2025 17th International Conference on COMmunication Systems and NETworks (COMSNETS) | 979-8-3315-3119-5/25/$31.00 ©2025 IEEE | DOI: 10.1109/COMSNETS63942.2025.10885573

Streamlined Data Pipeline for Real-Time Threat Detection and Model Inference

Rajkanwar Singh[1], Aravindan V[1], Sanket Mishra[1], Sunil Kumar Singh[2],
[1]School of Computer Science & Engineering,
VIT-AP University,Amaravati, Andhra Pradesh, India
[2]Department of Computer Science and Engineering, Manipal Institute of Technology Bengaluru,
Manipal Academy of Higher Education, Manipal, Karnataka, India
rajkanwars15@outlook.com,aravindanvemula2@gmail.com,sanketmishra@live.com,sunil.singh@manipal.edu

***Abstract*—Real-time threat detection in streaming data is crucial yet challenging due to varying data volumes and speeds. This paper presents an architecture designed to manage large-scale, high-speed data streams using deep learning and machine learning models. The system utilizes Apache Kafka for high-throughput data transfer and a publish-subscribe model to facilitate continuous threat detection. Various machine learning techniques, including XGBoost, Random Forest, and LightGBM, are evaluated to identify the best model for classification. The ExtraTrees model achieves exceptional performance with accuracy, precision, recall, and F1 score all reaching 99% using the SensorNetGuard dataset within this architecture. The PyFlink framework, with its parallel processing capabilities, supports real-time training and adaptation of these models. The system calculates prediction metrics every 2,000 data points, ensuring efficient and accurate real-time threat detection.**

***Index Terms*—Malicious Node, Big Data Analytics, Online Machine Learning, Internet of Things**

I. Introduction

Cyberattacks have increased as the digital landscape expands, and malicious actors employ more sophisticated tactics. Identifying and mitigating these threats is a high priority. The proliferation of connected devices and the increasing dependence on online platforms create more opportunities for cybercriminals to exploit vulnerabilities. In addition, the sophistication of attacks has grown, making them harder to detect and defend against. Intrusion detection systems (IDSs) must continuously improve their ability to detect new types of attack.

The surge in Internet of Things (IoT) devices and sensors generates large amounts of data that can contain threats or anomalies that require attention. Detecting these anomalies is essential in real-life applications, such as identifying fraudulent financial transactions, fake calls, network intrusions, and healthcare care anomalies. Such detection provides insights that might otherwise be overlooked. In a streaming environment, storing and processing data can be unfeasible due to high I/O usage or the need for real-time analysis.

Real-time streaming data processing demands high throughput, low latency, fault tolerance, and highly scalable platforms. Computing platforms are required to handle batch and stream processing, while streaming architectures must efficiently transmit, consume, and process data.

This study introduces a high-throughput architecture that combines deep learning and machine learning models to detect malicious nodes in real-time IoT data streams. The design utilizes a publish-subscribe mechanism, wherein data is posted to a Kafka topic observed by the consumer. PyFlink acts as the consumer, employing the data for threat classification and detection, and functions as the processor within Kafka's system framework. Various machine learning and deep learning models are utilized to select features, competing to produce the best set of features. The outputs from the first model to the final version are published in Kafka. This method aims to provide the quickest results, regardless of the dataset fed into the system. The proposed architecture selects the optimal models, guaranteeing reliable results even with changes in dataset. Identifies a feature set that can easily be adjusted to changes in data. Using this feature set, a PyFlink job trains models like Random Forest, XGBoost, and LightGBM; predictions are then produced using the model with the best performance. With PyFlink, a distributed computing and stream processing framework, these models are trained efficiently, enabling continuous model updates and retraining as needed. This allows the architecture to adapt and retain accuracy as the data evolve.

A. Organisation

This paper is organised as follows. Section I is the introductory section while Section II throws light on the research done in this field previously by other researchers. Section III explains various components used, the architecture and their functioning. Section IV discusses the application of the architecture to the SensorNetGuard data set [2]. The last section of the paper is the summary of the work and also the future scope of the study.

II. Literature Review

Ibra Him [3] explores the integration of Artificial Intelligence (AI) and Machine Learning (ML) into real-time threat intelligence frameworks to enhance cybersecurity. The paper outlines how AI and ML can be utilized to analyze large, complex data streams in real time, identifying patterns and anomalies that may indicate potential cyber threats. It also investigates how AI-driven analytics can be integrated

with existing security infrastructures, such as Security Information and Event Management (SIEM) systems, to offer contextualized insights, reduce alert fatigue, and prioritize threats more effectively. Additionally, the paper evaluates the performance of AI algorithms through prototype systems and simulations, comparing their scalability and effectiveness against traditional methods. Surianarayanan et al. [1] propose a high-throughput streaming architecture for real-time anomaly detection, leveraging Apache Kafka for data ingestion and Random Forest for anomaly detection. Their architecture achieves near-zero latency and accuracy as high as 98.6%, even in distributed node configurations. However, the data used in this study is not publicly available. Mudgal and Bhalla [4] focus on improving intrusion detection using Random Forest and k-means clustering techniques applied to the CIC-IDS 2018 dataset. Their approach achieves an impressive 99.66% precision while reducing latency by 30% through the implementation of their Honeypot Intelligence-enabled intrusion detection system, deployed on Apache Flink. Deepthi et al. [5] introduce a Flexible Real-Time Traffic Stream Processing System (FRTSPS) designed to handle real-time traffic data streams. Built on the Lambda architecture and leveraging Apache Flink, their system reduces latency by 25% compared to traditional systems and shows superior performance in terms of Events/sec per CPU core and Data Size when compared to Apache Spark and Apache Storm. An additional study [6] presents a modular and horizontally scalable intrusion detection system architecture, designed to work in real-time or near real-time in a 1 Gbps network environment. The architecture includes modules for packet capture, queuing, storage, and analysis, and the study provides an in-depth performance evaluation using technologies like tcpdump, Apache Kafka, HDFS, and Apache Spark. Kajiura and Nakamura [7] evaluate the performance of five popular machine learning classifiers (Decision Tree, Random Forest, Naive Bayes, SVM, and kNN) for network intrusion detection. Their findings reveal notable differences in throughput and latency across classifiers, with Decision Tree and Naive Bayes delivering the highest throughput. The study also identifies performance bottlenecks in components like Zeek, Logstash, and Elasticsearch, suggesting that choosing the right machine learning algorithm can reduce system load while maintaining classifier performance. Atbib et al. [8] propose a distributed intrusion detection system tailored for IoT environments, implemented within a Fog computing architecture. The system uses Apache Spark and machine learning algorithms to analyze data from the NF-ToN-IoT-v2 dataset in real time. Their results show that Decision Tree and Random Forest outperform other models, achieving accuracies of 86% and 87%, respectively. Jemili et al. [9] tackle evolving cyber threats by introducing a hybrid intrusion detection model that integrates Random Forest, XGBoost, and decision trees through ensemble learning. Their model, tested on datasets like N-BaIoT, NSL-KDD, and CICIDS2017, achieves over 97% accuracy, showcasing the effectiveness of hybrid approaches in Big Data environments. Ashraf et al. [10] focus on developing an IoT-based cybersecurity framework for intrusion detection in the Internet of Drones (IoD). Their approach combines machine learning (ML) and deep learning (DL) techniques to address security challenges in drone networks. Tested on CICIDS2017 and KDDCup 99 datasets, their system demonstrates high precision (89-90%), accuracy (91-91%), recall (81-90%), and F-measure scores (88-90%), providing robust detection capabilities for the Internet of Drones.

III. Methodology

This section proposes the BARS architecture and highlights its components and their working.

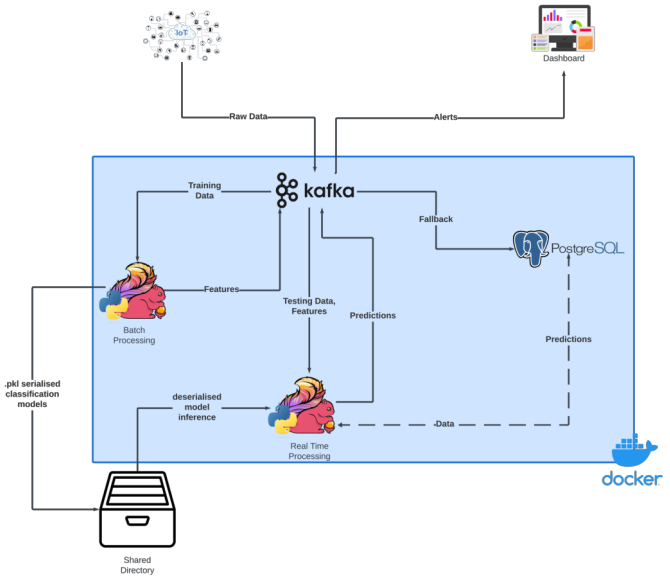

Fig. 1. BARS Architecture

```
Waiting...
Waiting...
Waiting...
Consumed event from topic race: key = None value = {"features": ["Route_Reply_Fr
equency", "Data_Transmission_Frequency", "Data_Reception_Frequency", "CPU_Usage"
]}
```

Fig. 2. Depiction of Kafka waiting for and consuming messages

A. Tools and Technologies Used

1) Flask API: Flask API hosted on port 5001 (mapped to 5000 on system) to receive sensor data from HTTP POST requests and send those data to a Kafka topic. It also maintains a delivery report function that logs whether Kafka message delivery was successful or not. Polling and Flushing are used to ensure that messages are sent by polling the producer and flushing the queue. The message key is a JSON string that contains the id of the data. The message value is the JSON string representation of the entire data.

2) Apache Kafka: Apache Kafka is an open source distributed streaming platform that offers highly scalable and fault-tolerant data processing capabilities for real-time data processing, alerting, and reporting. [11] Kafka is used in the work as a message broker and serves as the central nervous system of the framework. It provides a pub-sub model allowing various sources of inflow and outflow to various engines/frameworks, while also having higher throughput [12]. While other tools like RabbitMQ are ideal for reliable message queuing with complex routing, Kafka excels in high-throughput, real-time data streaming. Fig. 2 depicts of Kafka waiting for messages and consuming messages- results from feature selection jobs in this case.

3) Apache Flink: Apache Flink unifies diverse data processing applications in a single execution model, allowing real-time analytics, continuous data pipelines, historic data processing, and iterative algorithms [13]. Tools such as Spark process data in micro-batches, while Flink supports true real-time streaming making Flink better for low-latency applications.

4) PostegreSQL: The PostegreSQL database is used to maintain a record of data received and predictions on data to be used as a fallback in case of Kafka failure.

5) Docker: Docker, a Platform as a Service (PaaS) uses OS-level virtualization to deliver software in containers. Each container is built using an image and each of the containers is isolated [14]. Docker compose has been used to launch all containers in the same stack and hosted on the same network by means of a YAML file. Docker is used to orchestrate containerized applications that provide container flexibility, consistency, and user-friendliness. Fig. 3 depicts a screenshot of the Docker stack (with multiple containers) used in this work.

6) Feature Selection: Two deep learning methods for feature selection were long-short-term memory (LSTM) networks and Deep Neural Networks (DNN). 33,441 trainable parameters were split across two LSTM layers and one dense layer in the LSTM model. The DNN model included six layers and 12,673 trainable parameters. These models were optimized using Particle Swarm Optimization (PSO), which is renowned for its quick convergence. Furthermore, LightGBM and other machine learning models were used for feature extraction, and both PSO and Genetic Algorithms (GA) were used for optimization.Both PSO and GA are wrapper functions. A total of four feature sets were generated. To optimize the architecture, priority was given to the feature set that was produced first. This was achieved by creating a race condition through the use of multithreading techniques.

7) Classifiers: Batch mode on PyFlink was used to train five classifiers: Random Forest, Gradient Boosting, XGBoost, LightGBM, and Extra Trees. After that, the best model from this group of classifiers is chosen to predict the streaming data.

B. Integration and Flow

Apache Flink and Apache Kafka together create a complete set of tools for real-time data processing and analysis, improving decision-making and flexibility [15]. Fig. 1 depicts the flow of data in the pipeline. Raw data is collected via a REST API and produced to Kafka topic/s as per the API endpoint, i.e., different topic for different API endpoint. All of these topics are subscribed to by Flink sinks in various jobs. The first job performs feature extraction by way of Deep Learning frameworks namely- LSTM, DNN and machine learning algorithm (LightGBM) and are optimised with algorithms namely- Particle Swarm Optimization (PSO) and Genetic Algorithm.These algorithms run parallelly using multi threading. The extracted features are posted on various Kafka topics. A race condition is created to wait for the fastest algorithm to post its features, which are then posted to a final topic. LSTM optimised with PSO has consistently been the first to publish the results on the topic[1].We suspect this is because lstem captures better over longer term and pso algorithm being generally fastern then Ga algorithms. This topic is subscribed to by the next job for model training. The model training job ingests features and data from relevant Kafka topics and trains five machine learning algorithms on these data. Before training, the dataframe is passed through a customised resampler as highlighted in Algorithm 1. A comparison of the original and resampled class distributions can be seen in Fig 4. All trained models are serialized as pickle files (.pkl) and stored in a shared volume between the Flink Task-Manager and the Job-Manager. This job also saves the metrics of these models as a CSV file. The next job, which runs in a streaming environment, is that of prediction. An algorithmic approach is adopted in this prediction job to select the best performing model by means of the highest accuracy, precision, recall and F1 score and the lowest training time, as outlined in Algorithm 2. The best performing model is then deserialized and used to make predictions on the newly received data. All predictions are produced on a new Kafka topic. Another PyFlink job computes model performance metrics in fixed window sizes and produces them for a new Kafka topic. The predictions are also stored and persist in a PostgreSQL

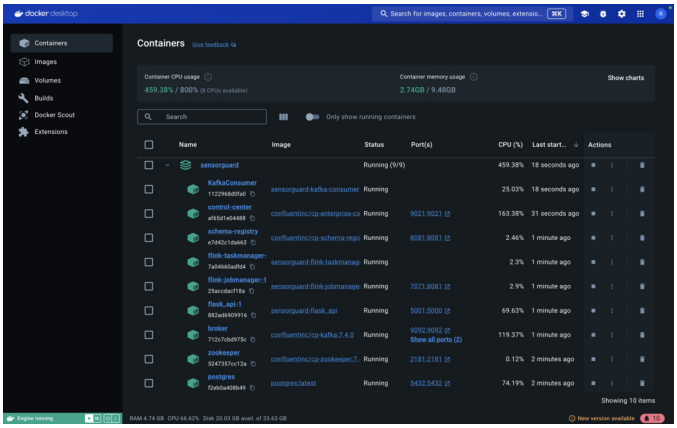

Fig. 3. A screenshot of pipeline components in Docker

[1]The results of other methods have been explored but have not been discussed due to page limitations

Algorithm 1 Resample Data with Target Count

Input: df: DataFrame with data to be resampled
Input: $class\_column$: Name of the column with class labels
$class\_counts \leftarrow df[class\_column]$.value_counts()
$majority\_class \leftarrow$ max$(class_counts)$
$minority\_class \leftarrow$ min$(class_counts)$
$target\_count \leftarrow \frac{majority\_class + minority\_class}{2}$ ▷ Target count for balancing
$resampled\_dfs \leftarrow []$ ▷ List to store resampled DataFrames
for each $(class\_value, count)$ in $class\_counts$ **do**
    $class\_df \leftarrow df[df[class\_column] == class\_value]$
    if $count < target\_count$ **then**
        $resampled\_df \leftarrow$ resample$(class_df,$ replace=True, n_samples=target_count, random_state=42)
    else if $count > target\_count$ **then**
        $resampled\_df \leftarrow$ resample$(class_df,$ replace=False, n_samples=target_count, random_state=42)
    else
        $resampled\_df \leftarrow class\_df$
    end if
    $resampled\_dfs$.append$(resampled_df)$
end for
$resampled\_df \leftarrow$ pd.concat$(resampled_dfs)$
Output: $resampled\_df$

database. This database is used to be the back-end, and the front-end is provided by a dashboard. All these components collectively form what is known as the BARS (Big data Analytics for Real-time Security) architecture.

The code repository can be accessed at https://github.com/Rajkanwars15/BARS

IV. Results and Discussion

The pipeline is tested in the SensorNetGuard dataset [2]. Based on Algorithm 2, the Extra Trees model is selected as the best performing model due to its superior metrics and efficiency. The model was then serialized as best_model.pkl for the prediction job.

The data set SensorNetGuard is a dataset for identifying malicious sensor nodes comprising 10,000 samples with 21 features. It is designed to facilitate the identification of malicious sensor nodes in a network environment, specifically focusing on IoT-based sensor networks. The dataset includes a diverse range of features that allow for the application of machine learning models to identify various types of attacks, such as black hole, gray hole, flooding attacks, and Sybil attacks.

The features that exhibit the highest percentage of outliers are $Is\_Malicious$ (4.87%) and $Energy\_Consumption\_Rate$ (4.47%), suggesting possible problems with power and security. Significant variability is also shown by the packet drop rate (3.68%) and error rate (4.42%), which may have an effect on the network dependability. The outlier proportions for the majority of other indicators range from 1% to 2%, indicating periodic aberrations but overall consistency in those areas. The two metrics with the fewest outliers are Bandwidth (0.67%) and Number of Neighbors (0.13%), indicating more consistent behavior in these areas.The observed outliers comprise malicious nodes.

The performance of the five machine learning models is evaluated and ranked based on accuracy, precision, recall, F1 score, and training time, as shown in Table I. The Extra Trees model achieved the highest rankings across all metrics, including accuracy (0.9994), precision (0.9994), recall (0.9994), and F1 score (0.9994), with the shortest training time of 2.25 seconds. It is worth noting that, even though LightGBM and XGBoost rank higher due to better Accuracy, Precision, Recall, F1 Score; both these models take significantly longer to train (58-59 seconds) as compared to Gradient Boosting and Random Forest.

TABLE I
Ranked Models with Metrics

Rank	Model	Accuracy	Precision	Recall	F1 Score	Training Time
1	ExtraTrees	0.9994	0.9994	0.9994	0.9994	2.2492
2	LightGBM	0.9984	0.9984	0.9984	0.9984	58.3919
3	XGBoost	0.9982	0.9982	0.9982	0.9982	24.2645
4	GradientBoosting	0.9974	0.9974	0.9974	0.9974	4.7614
5	RandomForest	0.9970	0.9970	0.9970	0.9970	4.1090

ROC Curve of the models can be seen in Fig. 5. All models perform exceptionally well, with AUC values close to or at 1.00. XGBoost and LightGBM are the best-performing models

Algorithm 2 Select and Serialize the Best Model

Input: List of models with their metrics and attributes
Output: Serialized best model as `best_model.pkl`
Parameters:
criteria ← List of criteria for sorting (e.g., accuracy, precision, recall, f1_score)
sort_order ← List indicating sorting order for each criterion (e.g., descending, ascending)
models_metrics ← list of tuples (model, metrics, attributes)
for criterion, order **in** zip(criteria, sort_order) **do**
    models_metrics.sort_by(criterion, order) ▷ Sort by the current criterion
end for
best_model ← models_metrics[0].model ▷ Select the top model
Serialize best_model to file best_model.pkl

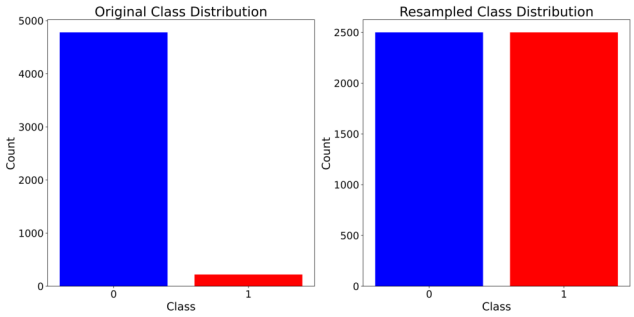

Fig. 4. Class Distributions

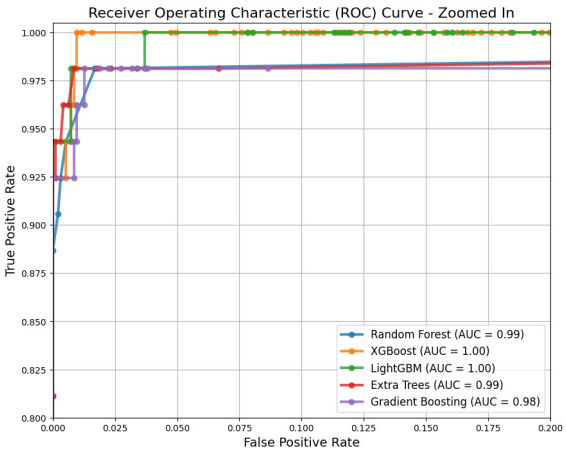

Fig. 5. ROC Curve of Models

while Random Forest, Extra Trees, and Gradient Boosting show a slight deviation from a perfect score.

The top 2 models are compared in figure 6.

However, the main difference is depicted in figure 7 in terms of training time.

ExtraTrees consistently outperforms LightGBM across all metrics, though the differences are minimal. The close values indicate that both models are highly effective for the task, but ExtraTrees has a slight edge in terms of predictive performance.

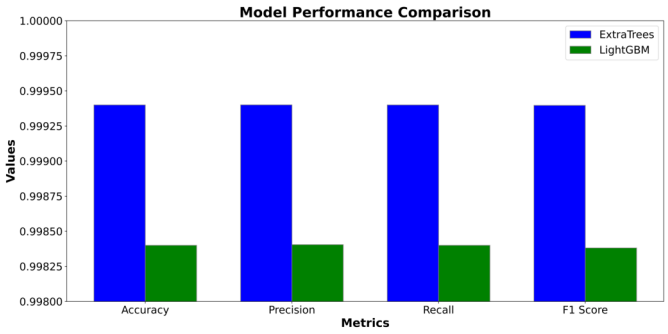

Fig. 6. Comparison of 2 best performing models

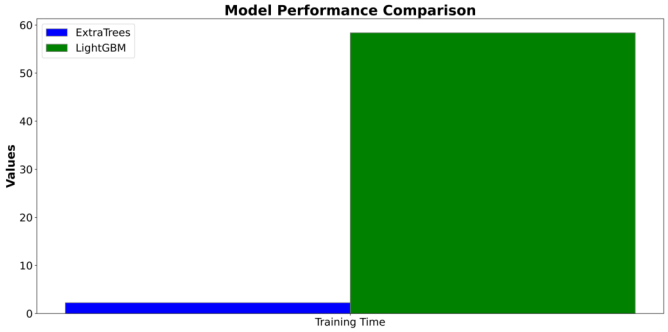

Fig. 7. Training Time Comparison of 2 best performing models

To ensure robustness, 5-fold cross-validation was performed for each model. The Extra Trees model consistently demonstrated high accuracy, precision, recall, and F1 scores across all folds, with only minor variations. In particular, the Extra Trees model achieved a perfect score of 1.0 in several places, reinforcing its superior performance.

LightGBM exhibited variability in its cross-validation results, with a peak accuracy of 1.0 in some cases but lower values in others, reflecting some instability. Similarly, XGBoost and Gradient Boosting showed strong results with occasional slight dips, while Random Forest displayed the most variability in accuracy and other metrics across folds.

After selecting and applying the optimal model, its perfor-

mance was evaluated across different window sizes to find the best evaluation threshold [16]. Testing window sizes of 500, 1000, and 2000 data points revealed that the accuracy for 1000 data points (0.968) was lower than for both 500 and 2000 data points. This suggests that training with 2000 data points offers better generalization. Additionally, Table II shows that predictions for 2000 points are made within 5 seconds.

TABLE II
PERFORMANCE EVALUATION ON VARYING WINDOW SIZES

Number of points	Accuracy	Precision	Recall	F1-Score	Inference Time
500	0.988	0.988	0.988	0.988	1.25s
1000	0.968	0.974	0.968	0.9701	2.47s
2000	0.993	0.993	0.993	0.993	5.05s

V. CONCLUSION AND FUTURE SCOPE

In this paper, we have developed and implemented the BARS architecture. The BARS architecture has been designed for stream-based IoT applications, incorporating multiple deep learning-driven feature selection techniques, various machine learning classifiers, race condition-based feature selection methods, and algorithmic classifier selection to enable real-time predictions with almost zero latency. This architecture performs feature selection and model training in batch mode from data streams and subsequently makes real-time predictions in stream mode. It also assesses its predictions in a windowed manner to ensure robustness. While Kafka serves as the core of the pipeline, PostgreSQL acts as a backup in case of failure. The architecture has been deployed on the SensorNetGuard dataset to highlight its practical viability. The entire application is containerized using Docker, making it simple to relocate and deploy this pipeline on another host machine or cloud environment. An advantage of using the BARS architecture is the lower computation cost and the ability to predict in real time with minimal deployment time, along with the flexibility to customize the algorithms for feature selection, training, and prediction. Future work will focus on scalability by introducing single/multi-broker setups and multiple worker nodes for Flink and Kafka. Due to the use of Docker, transitioning to Docker Swarm or Kubernetes to add scalability in the future is quite straightforward.Singh et al. [17] suggest integrating mobile crowdsourcing with intelligent agents and IoT devices for scalable, adaptive, and efficient intrusion detection, another promising area for future IDS research.

REFERENCES

[1] C. Surianarayanan, S. Kunasekaran, and P. R. Chelliah, “A high-throughput architecture for anomaly detection in streaming data using machine learning algorithms,” *International Journal of Information Technology*, vol. 16, no. 1, pp. 493-506, Nov. 2023, doi: 10.1007/s41870-023-01585-0.

[2] Dr Karthick Raghunath K M, Dr Arvind K S, September 5, 2023, ”SensorNetGuard: A Dataset for Identifying Malicious Sensor Nodes”, IEEE Dataport, doi: https://dx.doi.org/10.21227/ba0m-cy61.

[3] I. Him. ”Leveraging Artificial Intelligence and Machine Learning for Real-Time Threat Intelligence: Enhancing Incident Response Capabilities,”, 2024.

[4] A. Mudgal and S. Bhatia, ”Big Data Enabled Intrusion Detection with Honeypot Intelligence System on Apache Flink (BDE-IDHIS),” 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), Delhi, India, 2023, pp. 1-6, doi: 10.1109/ICCCNT56998.2023.10307506.

[5] Deepthi, B.G., Rani, K.S., Krishna, P.V., & Saritha, V. (2023). An efficient architecture for processing real-time traffic data streams using apache flink. Multim. Tools Appl., 83, 37369-37385.

[6] Oliveira, R., Pedrosa, T., Rufino, J., & Lopes, R.P. (2024). Parameterization and Performance Analysis of a Scalable, near Real-Time Packet Capturing Platform. Syst., 12, 126.

[7] Maho Kajiura, Junya Nakamura, ”Practical Performance of a Distributed Processing Framework for Machine-Learning-based NIDS,” 2024.

[8] S. Atbib, C. Saadi, H. Chaoui, ”Design of A Distributed Intrusion Detection System for Streaming Data in IoT Environments,” in 2023 9th International Conference on Optimization and Applications (ICOA), 2023, pp. 1-6.

[9] F. Jemili, R. Meddeb, O. Korbaa. ”Intrusion detection based on ensemble learning for big data classification,” in Cluster Computing, vol. 27, no. 3, pp. 3771–3798, 2023.

[10] Ashraf, S. N., Manickam, S., Zia, S. S., Abro, A. A., Obaidat, M., Uddin, M., Abdelhaq, M., & Alsaqour, R. (2023). IoT empowered smart cybersecurity framework for intrusion detection in internet of drones. Scientific reports, 13(1), 18422. https://doi.org/10.1038/s41598-023-45065-8

[11] Kiran Peddireddy. ”Streamlining Enterprise Data Processing, Reporting and Realtime Alerting using Apache Kafka,” in 2023 11th International Symposium on Digital Forensics and Security (ISDFS), pp. 1-4, 2023.

[12] Kreps, J. (2011). Kafka : a Distributed Messaging System for Log Processing.

[13] Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., & Tzoumas, K. (2015). Apache flink : Stream and batch processing in a single engine. IEEE Data(base) Engineering Bulletin, 36, 28-33.

[14] Akash, A.R., Yendluri, D., Kim, J., Kim, T., Kim, E., Park, J., Kim, K., Park, J., & Min, K. (2023). Online Battery Data Analytics Pipeline using Bigdata Tools for Electric Vehicles. 2023 11th International Conference on Power Electronics and ECCE Asia (ICPE 2023 - ECCE Asia), 1207-1211.

[15] Bozkurt, A., Ekici, F., & Yetiskul, H. (2023). Utilizing Flink and Kafka Technologies for Real-Time Data Processing: A Case Study. The Eurasia Proceedings of Science Technology Engineering and Mathematics, 24, 177–183.

[16] M. Munir, S. A. Siddiqui, A. Dengel, and S. Ahmed, ”DeepAnT: A deep learning approach for unsupervised anomaly detection in time series,” *IEEE Access*, vol. 7, pp. 1991–2005, 2019, doi: 10.1109/ACCESS.2018.2886457.

[17] V. K. Singh, A. S. Jasti, S. K. Singh, and S. Mishra, “QUAD: A Quality Aware Multi-Unit Double Auction Framework for IoT-Based Mobile Crowdsensing in Strategic Setting,” 2022, arXiv. doi: 10.48550/ARXIV.2203.06647.