "stddev", "sum", "min", "max", "rate".

Rationale for Feature Selection

The selected features were chosen for their strong correlation with network traffic behavior indicative of anomalies and attacks. By focusing on a combination of packet-level statistics (like pkts, bytes, mean, stddev) and connection attributes (like flgs, proto, dur), the model is better equipped to distinguish between normal traffic and various types of network attacks. This selection helps in reducing the dimensionality of the dataset while retaining critical information, thus improving the model's efficiency and accuracy.

By leveraging the rich set of features in the Bot_IoT dataset, we were able to develop a robust anomaly detection system that can efficiently identify and respond to various types of IoT network attacks in real time. The comprehensive nature of the dataset makes it ideal for evaluating the scalability and effectiveness of our proposed solution using the Hadoop-Spark framework.

MODEL TRAINING AND EVALUATION

In our evaluation of different machine learning models for anomaly detection, we experimented with **Random Forest**, **Logistic Regression**, and **XGBoost**. The goal was to identify the model that provided the highest accuracy, precision, recall, and F1-score, ensuring robust classification of network traffic into normal and attack categories.

1. Comparative Performance Analysis

The performance of each model was evaluated using the following metrics:

- **Accuracy**: Overall correctness of predictions.
- **Precision**: Ability to correctly identify positive instances (attack events).
- **Recall**: Capability to detect all relevant positive instances.
- **F1-Score**: Harmonic mean of precision and recall, providing a balanced measure.

Metric	Random Forest	Logisitc regression	XGBoost
ACCURACY	99.63%	90.20%	99.70%
PRECISION	99.63%	92.14%	99.70%
RECALL	99.63%	90.27%	99.70%
F1-SCORE	99.63%	90.37%	99.70%

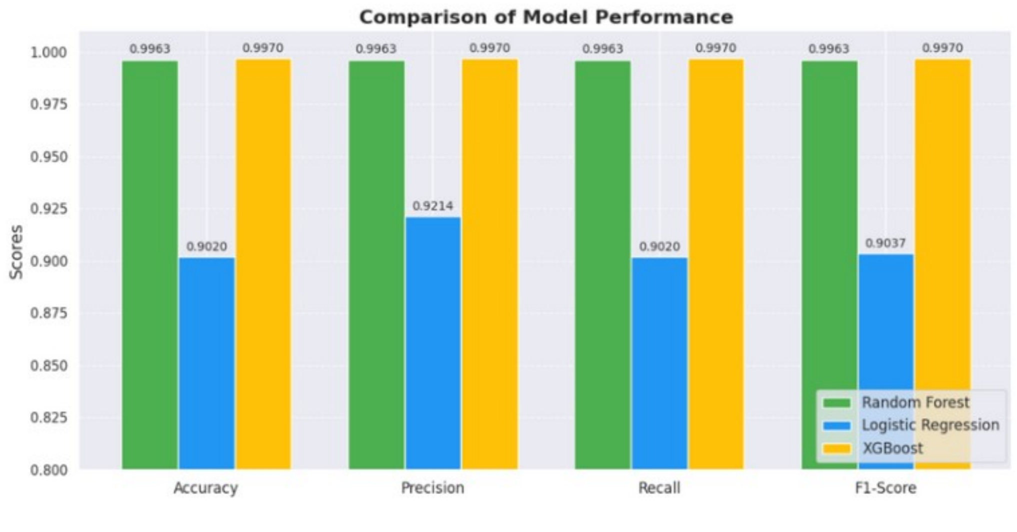


Fig: Comparison of Model Performance

As depicted in the bar chart above, **Random Forest** and **XGBoost** showed significantly higher accuracy and precision compared to Logistic Regression, making them strong candidates for real-time anomaly detection.

2. Why Random Forest Was Chosen as the Optimal Model

After analyzing the comparative performance, we chose **Random Forest** as the optimal model for the following reasons:

- **High Accuracy**: With an accuracy of **99.63%**, Random Forest demonstrated robust classification capabilities, ensuring minimal false positives and negatives.
- **Balanced Precision and Recall**: It achieved a balanced performance across precision (99.63%) and recall (99.63%), making it ideal for anomaly detection where false negatives (missed attacks) are critical.
- **Scalability and Efficiency**: The Random Forest algorithm is highly efficient when handling large datasets, making it a good fit for our **Hadoop-Spark** based system where scalability is crucial.
- **Feature Importance**: Random Forest provides insights into feature importance, helping us understand which network attributes are most indicative of attacks. This interpretability is beneficial for fine-tuning the model and understanding network vulnerabilities.
- **Reduced Overfitting**: By averaging the results of multiple decision trees, Random Forest reduces the risk of overfitting, which was a concern given the diversity in the dataset (normal vs. various attack types).

3. Model Selection Rationale

Although **XGBoost** showed slightly higher accuracy (99.70%) than Random Forest, we prioritized Random Forest due to its interpretability, lower training time, and robustness in handling large-scale data within the Hadoop-Spark framework. Additionally, Random Forest was less prone to overfitting during cross-validation, providing more consistent results across different subsets of data.

Integration with Hadoop and Kafka

To efficiently handle the real-time processing of the large volume of IoT network traffic data, we utilized **Apache Kafka** for data streaming and pipelining and **Hadoop Distributed File System (HDFS)** for scalable storage. This setup enabled us to ingest, process, and store data seamlessly, ensuring that our anomaly detection model could operate in real-time without bottlenecks.

HDFS for Scalable Storage

- **Purpose**: The Hadoop Distributed File System (HDFS) was used as the backbone for storing both the raw network traffic data and the results of the anomaly detection predictions. HDFS is designed for distributed storage, allowing us to store the large-scale Bot_IoT dataset efficiently.
- **Advantages**: HDFS is optimized for handling large files, providing high throughput access to data. It is resilient to hardware failures due to data replication, ensuring that our system remains robust even in the face of unexpected disruptions.

Kafka for Data Pipelining and Batch Processing

- **Purpose**: Apache Kafka was used to establish a real-time data pipeline, handling the continuous flow of network traffic data into our system. Kafka is ideal for managing high-throughput data streams, which is crucial given the volume and velocity of IoT traffic.
- **Batch Processing**: Instead of processing every single event in real time (which can be resource-intensive), Kafka's batch processing capability allows for efficient data handling. By grouping incoming messages into batches, the system processes data more efficiently, reducing latency and improving overall throughput.

Producer-Consumer Workflow in Kafka

The entire pipeline was designed to flow from data ingestion to anomaly detection and finally to result storage. Here's a step-by-step breakdown:

Raw Data Ingestion by Kafka Producer: The **producer** component in Kafka is responsible for receiving the raw IoT network traffic data. The producer reads the incoming data stream (from sensors, routers, or other IoT devices) and packages this data into messages.

Each message is then sent to a **Kafka topic**. A Kafka topic serves as a data stream where messages are stored sequentially. The topic acts as a buffer, allowing multiple consumers to read the data at their own pace.

Data Flow from Producer to Kafka Topic: Once the producer sends messages to the Kafka topic, they are stored temporarily until they are consumed. The topic acts as an intermediary, ensuring that data is not lost if the consumer is temporarily unavailable.

Kafka's fault-tolerance mechanisms ensure that even if there is a system failure, the data remains intact, making it highly reliable for real-time applications.

Data Processing by Kafka Consumer: The **consumer** component listens to the Kafka topic and retrieves batches of messages for processing. In our system, the consumer applies a **pre-trained machine learning model** (specifically, Random Forest) to each batch of network traffic data.

The consumer reads the data, applies the model to detect whether the traffic is normal or an anomaly, and generates a prediction. This process is repeated for each batch, allowing the system to process large volumes of data efficiently.

Saving Predictions to HDFS:After the consumer processes the data and makes predictions, the results (indicating whether a traffic event is normal or anomalous) are stored in **HDFS**. The storage in HDFS ensures that the predictions are accessible for future analysis and reporting.

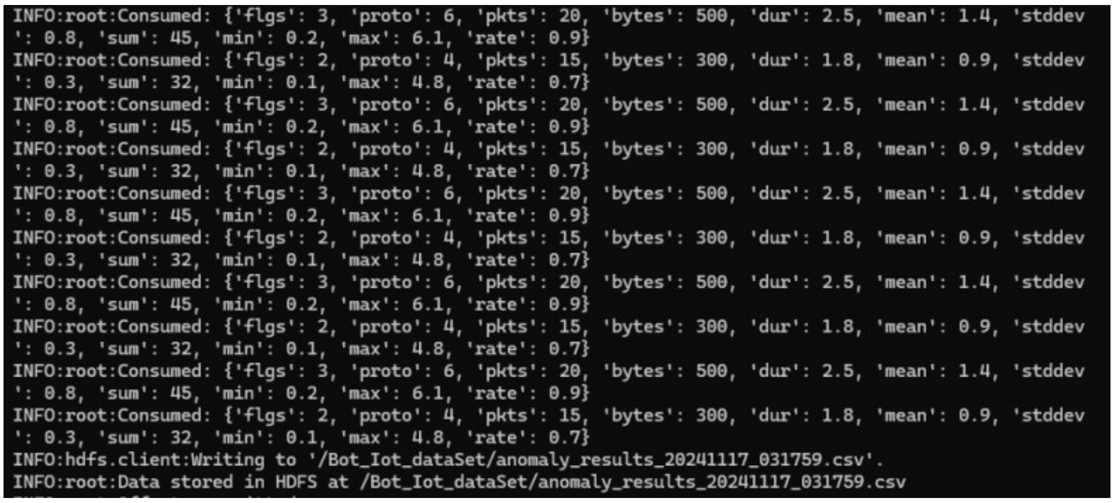
This integration allows the system to archive processed data, providing a historical record that can be used for auditing, retraining models, or further analysis.

Handling Producer-Consumer Deadlock and Batch Processing

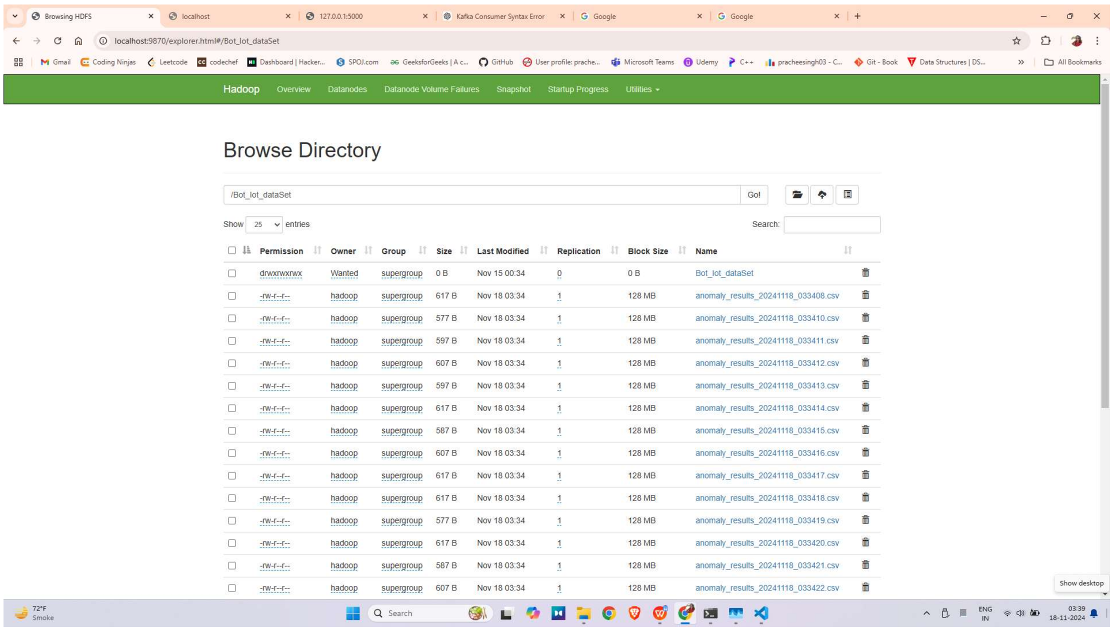
- **Producer-Consumer Deadlock**: In a typical Kafka setup, deadlock situations can occur if the producer is sending data faster than the consumer can process it. This imbalance can lead to the Kafka topic becoming overloaded, resulting in dropped messages or system crashes. To prevent this, we implemented batch processing, which allows the consumer to retrieve and process multiple messages at once instead of handling them individually.
- **Why Batch Processing is Necessary**:
    - **Efficiency**: Processing messages in batches reduces the overhead associated with individual message retrieval and processing, thus optimizing resource usage.
    - **Scalability**: By processing data in chunks, the system can handle higher volumes of data, making it more scalable.
    - **Latency Management**: Batch processing helps in reducing latency, as consumers can process a group of messages simultaneously rather than sequentially.

Summary of Data Flow

Raw IoT data → Kafka Producer → Kafka Topic → Kafka Consumer → Anomaly Detection Results → HDFS



```
INFO:root:Consumed: {'flgs': 3, 'proto': 6, 'pkts': 20, 'bytes': 500, 'dur': 2.5, 'mean': 1.4, 'stddev
': 0.8, 'sum': 45, 'min': 0.2, 'max': 6.1, 'rate': 0.9}
INFO:root:Consumed: {'flgs': 2, 'proto': 4, 'pkts': 15, 'bytes': 300, 'dur': 1.8, 'mean': 0.9, 'stddev
': 0.3, 'sum': 32, 'min': 0.1, 'max': 4.8, 'rate': 0.7}
INFO:root:Consumed: {'flgs': 3, 'proto': 6, 'pkts': 20, 'bytes': 500, 'dur': 2.5, 'mean': 1.4, 'stddev
': 0.8, 'sum': 45, 'min': 0.2, 'max': 6.1, 'rate': 0.9}
INFO:root:Consumed: {'flgs': 2, 'proto': 4, 'pkts': 15, 'bytes': 300, 'dur': 1.8, 'mean': 0.9, 'stddev
': 0.3, 'sum': 32, 'min': 0.1, 'max': 4.8, 'rate': 0.7}
INFO:root:Consumed: {'flgs': 3, 'proto': 6, 'pkts': 20, 'bytes': 500, 'dur': 2.5, 'mean': 1.4, 'stddev
': 0.8, 'sum': 45, 'min': 0.2, 'max': 6.1, 'rate': 0.9}
INFO:root:Consumed: {'flgs': 2, 'proto': 4, 'pkts': 15, 'bytes': 300, 'dur': 1.8, 'mean': 0.9, 'stddev
': 0.3, 'sum': 32, 'min': 0.1, 'max': 4.8, 'rate': 0.7}
INFO:root:Consumed: {'flgs': 3, 'proto': 6, 'pkts': 20, 'bytes': 500, 'dur': 2.5, 'mean': 1.4, 'stddev
': 0.8, 'sum': 45, 'min': 0.2, 'max': 6.1, 'rate': 0.9}
INFO:root:Consumed: {'flgs': 2, 'proto': 4, 'pkts': 15, 'bytes': 300, 'dur': 1.8, 'mean': 0.9, 'stddev
': 0.3, 'sum': 32, 'min': 0.1, 'max': 4.8, 'rate': 0.7}
INFO:root:Consumed: {'flgs': 3, 'proto': 6, 'pkts': 20, 'bytes': 500, 'dur': 2.5, 'mean': 1.4, 'stddev
': 0.8, 'sum': 45, 'min': 0.2, 'max': 6.1, 'rate': 0.9}
INFO:root:Consumed: {'flgs': 2, 'proto': 4, 'pkts': 15, 'bytes': 300, 'dur': 1.8, 'mean': 0.9, 'stddev
': 0.3, 'sum': 32, 'min': 0.1, 'max': 4.8, 'rate': 0.7}
INFO:hdfs.client:Writing to '/Bot_Iot_dataSet/anomaly_results_20241117_031759.csv'.
INFO:root:Data stored in HDFS at /Bot_Iot_dataSet/anomaly_results_20241117_031759.csv
```



Browse Directory

/Bot_lot_dataSet

Show 25 entries

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxrwxrwx	Wanted	supergroup	0 B	Nov 15 00:34	0	0 B	Bot_lot_dataSet
-rw-r--r--	hadoop	supergroup	617 B	Nov 18 03:34	1	128 MB	anomaly_results_20241118_033408.csv
-rw-r--r--	hadoop	supergroup	577 B	Nov 18 03:34	1	128 MB	anomaly_results_20241118_033410.csv
-rw-r--r--	hadoop	supergroup	597 B	Nov 18 03:34	1	128 MB	anomaly_results_20241118_033411.csv
-rw-r--r--	hadoop	supergroup	607 B	Nov 18 03:34	1	128 MB	anomaly_results_20241118_033412.csv
-rw-r--r--	hadoop	supergroup	597 B	Nov 18 03:34	1	128 MB	anomaly_results_20241118_033413.csv
-rw-r--r--	hadoop	supergroup	617 B	Nov 18 03:34	1	128 MB	anomaly_results_20241118_033414.csv
-rw-r--r--	hadoop	supergroup	587 B	Nov 18 03:34	1	128 MB	anomaly_results_20241118_033415.csv
-rw-r--r--	hadoop	supergroup	607 B	Nov 18 03:34	1	128 MB	anomaly_results_20241118_033416.csv
-rw-r--r--	hadoop	supergroup	617 B	Nov 18 03:34	1	128 MB	anomaly_results_20241118_033417.csv
-rw-r--r--	hadoop	supergroup	617 B	Nov 18 03:34	1	128 MB	anomaly_results_20241118_033418.csv
-rw-r--r--	hadoop	supergroup	577 B	Nov 18 03:34	1	128 MB	anomaly_results_20241118_033419.csv
-rw-r--r--	hadoop	supergroup	617 B	Nov 18 03:34	1	128 MB	anomaly_results_20241118_033420.csv
-rw-r--r--	hadoop	supergroup	587 B	Nov 18 03:34	1	128 MB	anomaly_results_20241118_033421.csv
-rw-r--r--	hadoop	supergroup	607 B	Nov 18 03:34	1	128 MB	anomaly_results_20241118_033422.csv

By utilizing Kafka’s robust streaming capabilities and HDFS’s scalable storage, our system efficiently manages real-time IoT network data, ensuring reliable anomaly detection and storage of results. The batch processing approach not only prevents potential deadlocks but also optimizes system performance, making it suitable for handling high-volume IoT traffic in real-time.

Dashboard Interface and Results

The **Real-time IoT Anomaly Detection Dashboard** is a key component of our project, designed to provide users with an intuitive interface for monitoring the status of network traffic in real-time. The dashboard is built using **Flask**, a lightweight Python web framework, and dynamically displays the output from the Kafka consumer after applying the anomaly detection model. It serves as a comprehensive tool for visualizing network activity, identifying anomalies, and ensuring prompt responses to potential threats. Below is a detailed explanation of the dashboard's functionality and its role in the system.

Dashboard Features

Real-time Data Display:

The dashboard continuously updates with the latest data retrieved from the Kafka consumer, ensuring that administrators can monitor network traffic as it occurs.
The dashboard includes a tabular format that lists critical network features along with their respective predictions (Attack/Normal).

Real-time Anomaly Classification:

Each row in the table represents a network event, with the **Prediction** column clearly indicating whether the traffic is suspicious (Attack) or benign (Normal). As seen in the dashboard, most entries are currently classified as "Attack," which suggests that the system is actively identifying potential threats in real time.

The classification results are color-coded (e.g., red for attacks), providing a quick visual cue for network administrators to take action.

Pagination and Navigation:

The dashboard supports pagination, allowing users to navigate through large volumes of data efficiently. With multiple pages available, users can review historical data as needed.

The navigation controls (First, Prev, Next, Last) ensure that even with high-frequency data updates, users can easily access past entries to analyze patterns and trends.

Workflow of Data Processing and Visualization

Data Ingestion: Raw network traffic data is continuously ingested by the Kafka producer and sent to a Kafka topic.

The Kafka consumer retrieves this data in batches, applying the pre-trained Random Forest model to detect anomalies.

Model Application: The consumer processes each batch of data, classifying each record as either "Attack" or "Normal" based on the selected features (flgs, proto, pkts, bytes, dur, mean, stddev, sum, min, max, rate).

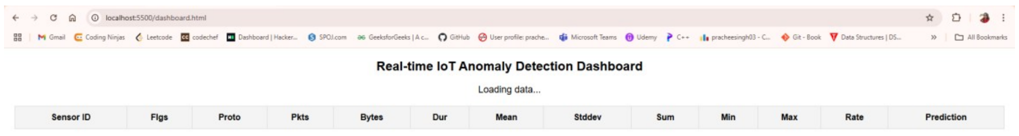
The results of these classifications are then stored in HDFS for persistence and future analysis.

Dashboard Updates: The Flask-based dashboard fetches the latest predictions from the HDFS, ensuring that the displayed data is always up-to-date.

This setup allows for real-time monitoring, enabling security teams to quickly respond to detected threats.

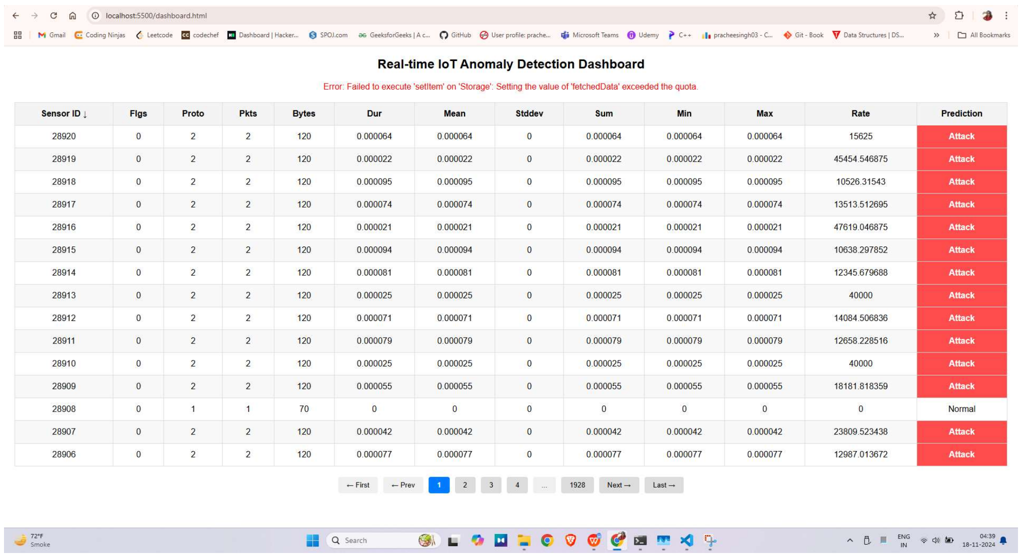
Addressing Key Challenges

- **Scalability**: The use of HDFS and Kafka ensures that the system can handle large volumes of streaming data without compromising performance.
- **Low Latency**: By leveraging Kafka's real-time data streaming capabilities and efficient batch processing, the dashboard remains responsive, providing near-instantaneous updates.
- **User-friendly Interface**: The intuitive design of the dashboard allows network administrators to quickly interpret the results, reducing the time required to respond to potential security incidents.


Real-time IoT Anomaly Detection Dashboard

Loading data...

Sensor ID	Flgs	Proto	Pkts	Bytes	Dur	Mean	Stddev	Sum	Min	Max	Rate	Prediction


Real-time IoT Anomaly Detection Dashboard

Error: Failed to execute 'setItem' on 'Storage': Setting the value of 'fetchedData' exceeded the quota.

Sensor ID ↓	Flgs	Proto	Pkts	Bytes	Dur	Mean	Stddev	Sum	Min	Max	Rate	Prediction
28920	0	2	2	120	0.000064	0.000064	0	0.000064	0.000064	0.000064	15625	Attack
28919	0	2	2	120	0.000022	0.000022	0	0.000022	0.000022	0.000022	45454.546875	Attack
28918	0	2	2	120	0.000095	0.000095	0	0.000095	0.000095	0.000095	10526.31543	Attack
28917	0	2	2	120	0.000074	0.000074	0	0.000074	0.000074	0.000074	13513.512695	Attack
28916	0	2	2	120	0.000021	0.000021	0	0.000021	0.000021	0.000021	47619.046875	Attack
28915	0	2	2	120	0.000094	0.000094	0	0.000094	0.000094	0.000094	10638.297852	Attack
28914	0	2	2	120	0.000081	0.000081	0	0.000081	0.000081	0.000081	12345.679688	Attack
28913	0	2	2	120	0.000025	0.000025	0	0.000025	0.000025	0.000025	40000	Attack
28912	0	2	2	120	0.000071	0.000071	0	0.000071	0.000071	0.000071	14084.506836	Attack
28911	0	2	2	120	0.000079	0.000079	0	0.000079	0.000079	0.000079	12658.228516	Attack
28910	0	2	2	120	0.000025	0.000025	0	0.000025	0.000025	0.000025	40000	Attack
28909	0	2	2	120	0.000055	0.000055	0	0.000055	0.000055	0.000055	18181.818359	Attack
28908	0	1	1	70	0	0	0	0	0	0	0	Normal
28907	0	2	2	120	0.000042	0.000042	0	0.000042	0.000042	0.000042	23809.523438	Attack
28906	0	2	2	120	0.000077	0.000077	0	0.000077	0.000077	0.000077	12987.013672	Attack

Challenges Encountered

Implementing a real-time anomaly detection system using the Hadoop and Kafka ecosystem for IoT security involved several challenges. Here's a summary of the key obstacles and solutions:

1. Data Handling and Storage

- **Challenge**: Managing the **Bot_IoT dataset** with over 70 million records posed difficulties in storage and processing due to its size and high dimensionality.
- **Solution**: We utilized **HDFS** for distributed storage, optimizing it with techniques like data partitioning and compression to improve data retrieval speed and storage efficiency.

2. Real-time Streaming with Kafka

- **Challenge**: Ensuring smooth integration between **Kafka** and HDFS for real-time data ingestion was complex, especially under high data velocities. Managing the flow between the Kafka producer and consumer to avoid latency was critical.
- **Solution**: By leveraging **Kafka's batch processing**, we optimized message throughput and minimized processing overhead. Additionally, adjustments to buffer sizes and error handling were implemented to prevent **producer-consumer deadlocks**, ensuring efficient data flow and reducing bottlenecks.

3. Dashboard Optimization and Storage Limits

- **Challenge**: The **Flask-based dashboard** faced issues with browser storage limits during real-time updates, resulting in quota errors due to excessive local data caching.
- **Solution**: We implemented **pagination** and optimized the data fetched by the dashboard to load only what's necessary. Shifting some data storage to server-side caching helped maintain dashboard responsiveness, even with continuous data updates.

4. Addressing Class Imbalance

- **Challenge**: The dataset had an imbalance, with more normal traffic records than attacks, causing potential biases in model predictions.
- **Solution**: To counteract this, we applied **oversampling** for minority attack classes and **undersampling** for normal traffic. Additionally, **weighted loss functions** in the Random Forest model improved detection accuracy for attack events.

Future Scope

The current system has proven to be a robust solution for real-time anomaly detection in IoT networks, but there are several avenues for future enhancements to improve its effectiveness and scalability:

Integration of Advanced Machine Learning Models: While the Random Forest model provided strong performance, future work could explore the integration of deep learning techniques like Convolutional Neural Networks (CNNs) or Long Short-Term Memory (LSTM) networks. These models can potentially capture more complex patterns in network traffic, improving the detection of sophisticated attacks.

Deployment on Cloud Platforms: To enhance the system's scalability, deploying it on cloud platforms such as AWS, Azure, or Google Cloud would enable it to handle even larger datasets and higher data velocities. Cloud-based deployment would also allow for easier scaling of resources, high availability, and better fault tolerance.

Edge Computing Integration: For IoT environments with bandwidth constraints, integrating the system with edge computing devices could enable localized processing closer to the data source, reducing latency and bandwidth usage. This would be particularly beneficial in scenarios where real-time response is critical, such as in smart cities or healthcare IoT networks.

Real-time Alerting and Notification Systems: Building an automated alerting mechanism that notifies network administrators in real-time when anomalies are detected would enhance the system's usability. Integrating with communication tools like Slack, email, or SMS can ensure immediate responses to detected threats.

Enhanced Visualization and Analytics: The current Flask dashboard provides real-time monitoring, but expanding its capabilities to include advanced analytics, trend analysis, and visual reports could help administrators better understand network behaviors and attack patterns over time. Interactive dashboards with drill-down capabilities could be added for deeper insights.

Testing on Diverse Datasets: To improve the generalizability of the system, future research should include testing with other IoT datasets beyond Bot_IoT. This would validate the model's effectiveness in detecting anomalies across different environments, ensuring its adaptability to various IoT setups.

Implementing Security Measures: As this system handles sensitive IoT network data, future iterations should include robust security measures such as encryption, secure data transmission, and user authentication to protect the data pipeline from potential breaches.

Conclusion

The exponential growth of IoT devices has introduced new challenges in maintaining network security, given the vast amount of data generated continuously. This project successfully developed a scalable, real-time anomaly detection system utilizing the **Hadoop ecosystem, Apache Kafka**, and **machine learning algorithms**. By leveraging distributed computing frameworks, the system was able to efficiently process large volumes of IoT data, achieving high accuracy in detecting network anomalies.

The integration of **HDFS for scalable storage**, Kafka for high-throughput data streaming, and a Flask-based dashboard for real-time visualization demonstrated the system’s ability to handle real-world IoT traffic scenarios. The use of a Random Forest model enabled efficient classification, but future enhancements, such as incorporating deep learning models and cloud deployment, can further elevate the system's capabilities.

The results indicate that the developed solution can significantly enhance IoT network security by providing timely insights into potential threats. By addressing scalability, real-time processing, and user-friendly visualization, the project lays a strong foundation for future research and development in the field of IoT security. With further improvements, the system has the potential to be deployed in real-world environments, providing robust protection for critical IoT infrastructure across various industries.

REFERENCES

[1] Al-Haija QA, Droos A. A comprehensive survey on deep learning-based intrusion detection systems in Internet of Things (IoT). Expert Systems.:e13726.
[2] Hadj-Attou A, Kabir Y, Ykhlef F. A Big Data Security Framework for IoT Networks using Weighted Average Ensemble. In2024 2nd International Conference on Electrical Engineering and Automatic Control (ICEEAC) 2024 May 12 (pp. 1-6). IEEE.
[3] Motylinski M, MacDermott Á, Iqbal F, Shah B. A GPU-based machine learning approach for detection of botnet attacks. Computers & Security. 2022 Dec 1;123:102918.
[4] Mohammed MM, Alheeti KM. Deep Learning Model For IDS In the Internet of Things. In2021 7th International Conference on Contemporary Information Technology and Mathematics (ICCITM) 2021 Aug 25 (pp. 153-159). IEEE.
[5] Popoola SI, Adebisi B, Hammoudeh M, Gacanin H, Gui G. Stacked recurrent neural network for botnet detection in smart homes. Computers & Electrical Engineering. 2021 Jun 1;92:107039.
[6] Abushwereb M, Alkasassbeh M, Almseidin M, Mustafa M. An accurate IoT intrusion detection framework using apache spark. arXiv preprint arXiv:2203.04347. 2022 Feb 21.