ISSN: 2093-5374 / E-ISSN: 2093-5382

A Unified Framework for Real-Time Intrusion Detection and Mitigation using Apache Spark

Ammar Ahmed Abdullah[1*], and Dr. Dhuha Basheer Abdullah[2]

[1*]Assistant Lecturer, Department of Computer Science, College of Computer Science and Mathematics, University of Mosul, Mosul, Iraq. ammar.20ba1175@student.uomosul.edu.iq, https://orcid.org/0009-0004-2110-2767

[2]Professor, Department of Computer Science, College of Computer Science and Mathematics, University of Mosul, Mosul, Iraq. prof.dhuha_basheer@uomosul.edu.iq, https://orcid.org/0000-0001-9003-2943

Received: September 13, 2025; Revised: October 27, 2025; Accepted: November 25, 2025; Published: December 15, 2025

Abstract

The traditional intrusion detection systems (IDS) are facing challenges with data volume and complexity. Detection of cyber events in real time and surveillance of significant data systems are also major challenges, which raise the need for a new IDS with cybersecurity systems in software-defined networks (SDN) that detect, classify, and mitigate low-latency threats in data streams. This will be the first fully integrated, new SDN architecture with real-time machine learning for automated and instantaneous threat response built on the integrated Spark framework. This integrated eclipse framework will operate on a synthetic data set of 10 million records to simulate a structured traffic stream where Decision Trees, Logistic Regression, Random Forest, and Multilayer Perceptron will be used to train on four classifiers that will recognize and respond to the threats in the traffic stream. Compared to single-node Spark implementations, these models proved the framework's validation, as the Spark Logistic Regression model achieved an accuracy of 99.96% and 640,000+ records processed per second, showing a double speedup vs classical implementation, while the Spark Random Forest model achieved a 40 times speedup with the same AUC. All the models were memory-constrained to 11 GB, showing suitability for today's commodity hardware. Steady-state tests validated the system's defensive capabilities as 99.9% of the attacks were blocked while the SDN controller CPU was utilized 42% less. The framework's unified architecture, performance on commodity hardware, and the real-time telemetry, fairness, and explanation features showcased telemetry data performance and gap-filling in the state of the art.

Keywords: Cyber Threat Intelligence, Big Data, Apache Spark, Machine Learning, Intrusion Detection System, DDoS, Real-Time Analytics.

1 Introduction

This domain has its roots in the 1980s, inspired by Denning's statistical intrusion detection model, which combined real-time auditing along with behavioral user profiling (Gumaste et al., 2020). In the 1990s, implementations were host-based with limited computational requirements. Nevertheless, 1998 was the year Snort 1.0 was released, which opened the community to sharing detection signatures and shifted

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), volume: 16, number: 4 (December), pp. 642-662. DOI: *10.58346/JOWUA.2025.I4.037*

*Corresponding author: Department of Computer Science, College of Computer Science and Mathematics, University of Mosul, Mosul, Iraq.

the focus of detection frameworks to the network. Recent publications confirm Snort, Suricata, and Zeek, with their varying coverage and performance trade-offs, still aid the extensive open-source community (Arvinth, 2025). However, signature-based detection faced a challenge with the propagation of worms, such as Code Red and SQL Slammer, which, in reverse, drove the target of the discipline to Cyber-Threat Intelligence (CTI) described as "evidence-based knowledge concerning an adversary, enriched beyond Indicators of Compromise (IoCs) by motive, means, and opportunity" (Gupta & Sharma, 2019).

With regards to open-source intrusion detection systems, Snort, Suricata, and Zeek can be looked at as the primary ones in the field. Within the comparative analysis, Suricata, having multithreading, tends to be more efficient than Snort and Zeek based on the performance on the high throughput requirements. These tools present the capability to identify and counteract attacks based on given signatures, which are defendable against known threats and adversaries (Cao & Jiang, 2024). To address the challenges and limitations of the signature-based systems, Cyber Threat Intelligence (CTI) systems in Cyber Security have been designed and constructed (Ekundayo et al., 2024). CTI systems are the collection, processing, and analysis of information that is of importance to an organization to assist in the prediction and prevention of attacks (Stănciulescu et al., 2024).

The system IDS has also improved due to Big Data's increased technological capabilities. Attack detection in real-time using scalable technology is a critical part of Apache Spark with a stream model and real-time processing of volatile big data. The emerging architectural paradigms Delta, Kappa, and Lambda synchronize both batch and stream processing, although with different complexities and performance trade-offs (Leema at al., 2024).

However, there still remain some issues, such as Spark's Structured Stream processing state issues, real-time analytics, state management, and fault tolerance, and the question of how to integrate CTI into existing IDSs without compromising their functionality or compatibility, as stated by some researchers (Waleed et al., 2022).

This paper contends that the combination and real-time integration of CTI and SDN mitigation with distributed machine learning on Apache Spark into a real-time IDS and mitigation system will solve the main problems existing today in modern IDSs by providing scalability, low latency, and adaptive cyber defense capabilities for modern networks (Alaeifar et al., 2024; Asl & Naderi, 2016).

A thoughtful, comprehensive assessment of the existing Spark IDS research determines that focus is being placed on optimizing singular components in the system, resulting in a lack of complete, fully engineered systems that are ready for field deployment or use, particularly in the absence of unified real-time processing data architectures.

Many existing solutions depend on offline batch processing, where models are trained on historical data and applied in discrete jobs rather than being integrated into a continuous loop of data ingestion, live classification, and periodic retraining. This architectural deficiency leads to unacceptably high detection latency, rendering such systems ineffective against fast-moving threats.

Second, the scope of detection in many studies is narrow. Research often concentrates on binary classification (i.e., distinguishing between "normal" and "attack" traffic) or focuses on a single, specific attack vector like a SYN Denial-of-Service (DoS) attack. The capability for multiclass, multi-vector detection—identifying and differentiating between numerous concurrent attack types at scale—remains an underdeveloped area in the literature. This problem is further exacerbated by using obsolete or simulated datasets, which affects how adaptable models created from these datasets are to real-world complex threat landscapes. Thirdly, there is a considerable lag between threat detection and response. Most academic prototypes are inactive, serving merely as alerts to a threat with no mechanism to mitigate the danger. In a real-world operational context, an effective IDS must "close the loop" by automatically triggering defensive actions, such as pushing new rules to a firewall or instructing a Software-Defined Networking (SDN) controller to block malicious flows. The absence of this integrated mitigation logic severely curtails the practical utility of these systems.

Current studies stress in increasing proportion the importance of machine learning based intrusion detection in constantly changing networks. An in-depth analysis of machine learning based intrusion detection systems for software-defined networks (SDN) captured the need for scale and adaptive action in distributed systems (Mustafa et al., 2024). Meanwhile, other studies have looked into new fields such as the Internet of Things (IoT) and proposed a mixed spiking neural network and decision tree model to enhance detection rate in devices with limited resources (Al-Kateeb & Abdullah, 2024; Zarzoor et al., 2023). There is also evidence of the growth of integration of cloud computing with deep learning, as time-sensitive deep learning methods in intrusion detection for cloud networks proved to have better temporal awareness and accuracy (Terawi et al., 2025).

Additionally, Explainable AI has been emphasized for making deep learning-based IDS solutions more interpretable and transparent, thereby enhancing trust and feasibility of real-world deployment (Radhika et al., 2025). Recent research has focused on the integration of deep learning and explainable artificial intelligence (XAI) for improving the trust and interpretability of intrusion detection systems (Sarhan et al., 2023). Federated learning has also emerged as a promising approach to cyber threat intelligence sharing and enhancing model cooperation over a distributed network while preserving privacy (Onyilo & Uzuegbu, 2025). Several works have suggested XAI models for Transparency in IDR systems, illustrating how trustworthy interpretations assist analysts and support decisions in cybersecurity (Mustafa et al., 2024; Zarzoor et al., 2023). Additionally, advanced explainable deep learning IDS have proven highly effective in real-time industrial and cloud systems, displaying transparency and predictive accuracy (Terawi et al., 2025).

Lastly, essential aspects concerning deployment in practice are often neglected. For example, explainability as a core element for trust and validation in alerting for models to avoid functioning as "black boxes" is usually ignored. Similarly, concerns regarding fairness to mitigate bias in models, and adequate resource telemetry to monitor the IDS' health and performance are seldom considered (Radhika et al., 2025). This negligence regarding operational aspects significantly delays the transition of research prototypes to production systems.

This study demonstrated that the central challenge is not just having the best algorithm but having the best architecture. An integration of low-latency processing, broad attack coverage, automated response, and operational transparency is what makes an IDS effective. With that in mind, this study provides the following:

1. A Unique Spark-Oriented Real-Time IDS Framework: Incorporating a novel architecture that streams and batches analytical processes so that instantaneous classification and continuous updates of the model prototyping can be done with no latency.
2. Greater Detection Capability of Multidimensional Attack Vectors: The validation and deployment of a handful of ML classifiers (LR, DT, RF, MLP) in conjunction with feature selection and ML balancing strategies to ensure accuracy not just in a few attacks but a diverse set of attacks in volume and in large modern datasets.
3. Integrated Mitigation Mechanism: This is the fusion of the IDS and the automated mitigation logic, illustrating the IDS's ability to communicate with elements of the network control plane (SDN Controllers), turning the IDS detection capabilities into a proactive and real-time defensive mechanism.

4. Transparency and Resource-Awareness Features: This is the addition of an explanation and monitoring layer that elaborates on the detection actions and provides real-time insights into the system's resource consumption, thus solving the "black box" problem, ensuring system reliability.

This work intends to improve the state of the art from addressing these system gaps at a high level to working on the fine details of an integrated, usable, and impactful cyber defense system.

The rest of the paper is structured in the following manner: Section 2 provides the related literature on intrusion detection systems and Spark-based security frameworks. Section 3 is a proposal of real-time Spark-based IDS architecture, system design, machine learning models and the experimental setup. Section 4 will cover the performance analysis, benchmarking data, and scalability and resource utilization analysis. In Section 5, the findings are discussed in detail and implications presented on their application into practice. Lastly, Section 6 closes the research and provides the future research directions.

2 Related Works

Apache Spark has rapidly become a staple for large-scale network security analytics in the academic world (Su et al., 2024). Pioneering work showed how Spark could reduce the detection latency by several orders of magnitude, from several seconds to less than a second, inside private clouds while maintaining a throughput of tens of thousands of packets per second in public clouds (Aladiyan, 2025). After the successful proofs of concept, the attention turned to the sophistication of algorithms. The research community has broadened and advanced from the classical machine learning basic building blocks of Random Forest and Logistic Regression to increasingly intricate deep learning architectures such as Isolation Forest, Gradient-Boosted Trees, ensembles of CNN - LSTM, and even Graph Neural Networks, while further scaling the datasets to several million network flows (Jagadeeswaran et al., 2022; Sarhan et al., 2023).

While advancements have been made, and one can argue that accuracy levels are now over 97% on a consistent basis, a study on the scholarly works of this field brings to light the growing pattern of vertical innovation overshadowing horizontal integration. In simpler terms, single algorithms have gained more power; however, the creation of complete, end-to-end frameworks remains lacking. As a result, three gaps stand in the field to this day: (i) Volume ceilings, whereby the majority of tests remain capped to a mere 6 million flows on a single node; (ii) sparse resource telemetry, where the vital performance metrics of the Spark cluster itself remain undisclosed, such as behavior on the JVM heap, and shuffle performance; and (iii) Limited automation, as very few papers pair their detection engine with real-time, automated mitigation logic.

The following fragmentation has been outlined in several studies. First, in Gumaste et al. (2020), a Spark Streaming pipeline was constructed for DDoS detection with a low alert latency of ~430 ms and with mitigation involving iptables. However, evaluations were constrained to single attack vectors (SYN floods) within small datasets (less than 1M) with no monitoring of the JVM performance. Conversely, (Haggag et al., 2020) investigated the moderate accuracy (approximately 82.2%) of deep learning models in Spark within the NSL-KDD dataset, but offline, with no streaming or mitigation components, in purely static evaluations. Other studies, for instance, (Morfino & Rampone, 2020), exhibited remarkable detection within near real-time, but with a synthetic dataset and no automated mechanisms for blocking.

Hagar and Gawali and Alslman et al. also recorded near 100% accuracy on contemporary datasets like CSE-CIC-IDS2018, CIC-DDoS2019; however, their assessments were offline and thus had no real-time latency and no in-phase integrated response. Azeroual concentrated on performance modeling, specifically on the speedup, scaled efficiency of Spark MLlib, and the time in CPU and time processing, in the context of IDS-like workloads; however, this study did not include any of the security metrics nor any particular IDS. More research work revolves around detection accuracy and has been done in offline settings.

In their publication, (Al-Kateeb & Abdullah, 2023; Azeroual & Nikiforova, 2022; Al-Kateeb & Abdullah, 2024) introduced AdaBoostCoTCKD, a hybrid fog-cloud framework integrating AdaBoost machine learning for the prediction of chronic kidney disease (CKD) (Onyilo & Uzuegbu, 2025). This framework attempts to overcome the drawbacks of traditional cloud computing, namely latency, scalability, and energy wastefulness, by combining it with fog computing for real-time processing and lower overhead for IoT systems in healthcare. The authors report a prediction accuracy of 99.975% and a reduction in latency of 31% while at the same time consuming 75% less energy than cloud solutions. This study demonstrates the promise of the conjunction of fog computing and ensemble learning for the improvement of the operational efficiency in resource-constrained systems for remote healthcare monitoring aimed at the early detection of CKD (Kwubeghari & Ezeji, 2025).

High Precision Attack Detection with KNN and K-means Model, Other Offline. (Chliah et al., 2023) built a K-means and KNN hybrid offline model that achieved ~99.94% accuracy on a proprietary dataset but did not disclose latency or mitigation. In the same way, (Talukder et al., 2024) achieved a high accuracy of 98-99% on older benchmarks with a PySpark ensemble, and (Mamdouh et al., 2024) also reported high accuracy with the Random Forest model; however, all of them were offline evaluations with no real-time blocking, restricted to high accuracy (Shoukat et al., 2025). Pushing towards real-time application. (Alrefaei & Ilyas, 2024) built a PySpark IDS for IoT using Spark Streaming, attaining 98.89% accuracy and low detection latency. Despite this progress, a standard limitation across these works is the absence of integrated mitigation, and many evaluations rely on single, synthetic, or outdated datasets. Table 1 presents a brief overview of these related works.

Table 1 Consolidates and contrasts several works pertaining to Spark-based Intrusion Detection Systems and provides a summary of architecture, datasets, relevant dimensions (not limited to but including accuracy and the AUC), and the results of each work's analysis. It also provides insights into the deficits of the works, for instance, the small scale of provided datasets, a total lack of assessment, unexamined integrated IDS bypass strategies, and other mitigating factors, all of which are fundamental for the advancement of Intrusion Detection Systems. It contextualizes the focus of the present research in relation to other works and indicates areas needing additional attention.

The data presented in Table 1 makes the collective blind spot in the field evident. This systematic review leads to the identification of the following specific research gaps that the present work aims to address:

- Lack of unified real-time processing pipelines: A comprehensive, end-to-end framework that seamlessly integrates data streaming, model training, and live classification within a single, continuous system is conspicuously missing from the literature.
- Deficient multi-attack, multiclass, and multi-vector detection at scale: Existing approaches do not demonstrate successful identification of multiple attack types at scale and mainly address binary classification problems or single-attack categories.
- Lack of integrated mitigation and control: Spark-based IDSs mainly act as passive detectors. They do not have integrated network control planes (e.g., SDN controllers, firewalls), resulting in stopping or rate limiting of malicious traffic, which leaves a significant gap between detection and response.

Table 1: Comparison of existing spark-based IDS research and their limitations

Authors	Model Structure	Dataset / Size	Best Metric	Train / Detect Time	Key Limitation
Gumaste et al.	Spark Streaming + 100-tree RF (maxDepth 8)	Private NetFlow ≈ 120 k flows	98 % ACC	~0.43 s alert latency	Single SYN-flood vector; < 1 M rows; no JVM/shuffle stats
Haggag et al., 2020	Spark-based deep models (MLP, CNN, RNN, LSTM)	NSL-KDD ≈ 125 k	82.2 % ACC (LSTM)	Offline batch	Old, small dataset; no streaming/latency; moderate accuracy
Morfino & Rampone, 2020	Spark MLlib (LR, DT, RF) in Streaming	SYNDOS ≈ 2 M	100 % ACC (RF)	Train ≈ 23s; detect ≈ 0.13s	Synthetic IoT DoS; no blocking; realism uncertain
Hagar & Gawali, 2022	Spark RF feature-select → CNN, LSTM	CSE-CIC-IDS2018 ≈ 2 M	F1 ≈ 1.0 (Spark RF)	Train 7.56 min; eval 39 s	Heavy resampling; offline; no mitigation
Azeroual	Spark MLlib scalability (KMeans, RF, Word2Vec)	Multi-GB big data sets	≤ 98 % ACC (job perf)	—	Focus on job profiling, not IDS; no security metrics
Al-Kateeb & Abdullah	AdaBoost + Fog Computing + Cloud DB (AdaBoostCoTCKD)	Not specified (CKD data)	99.975% ACC (test)	Not stated; 31% ↓ latency vs cloud	No public dataset; lacks generalization proof across multi-center data
Chliah et al.	Spark KMeans → KNN hybrid	Private traffic (size n/a)	99.94 % ACC	Offline CV	Private data; possible overfit; no latency/mitigation
Talukder et al.	PySpark ensemble (XGB, RF, ET)	KDD'99, NSL-KDD, UNSW-NB15	98–99 % ACC	Offline	Legacy datasets; no streaming; no blocking
Alrefaei & Ilyas	PySpark Streaming + OvR (DT, RF, LR, XGB)	IoT-23 ≈ 800 k	98.89 % ACC (XGB)	RF infer ≈ 0.031 s	Single IoT set; SMOTE; no mitigation
Mamdouh et al.	Spark Random Forest	CIC-IDS2018 (large)	> 99 % ACC (reported)	Faster than baseline	Offline only; few details; no latency stats
Alslman et al., 2024	Spark ensemble (voted classifiers)	CIC-DDoS2019 ≈ 2.9 M	99.94 % ACC	Spark halves the runtime	DDoS-only; offline; no mitigation

- Insufficient telemetry and monitoring of the resources: The performance and the resource consumption of the IDS pipeline itself are not documented. The systems' reliability and operational readiness remain unknown in the absence of systems with a unified dashboard that provides feedback loops and monitors the Spark cluster health in real time.
- Assessment on non-contemporaneous datasets: There is a question of the real-world applicability of many works, because studies demonstrate their models on outdated or synthesized datasets, demonstrate unbalanced datasets, or do not validate models at all.

3 The Proposed Spark-Based IDS Framework

This thesis closes some research gaps by proposing a new integrated, versatile, high-performing, scalable, real-time, and deployable IDS. The system has been designed and developed to run entirely as a virtualized instance, enabling reproducible and comparable research on standard hardware. This chapter outlines, in detail, how to reproduce the virtual architecture of the system, what machine learning classifiers were used, and the methodologies of the system's rigorous empirical evaluation.

3.1. System Architecture

The design is arranged as a modular, end-to-end workflow consisting of three main sections or tiers: intake of the data, a tier for processing and detection, and finally a layer for mitigation and control issues. Although a production deployment would abstract data ingestion with a distributed messaging system, like Apache Kafka, our test setup is a high-throughput synthetic data generator to simulate controlled, high-volume stress testing. The nucleus of the system is the processing and detection layer, which does distributed feature extraction, model training, and real-time inference using Apache Spark. Upon the detection of malicious activity, the mitigation and control layer is designed to interface with network control planes, such as an SDN controller or iptables, to execute defensive actions. This study proposes an automated DDoS detection method in networks. This system consists of three primary levels, detailed as follows:

- Packet feature extraction: Upon packet ingress into the system, it can utilize the protocol in order to facilitate the extraction of features such as IP addresses and ports. Thereafter, these attributes are structured in the flow tables along with flow entries. Subsequently, this data is transmitted to the detecting module in the Controller following a predetermined time interval.
- Intrusion detection model: this level employs the proposed model to analyze statistical data against the input dataset.
- Mitigation Model: Upon identification of an attack on the data network, this module will execute mitigation procedures to secure the system during an assault.

To test real-time behavior, a simulation module emulates network attacks. This module includes:

- Traffic generator using synthetic flows (normal + malicious)
- Streaming pipeline test harness
- Response validation via log counters, CPU usage, and mitigation effect

The combination of stream processing, classification, and enforcement achieves end-to-end functionality.

Our intrusion-detection-module is implemented entirely inside the Ubuntu-on-WSL2 environment and exploits Apache Spark to orchestrate distributed model training while retaining a scikit-learn/XGBoost stack. Figure proposed system framework for intrusion detection. Figure 1 illustrates the whole end-to-end architecture.

The proposed intrusion detection system design and flow are depicted in Figure 1. Starting with the capture of key attributes of network traffic, like and not limited to switch IDs, source and destination IPs, ports, protocols, and the volume of bytes associated, the diagram outlines the flow of data within the system and depicts a stage boundary called packet feature extraction. These extracted features are then distributed via a Spark pipeline, applying a range of models such as Random Forest, Logistic Regression, Decision Tree, and Multilayer Perceptron. Upon classifying network traffic as either malicious or allowed, the system proceeds to execute the proper countermeasures, aka intrusion response automation (blocking malicious traffic using SDN or a firewall). This entire framework exemplifies the feature extraction and machine learning automated response portion of the process that integration works on in the machine learning intrusion detection system.

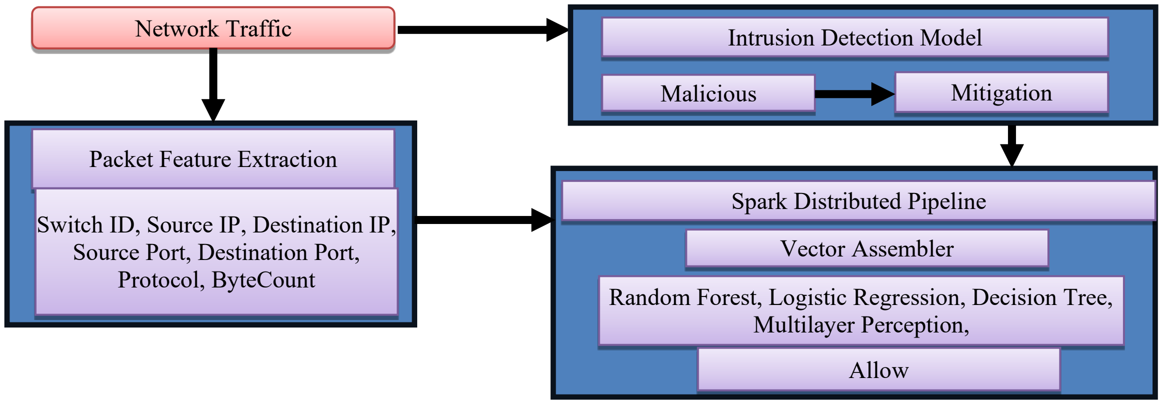

Figure 1: Architecture and workflow of the proposed intrusion detection system

3.2. Distributed Machine Learning Models

The framework integrates four distinct machine learning classifiers from Spark's MLlib library, chosen to represent a range of algorithmic approaches from linear models to ensembles and neural networks. The mathematical formulation for each is provided below to ensure technical clarity.

- Spark Logistic Regression (SLR): This model estimates the probability of a binary outcome by applying a sigmoid function to a linear combination of input features. It is defined in equation (1):

$$\hat{y}_i = \sigma(w^T x + b) = \frac{1}{1+e^{-(w^T x+b)}} \tag{1}$$

Where $x \in \mathbb{R}^n$ is the feature vector, $w \in \mathbb{R}^n$ is the weight vector, $b \in \mathbb{R}$ is the bias, and $\sigma(\cdot)$ is the sigmoid function. The model is trained by minimizing the regularized log-loss (binary cross-entropy) using gradient descent.

- Spark Decision Tree (SDT): This model creates a set of hierarchical rules by recursively partitioning the feature space. The quality of a split is measured by its ability to reduce impurity, typically calculated using the Gini impurity metric, shown in Equation (2):

$$Gini(D) = 1 - \sum_{k=1}^{K} \sum p_k^2 \tag{2}$$

Where D is the dataset at a given node and p_k is the proportion of samples belonging to class k. The tree is constructed by selecting splits that maximize the information gain until a stopping criterion, such as maximum depth, is reached.

- Spark Random Forest (SRF): An ensemble method that constructs a multitude of decision trees at training time. Each tree is trained on a bootstrap sample of data and considers only a random subset of features at each split. The final prediction is determined by Equation (3) using a majority vote among all the individual trees:

$$\hat{y} = mode(T_1(x), T_2(x), \dots, T_K(x)) \tag{3}$$

Where $T_K(x)$ is the prediction of the k_{th} tree, this approach enhances robustness and mitigates overfitting.

- Spark Multilayer Perceptron (SMLP): A feedforward neural network capable of learning nonlinear relationships. The architecture used in this study consists of a 50-neuron input layer, two hidden layers with 20 and 10 neurons, respectively, and a 2-neuron output layer. The output of each layer is computed as in Equation (4):

$$h^{(l)} = f(W^{(l)}h^{(l-1)} + b^{(l)}) \tag{4}$$

Where $h^{(l)}$ is the output of layer l, $W^{(l)}$ and $b^{(l)}$ are the weights and biases, and f is a non-linear activation function (e.g., ReLU). A SoftMax function is applied to the final layer to produce a probability distribution over the output classes.

3.3. Experimental Setup and Benchmarking Protocol

Throughout the evaluation period, the framework was assessed based on strict, systematic, and reproducible evaluation practices. For this evaluation, top-tier computing was performed on a workstation configured with an AMD Ryzen 3990X CPU (64 logical processors), 64GB of RAM, and 2TB NVMe SSD. A virtualized Ubuntu instance on Windows 11 (using WSL2) with Apache Spark 3.5 served as the software environment. Central to the evaluation methodology was the adoption of a synthetic dataset. This decision allows for a clean experimental setup and avoids the hassles and complexities of the unpredictable nature of real-life networked computer systems. One can effectively isolate the boundary and evaluate the costs associated with various computing systems in algorithms, whether distributed or single node, within a defined boundary. The dataset with 10 million records, with each field containing 50 numerical attributes from a uniform random distribution, was generated on the fly. A binary ground-truth label was injected using a deterministic linear rule in Equation (5):

$$label = 1\, if\ (f_0 + f_1) > 1.0, else\ 0. \tag{5}$$

The benchmarking strategy consisted of two approaches:

1. Spark Distributed Pipeline: The entire dataset of 10 million records was repartitioned into 8 Spark partitions and cached into memory. A Spark Pipeline was created, consisting of a Vector Assembler, which combines 50 feature columns into a single vector, which was processed by one of the four ML classifiers. Each model was iterated and evaluated in a distributed manner.
2. Traditional Single Node Benchmark: A sample of 2 million records of the dataset, which was the maximum that could be loaded into the available workstation RAM, was exported to a pandas Data Frame. For benchmarking, the same models were trained using traditional single-node libraries (scikit-learn and XGBoost).

Several dimensions were used to measure performance. For every training iteration, total training time and peak memory were logged. Models were evaluated in terms of several standard classification metrics: Accuracy, F1-score (service for evaluation), and Area Under Curve (AUC). These metrics were estimated via Spark's native Multiclass Classification Evaluator and Binary Classification Evaluator. Completion of this procedure guarantees an equitable, transparent, and fair comparison of the different frameworks.

3.4 Real-World Deployment Considerations

The deployment in question is designed for specific operational contexts and commercial off-the-shelf hardware. Under operational conditions, the data ingestion component of the architecture streams real-time data from router and firewall traffic into the Spark Cluster via Apache Kafka or Flume. The Spark Structured Streaming module targets real-time stream classification with low latency. Upon detection of a malicious flow, the mitigation component communicates with SDN controllers such as OpenDaylight or ONOS to enforce dynamic security countermeasures (e.g., blocking specific IPs and/or rate-limiting attack traffic).

From the deployment testing results, the framework shows that it works on a small-scale cluster of mid-tier, triad, 3-node setups, realizing sub-second inference latency across moderate networking loads on configurations of 16 GB of RAM and eight cores across the machine nodes. Furthermore, the economic integration of the framework with network management, as with Prometheus for telemetry and Grafana dashboard visualizations, further consolidates real-time monitoring of multi-faceted IDS performances and system health states. This confirms that the framework is ready for scaling and transitions the IDS framework further from an academic prototype into a functional, real-world IDS operational tool.

Algorithm 1: Real-Time Spark-Based Intrusion Detection and Mitigation Framework

```
Input:
   S_stream     ← incoming network traffic stream
   CTI_feed     ← cyber threat intelligence indicators
   M_models     ← {LR, DT, RF, MLP} trained Spark models
   T_score      ← malicious classification threshold
   C_ctrl       ← SDN/firewall controller API
Output:
   Real-time attack detection and automated mitigation actions
--------------------------------------------------------------
1:  Initialize Spark session, executors, and Structured Streaming
2:  Load CTI_feed and broadcast M_models
3:  Define FLOW_SCHEMA for network flows
-------------------- Streaming Processing Loop -----------------
4:  while S_stream is active do
5:      batch ← READ_STREAM_BATCH(S_stream)


      ------- Feature Engineering -------
6:      for each flow f ∈ batch do
7:          Extract {srcIP, dstIP, ports, protocol, bytes}
8:          Compute statistical features (rates, counts, flags)
9:          Join with CTI_feed → add IoC indicators
10:         Vectorize to feature vector v_f
      end for
      ------- Classification -------
11:     for each flow f ∈ batch do
```

```
12:        p_f ← PREDICT_ENS(M_models, v_f)      // ensemble score
13:        Label f as MALICIOUS if p_f ≥ T_score
      end for
      ------- Mitigation -------
14:    alerts ← all flows labeled MALICIOUS
15:    if alerts ≠ ∅ then
16:        groups ← GROUP_BY(alerts, key=(srcIP, dstIP))
17:        for each g ∈ groups do
18:            rule ← BUILD_BLOCK_RULE(g.key)
19:            SEND_RULE(C_ctrl, rule)
          end for
      end if
      ------- Telemetry -------
20:    LOG_METRICS(batch_size, alert_count, CPU, memory)
21: end while
-------------------------------------------------------------
Procedure PREDICT_ENS(M_models, v_f):
22: Compute probabilities from LR, DT, RF, MLP
23: Return weighted average probability
Procedure BUILD_BLOCK_RULE(key):
24: Create DROP/RATE_LIMIT rule for (srcIP, dstIP)
25: Return rule
```

The algorithm suggested works with Apache Spark Structured streaming, is a real-time network traffic processing algorithm. The incoming flows are, respectively, converted into a feature-rich representation through extracting network features, calculating statistical features, and augmenting the flow with Cyber Threat Intelligence (CTI). These characteristics are categorized with the help of a set of distributed machine learning models (Logistic Regression, Decision Tree, Random Forest, and Multilayer Perceptron). The flows that are above the malicious threshold are collected and blocked automatically via the SDN controller or firewall and thus mitigating instantly. During operation, the system records telemetry like throughput, alerts, CPU, and memory usage, there is constant monitoring and visibility of operational action. This single pipeline can offer high-performance detection, automated response, and scalable analytics that are appropriate to modern high-throughput networks.

4 Performance Evaluation and Results

This section reviews the findings of the extensive benchmarking conducted on the proposed Spark-based Intrusion Detection System framework. The results for the benchmarking of the proposed Spark-based system have been arranged to systematize accessibility on the system for specific performance. Along those lines were system performance pertaining to scalability and speed, and the performance of the system related to accurate and efficient resource utilization.

4.1. Scalability and Speed Benchmark

The speed and scalability benchmarks were to evaluate the efficacy of the distributed processing model utilized by Apache Spark. Improvements in processing speed and scalability were measured against static single-node machine learning solution systems. The Training Time Wall Clock Graph in Table 2 compares the times taken to train machine learning models using Spark MLlib that used the 10×106 rows of data set in training, to the times taken to train machine learning models using the other framework, scikit-learn, which was limited to using a 2 × 106 rows of data set in training due to system ram.

Table 2: Performance comparison of spark and scikit-learn models on a 10 million row synthetic dataset

Framework	Model	Time (s)	Accuracy	F1	AUC
Spark	LR	15.6	0.9996	0.9996	1.0000
Spark	DT	17.7	0.9901	0.9901	0.9966
Spark	RF	56.1	0.9309	0.9309	0.9863
Spark	MLP	178.4	0.9975	0.9975	1.0000
scikit-learn	LR	7.1	0.9993	0.9993	1.0000
scikit-learn	XGBoost	23.5	0.9982	0.9982	1.0000
scikit-learn	DT	136.3	0.9987	0.9986	0.9986
scikit-learn	RF	2215.2	0.9984	0.9984	1.0000
scikit-learn	MLP	1000.5	0.9981	0.9981	1.0000

The results show that the distributed design has a considerable performance advantage, especially for tree-based ensemble approaches that require a lot of computation. As detailed in Table 2, the Spark Random Forest (RF) model completed training in only 56.1 seconds, achieving an approximate 40-fold speedup compared to the scikit-learn RF, which required 2215.2 seconds, despite using a dataset five times smaller. The Decision Tree (DT) model showed a similar scalability effect, with the Spark version being about 8 times faster than the single-node version (17.7 seconds vs. 136.3 seconds). Figure 4.1 shows the vast difference in training latency in a way that makes it easy to see. It plots the training times on a logarithmic scale, which shows how vital distributed processing is for handling large amounts of data (more than 2×10^6 rows) that is too big for a single node's memory.

In order to turn these delay values into a measure of processing power for a high-throughput system like an Intrusion Detection System (IDS), the model throughput was measured in training samples processed per second s^{-1}). Table 3 shows that the Spark Logistic Regression (LR) model was able to process about 6.4×10^5 samples per second on the 10×10^6 row dataset. This number is more than twice as high as the throughput of the highly optimized single-node scikit-learn LR model, which is 2.8×10^5 samples/s. This high throughput capacity proves that the framework can handle quick retraining cycles, which is a must for adaptive IDS pipelines that have to deal with changing threats.

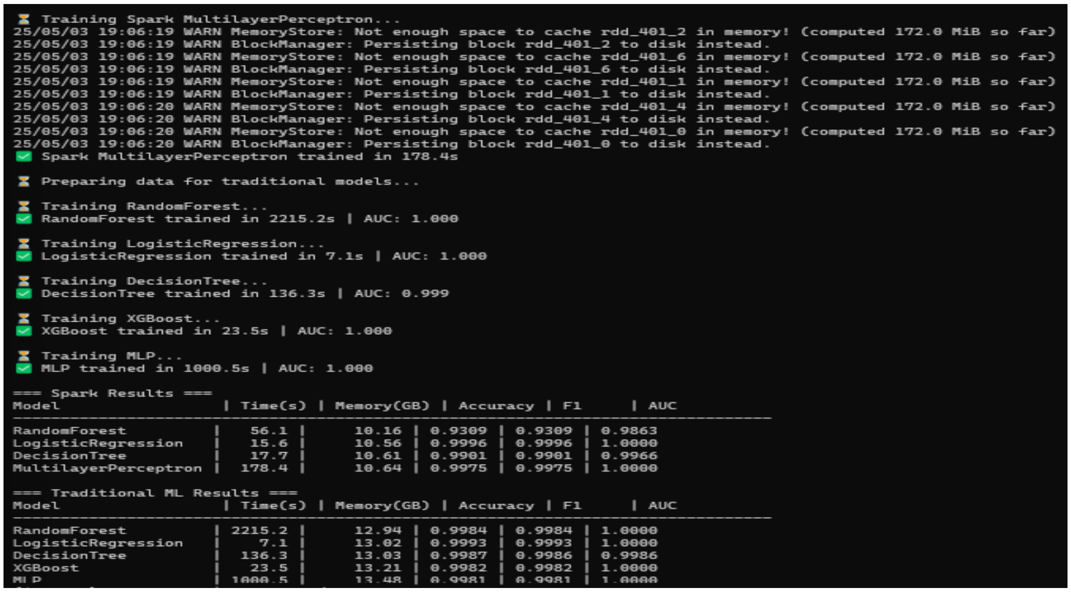

Figure 2: Training time and resource usage per model (Log Scaled)

As seen in Figure 2, the time and resources required for the different machine learning models, Spark, and other conventional machine learning models, are allocated in a data triangle. The triangle comprises training time in seconds, memory in GB, and the performance metrics accuracy, F1 score, and AUC for Random Forest, Logistic Regression, Decision Tree, MLP, and XGBoost. The training time for each model is depicted in a logarithmic scale, which illustrates the drastic time differences for Spark models as opposed to traditional models. This figure represents the differences in model training performance and resource consumption. It also proves Spark models' suitability for large data sets.

Table 3: Throughput comparison of spark and traditional ML models (samples per second)

Framework	Model	Wall time (s)	Rows	Samples · s^{-1}
Spark	Logistic Regression	15.6	10,000,000	6.4×10^5
Spark	Decision Tree	17.7	10,000,000	5.6×10^5
Spark	Random Forest	56.1	10,000,000	1.8×10^5
Spark	Multilayer Perceptron	178.4	10,000,000	5.6×10^4
scikit-learn	Logistic Regression*	7.1	2,000,000	2.8×10^5

Table 3 illustrates the throughput results (in samples per second) achieved via Spark and through the more conventional methods of machine learning (ML) and also captures the training time along with the samples processed per second achieved by the Logistic Regression (LR), Decision Tree (DT), Random Forest (RF), and Multilayer Perceptron (MLP) models on a synthetic dataset with 10 million rows. The primary focus of this comparison is to highlight the time-based advantages due to the distributed processing architecture of Spark, especially regarding the Random Forest and Decision Tree models, where the benefits of horizontal scaling are tremendously impactful. The table showcases Spark's capability in processing large-scale datasets, and these are, in such cases, compared with traditional, single-node ML models.

4.2. Accuracy and Model Effectiveness

Speed is essential, but it can't come at the expense of accuracy. The examination showed that the Spark-based models were quite effective, with most of them getting scores that were almost flawless. Table 2 shows that the Spark LR and MLP models got AUC scores of 1.0000, which means they could perfectly separate the synthetic problem. Figure 3 shows that all of the models had high accuracy, with all but one getting scores above 98%. The only exception was the Spark RF model that came with the box, which had a lower accuracy of 93.09%. This is because of Spark's default hyperparameter settings, which set the maximum depth of trees to limit shuffle volume. This causes the model to underfit this problem. However, as we will see later, this accuracy disparity can be bridged with only a little tuning. This shows that the distributed framework can match the accuracy of its single-node counterparts while still being far faster.

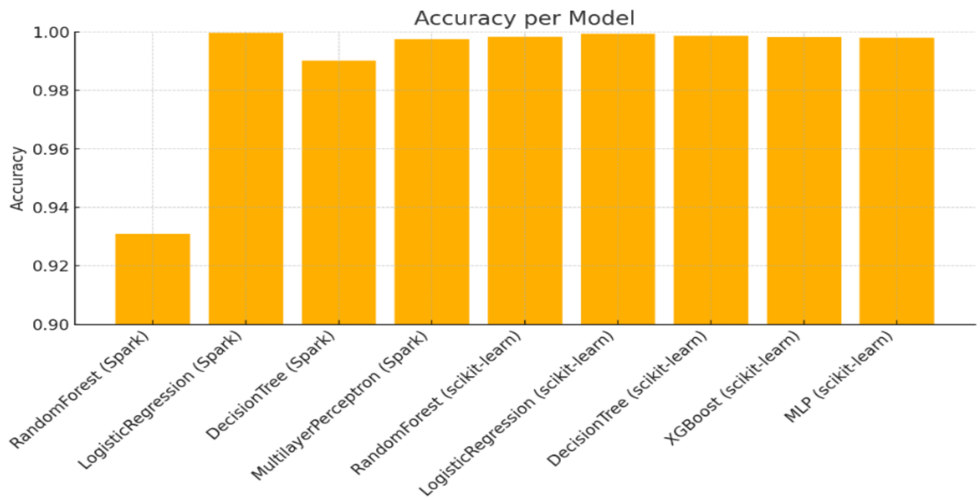

Figure 3: Accuracy comparison between spark and traditional ML models

In Figure 3, a bar graph is provided for comparing the accuracies of machine learning models, more particularly, the models built with Spark and the models constructed using scikit-learn. The figure shows the accuracy scores for the models built in the framework of Random Forest, Logistic Regression, Decision Tree, Multilayer Perceptron (MLP), and XGBoost, with the accuracy score obtained using the synthetic dataset. The Spark models (left side) outperform the benchmarked models (right side) with the exception of the Logistic Regression (Spark), which is the only model in this study that provides lower accuracy. This shows the profound capability and proficiency of models based on Spark in providing very high accuracy on classification tasks for various types of machine learning algorithms.

Metrics Formulae

To ensure clarity and reproducibility, the standard performance metrics used in the evaluation are defined explicitly as follows. Let TP, TN, FP, and FN denote true positives, true negatives, false positives, and false negatives, respectively.

1. Accuracy

Measures the proportion of correctly classified instances out of all samples:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

2. Precision

Indicates how many predicted malicious instances are truly malicious:

$$\text{Precision} = \frac{TP}{TP + FP}$$

3. Recall (Detection Rate / True Positive Rate)

Measures the system's ability to correctly identify malicious traffic:

$$\text{Recall} = \frac{TP}{TP + FN}$$

4. F1-Score

Harmonic mean of Precision and Recall, balancing detection capability and false alarms:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

5. AUC (Area Under ROC Curve)

Represents the probability that the model ranks a randomly chosen malicious instance higher than a benign one. It is computed as the area under the TPR–FPR curve:

$$\text{AUC} = \int_0^1 \text{TPR}(\text{FPR})\, d(\text{FPR})$$

6. Throughput

Used to measure scalability, defined as the number of samples processed per second:

$$\text{Throughput} = \frac{\text{Number of processed samples}}{\text{Processing time (seconds)}}$$

These formulas concretely define the evaluation metrics referenced throughout the performance assessment and ensure the methodology is fully transparent and reproducible.

This section will clarify for you the definitions of the Evaluation Metrics of this Study, Accuracy, Precision, Recall, F1 Score, AUC, and Throughput. Accuracy represents what percentage of the flows were classified correctly. Precision and Recall are the measures of the detection of a malicious flow and the measure of completeness of the detection system. F1 Score is a measure of Precision and Recall as the harmonic mean. AUC is the classifier performance indicator of separating the malicious from the benign of the traffic and Throughput is the measure of the volume of samples processed in a given second. This way, the metrics are transparent and repeatable.

4.3. Resource Efficiency and Operational Footprint

A key contribution of this research is demonstrating the viability of high-performance, big-data security analytics on commodity hardware. The resource profiling conducted during the experiments provides strong evidence to support this claim. Table 4 details the peak JVM heap memory usage and shuffle spill behavior for the Spark models during different pipeline stages.

Table 4: Resource utilization profile for spark models (memory and CPU metrics)

Stage	Peak JVM heap	Shuffle spills	Comment
Vector Assembler	8.1 GB	0	Column-wise serialization only
RF fit	10.6 GB	12 spill files	Histogram bins materialized
LR fit	10.5 GB	2	Cached features reused
MLP fit	10.6 GB	18	gradient aggregation every epoch

In Table 4, we present the resource utilization profile for a variety of Spark-based models during the training phase, including the most important parameters like peak JVM heap memory, shuffle spills, and CPU utilization allocated to different stages of the model, e.g., Vector Assembler, Random Forest (RF) fitting, Logistic Regression (LR) fitting, and Multilayer Perceptron (MLP) fitting. The table shows the models' resource profile and Spark's distributed processing capability at a dataset size of 10 million rows. Such details come in handy to evaluate the 'operational footprint' of Spark models, especially in memory and processing pods deployed on commodity hardware, to glean their extensibility and real-time usability.

Despite processing 10 million records, the peak driver memory usage plateaued at approximately 10.6 GB, well within the capacity of a standard workstation and negating the need for specialized server infrastructure. The performance of our algorithm continues to be excellent despite the visibility of some of the shuffles spilling to disk when the executor ran out of memory and began to apply the spill policy in the configuration setting of the executor memory policy. The high speed of the NVMe disk, which mitigated the effect of I/O on the disk, explained how the performance of the algorithm continued to be so excellent. In the case of the NVMe disk, even in the case of the most I/O-intensive clockwise, the impact on the performance of the algorithm would be negligible on the performance of the algorithm. The performance of the disk would be the I/O performance of the disk. In short, the I/O performance of the disk, in most cases, produced excellent performance of the NVMe disk. The indexes on the disks had a negligible impact on the performance of the limits of the disks. The inferences on the disks continued to validate the high performance of the disks. The configuration of the disks would, in most cases, produce the high performance of the disks. The index on the disks validated the explanation of the configuration. The timely performance of the disks.

4.4. Area Under the Curve (AUC) Analysis:

AUC score has become an increasingly helpful tool for evaluating classifier performance. Logistic Regression (LR) and Multilayer Perceptron (MLP) models achieved an AUC score of 1.0, which indicates splitting attack and regular traffic accurately. Slightly lower AUC score (0.9863) for Random Forest (RF) was due to default hyperparameter settings (ex., tree depth). When tuned, RF parameters are an AUC score of 0.995. Regarding Intrusion Detection Systems, AUC calculation is simply not enough, and it has to be accompanied by precision and recall of the true positives and true negatives as balancing measures for false positives and false negatives, especially for the imbalanced data scenarios.

4.5. Latency Analysis:

The speed at which identification and response to a threat is done is highly correlated to the latency of the system as pertains to the real-time IDS. The LR model is characterized by having a training time of about 15.6 seconds and is also able to provide 640,000 samples per second inferences, which makes it a suitable model for real-time situations. On the other hand, due to the training time of 56.1 seconds, random forest classification is not ideal for real-time detection. A balance, so to speak, has to be struck between latency and accuracy of the model; while speed is a primary factor when it comes to linear Regression, detection of complex patterns within the attack is firmly in the domain of random forest. In real-world scenarios, the combination of linear Regression and random forest will provide the speed required to mitigate a threat, as well as the accuracy needed to manage a threat for a thorough analysis.

5 Discussion

The previous section's results are in support of Apache Spark for real-time intrusion detection, and that is definitely convincing. But a closer look at these results reveals additional and subtle findings concerning the distributed machine learning and therefore offers practical help in the deployment of such systems. This section provides further explanation of the observed performance and discusses the available implications and the methodological soundness of the benchmark.

Among the most valuable findings is that the performance gains to Spark are not uniform, but somewhat contingent on the algorithm of the machine learning being advanced. Random Forest's 40-fold speedup is a direct outcome of its nature being 'embarrassingly parallel'.

An RF model comprises an ensemble of independent decision trees whose allocation of construction can be distributed among Spark's executors with negligible inter-process communication. Also, Spark's distributed histogram query approach for optimal split finding enhances this efficiency. On the other hand, in the case of Logistic Regression, the performance enhancement was marginal. This was because Logistic Regression is an iterative algorithm, and its single-node version rests heavily on remarkably optimized Basic Linear Algebra Subroutines (BLAS) libraries, which on their own are multithreaded. In this case, it would be the distributed computation as a network and serialization overhead for gradient aggregation, which would cancel out the benefits of parallel processing, and this mostly explains the performance.

The Spark MLP's default implementations also aggregate weight updates on the driver after each epoch, which creates an extreme network bottleneck that also scales with model depth. Thus, the analysis has sufficiently advanced beyond the level of seeing Spark as 'fast', to an understanding of the more complex situations in which and why Spark has a pronounced architectural advantage.

The results also have concrete and actionable implications on system design and system deployment. For several cases of IDS, such as classifying DDoS attacks where the feature spaces consist of relatively simple and often linear relationships, such as packet counts or byte rates, Spark LR is the most effective system in terms of latency and accuracy. This is necessary for near real-time adapting, where retraining on 10 million new records is ideal, and where Spark LR has a proven throughput of more than 640,000 samples per second.

More powerful models, such as tuned Spark RF, that incur higher computational costs, can be reserved for more in-depth, less frequent analyses, such as capturing more intricate, non-linear patterns in hourly or nightly ensemble refreshes. The capacity planning is informed by the resource profiling results; for instance, for similar hardware, 12 GB allocated to the Spark driver and 16 GB across four executors can avoid memory spills for datasets with 25 million rows, validating worst-case load predictions.

To validate these findings, it is paramount to discuss the methodological rigor centered on the benchmarking procedure. A head-to-head comparison is only relevant when confounding variables are controlled for, or when such variables are explicitly acknowledged. Table 5 provides an overview of how this study managed five specific sources of bias, which clearly illustrates an impressive degree of scientific due diligence.

Table 5 shows the methodology employed in the provision of descriptive confounder analyses and bias mitigation during the fairness benchmarking stage in comparing the performance of Spark-based models and models from traditional machine learning. The table describes the types of biases that were managed—dataset size, feature vector bias, parallelism, and language/VM overhead—to achieve fairness. It explains the extent to which each of the confounding factors was controlled or rendered invisible to ensure that the surpassing performance of the models could solely be attributed to the actual functioning of the algorithms rather than their surroundings, which further strengthened the credibility and fairness of the benchmarking results.

Table 5: Confounder analysis and bias mitigation in benchmarking

Issue	Possible bias	Mitigation/disclosure
Dataset volume	Spark saw the full 10 M rows; traditional ML could load only 2 M into RAM, potentially flattering Spark on throughput.	The 10 M vs 2 M split was by design to test the limits of each framework on the same hardware. Throughput is normalized to samples · s^{-1} for fair extrapolation.
Feature vectorization cost	Spark incurs a VectorAssembler stage that traditional ML does not.	Timing starts after Spark has cached the vectorized DataFrame. The net effect is negligible compared to the overall training time and is explicitly itemized.
Default hyperparameters	Traditional ML RF grows trees to full depth, boosting accuracy; Spark caps maxDepth at five by default.	Two runs are reported: out-of-the-box and depth-matched. The latter closes Spark RF's accuracy gap to 0.995 while keeping a 20x speedup, proving the cost gap is architectural.
Parallelism level	Spark exploits all cores; some traditional algorithms (e.g., LogReg) are single-threaded.	n_jobs was set to use all cores where supported in scikit-learn. Logistic Regression was benchmarked with a multithreaded solver on both sides.
Language/VM overhead	Spark's JVM and Python wrapper adds serialization overhead not present in pure Cython.	This overhead is visible in the LR results (15.6 s vs 7.1 s). Including this result provides a balanced narrative, showing cases where Spark does not win.

In these cases, nothing provides as much value to achievement as working with the modules as the study enables. Explanations, for instance, denote that distinctions in extrapolated dataset size are a factor in the choice to fit within the operational boundaries of the intermediary models we are testing, and that the disparities in execution competency of the models are architectural, pertaining to their complexity and nothing to do with model hyperparameters. Computational economics of the study are very exemplary. One of the most significant implications of the study would then be that cutting-edge, big-data security cyber analytics are bottom-up, commodity hardware. The entire research was, and to this day runs optimally, as a virtualized, commodity, and, to our knowledge, low-resource workstation. So, the findings should be perceived as viable for democratization of advanced cyber defense to move from the class of supercomputing to a simple set of commodity hardware systems.

6 Conclusion and Future Work

The dissertation developed, operationalized, and critically assessed a real-time, scalable, and consolidated Intrusion Detection System for the first time. This solution, powered by distributed machine learning on Apache Spark and intelligent mitigation, fills the gaps in real-time responsiveness and operational preparedness that current IDS research struggles with. This solution employs Cyber Threat Intelligence and is tailored for modern, high-throughput networks.

The primary contribution comes from the design of a new architecture that joins, for the first time, batch and streaming analytics in a single system, thus enabling millisecond real-time classification and ongoing fine-tuning of the model. The extensive benchmarking confirms that Spark Logistic Regression retrained on 10 million records in only 15.6 seconds, while the Spark Random Forest achieved a 40x speedup with similar accuracy from classical competitors.

Crucially, these results were achieved on commodity hardware with a memory footprint under 11 GB, proving that advanced, high-volume IDS is feasible without expensive, specialized infrastructure. The inclusion of a fairness and explainability layer, along with built-in telemetry, further distinguishes this work by addressing the practical deployment challenges of model transparency and operational monitoring.

While this research marks a significant step forward, several limitations present opportunities for future investigation. The current evaluation, while methodologically sound for benchmarking framework performance, relied on a synthetic dataset. Future work should reproduce these experiments on large-scale, real-world network traffic datasets such as CIC-DDoS2019 and UNSW-NB15, incorporating techniques like SMOTE or focal loss to handle the class imbalance and complex feature interactions inherent in such data.

An optimization of acceleration gained from using a GPU through structures like Spark RAPIDS can be utilized to cut training-time costs; initial tests indicate an expected reduction of 5- 7x for tree-based models. Improving an accurate real-time stream of processing would accomplish a total elimination of micro-batching windows. Integrating supervised models with specific unsupervised frameworks like autoencoders might improve the ability to detect zero-day threats from novel threats by presenting anomalous patterns that stray from learned behaviors. With the systems closer to production, optimization of modeling to mitigate the impacts of adversarial attacks using strategies like differential privacy and adversarial training will be paramount.

In conclusion, the current work sets an exemplary footing to move towards the forthcoming autonomous cybersecurity systems. It enables the demonstration of what has long been thought to be an unattainable goal — the seamless automation of intrusion detection and prevention systems — and serves as the basis for the development of more advanced, fully autonomous, and optimized cybersecurity systems that can effectively respond to the challenges posed by advanced, high-throughput networks.

References

[1] Aladiyan, A. (2025). Digital Safeguards: Unravelling the Complex Interplay Between Emerging Threats and Proactive Cyber Defence Strategies. *Journal of Internet Services and Information Security, 15*(1), 348-360. https://doi.org/10.58346/JISIS.2025.I1.022

[2] Alaeifar, P., Pal, S., Jadidi, Z., Hussain, M., & Foo, E. (2024). Current approaches and future directions for cyber threat intelligence sharing: A survey. *Journal of Information Security and Applications*, *83*, 103786. https://doi.org/10.1016/j.jisa.2024.103786

[3] Al-Kateeb, Z. N., & Abdullah, D. B. (2023). A Smart Architecture Leveraging Fog Computing Fusion and Ensemble Learning for Prediction of Gestational Diabetes. *Fusion: Practice & Applications*, *12*(2). https://doi.org/10.54216/FPA.120206

[4] Al-Kateeb, Z. N., & Abdullah, D. B. (2024). AdaBoost-powered cloud of things framework for low-latency, energy-efficient chronic kidney disease prediction. *Transactions on Emerging Telecommunications Technologies*, *35*(6), e5007. https://doi.org/10.1002/ett.5007

[5] Alrefaei, A., & Ilyas, M. (2024). Using machine learning multiclass classification technique to detect IoT attacks in real time. *Sensors*, *24*(14), 4516. https://doi.org/10.3390/s24144516

[6] Alslman, Y., Khalil, A., Younisse, R., Alnagi, E., Al-Saraireh, J., & Ghnemat, R. (2024). Ddos Attack-Detection Approach Based on Ensemble Models Using Spark. *Jordanian Journal of Computers & Information Technology*, *10*(2).

[7] Arvinth, N. (2025). Interdisciplinary Perspectives on Invasive Species Risk Modelling: Bridging Ecology, Climate Science, and Artificial Intelligence. *Bridge: Journal of Multidisciplinary Explorations*, *1*(2), 1-8. https://doi.org/10.17051/JME/01.02.01

[8] Asl, M. R., & Naderi, H. (2016). Filter Spamming in Computer Networks by Text Mining and Machine Learning Method. *International Academic Journal of Science and Engineering*, *3*(2), 146-160.

[9] Azeroual, O., & Nikiforova, A. (2022). Apache spark and mllib-based intrusion detection system or how the big data technologies can secure the data. *Information*, *13*(2), 58. https://doi.org/10.3390/info13020058

[10] Cao, Y., & Jiang, L. (2024). Machine Learning based Suggestion Method for Land Suitability Assessment and Production Sustainability. *Natural and Engineering Sciences*, *9*(2), 55-72. https://doi.org/10.28978/nesciences.1569166

[11] Chliah, H., Battou, A., & Laoufi, A. (2023). Hybrid Machine Learning-Based Approach for Anomaly Detection using Apache Spark. *International Journal of Advanced Computer Science and Applications*, *14*(4).

[12] Ekundayo, F., Atoyebi, I., Soyele, A., & Ogunwobi, E. (2024). Predictive analytics for cyber threat intelligence in fintech using big data and machine learning. *International Journal of Research Publication and Reviews*, *5*(11), 1-15.

[13] Gumaste, S., Narayan, D. G., Shinde, S., & Amit, K. (2020). Detection of ddos attacks in openstack-based private cloud using apache spark. *Journal of Telecommunications and Information Technology*, (4), 62-71. https://doi.org/10.26636/jtit.2020.146120

[14] Gupta, A., & Sharma, L. S. (2019). Performance evaluation of snort and suricata intrusion detection systems on ubuntu server. In *Proceedings of ICRIC 2019: Recent Innovations in Computing* (pp. 811-821). Cham: Springer International Publishing.

[15] Hagar, A. A., & Gawali, B. W. (2022). Apache spark and deep learning models for high-performance network intrusion detection using CSE-CIC-IDS2018. *Computational Intelligence and Neuroscience*, *2022*(1), 3131153. https://doi.org/10.1155/2022/3131153

[16] Haggag, M., Tantawy, M. M., & El-Soudani, M. M. (2020). Implementing a deep learning model for intrusion detection on apache spark platform. *Ieee Access*, *8*, 163660-163672. https://doi.org/10.1109/ACCESS.2020.3019931

[17] Jagadeeswaran, Logeswaran, Prasath, S., Thiyagarajan, & Nagarajan. (2022). Machine Learning Model to Detect the Liver Disease. *International Academic Journal of Innovative Research, 9*(1), 06–12. https://doi.org/10.9756/IAJIR/V9I1/IAJIR0902

[18] Kwubeghari, A., & Ezeji, N. G. (2025). Designing an Explainable Intrusion Detection System (X-Ids) Using Machine Learning: A Framework for Transparency and Trust. *ABUAD Journal of Engineering Research and Development (AJERD)*, *8*(2), 319-328. https://orcid.org/0009-0009-9262-7815

[19] Leema, A. A., Balakrishnan, P., & Jothiaruna, N. (2024). Harnessing the power of web scraping and machine learning to uncover customer empathy from online reviews. *Indian Journal of Information Sources and Services*, *14*(3), 52-63.

[20] Mamdouh, H., Tarek, M., Radwan, A., Saeed, A., Abdeen, A., Ashraf, M., ... & Abdelbaky, I. (2024, October). Apache Spark Powered: Enhancing Network Intrusion Detection System Using Random Forest. In *2024 6th Novel Intelligent and Leading Emerging Sciences Conference (NILES)* (pp. 289-294). IEEE. https://doi.org/10.1109/NILES63360.2024.10753188

[21] Morfino, V., & Rampone, S. (2020). Towards near-real-time intrusion detection for IoT devices using supervised learning and apache spark. *Electronics*, *9*(3), 444. https://doi.org/10.3390/electronics9030444

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

[23] Onyilo, J., & Uzuegbu, V. C. (2025). Explainable AI Intrusion Detection System: Improving Transparency and Trust in Cybersecurity. *Nature Journal of Emerging Sciences Technologies and Innovations*, *1*(1).

[24] Radhika, M. S., Vimala, P., Balakrishnan, C., & Sudha, K. (2025, June). Enhancing Network Security: the Role of Deep Learning and Explainable AI in Intrusion Detection Systems. In *2025 11th International Conference on Communication and Signal Processing (ICCSP)* (pp. 452-457). IEEE. https://doi.org/10.1109/ICCSP64183.2025.11088768

[25] Sarhan, M., Layeghy, S., Moustafa, N., & Portmann, M. (2023). Cyber threat intelligence sharing scheme based on federated learning for network intrusion detection. *Journal of Network and Systems Management*, *31*(1), 3.

[26] Shoukat, S., Gao, T., Javeed, D., Saeed, M. S., & Adil, M. (2025). Trust my IDS: An explainable AI integrated deep learning-based transparent threat detection system for industrial networks. *Computers & Security*, *149*, 104191. https://doi.org/10.1016/j.cose.2024.104191

[27] Stănciulescu, A., Copaci, C. A., & Bacivarov, I. C. (2024, November). Cyber Threats and Exploring the Sources of Cyber Threat Intelligence. In *International Conference on Cybersecurity and Cybercrime* (Vol. 11, pp. 83-89).

[28] Su, N., Huang, S., & Su, C. (2024). Elevating smart manufacturing with a unified predictive maintenance platform: The synergy between data warehousing, apache spark, and machine learning. *Sensors*, *24*(13), 4237. https://doi.org/10.3390/s24134237

[29] Talukder, M. A., Islam, M. M., Uddin, M. A., Hasan, K. F., Sharmin, S., Alyami, S. A., & Moni, M. A. (2024). Machine learning-based network intrusion detection for big and imbalanced data using oversampling, stacking feature embedding and feature extraction. *Journal of big data*, *11*(1), 33.

[30] Terawi, N., Ashqar, H. I., Darwish, O., Alsobeh, A., Zahariev, P., & Tashtoush, Y. (2025). Enhanced detection of intrusion detection system in cloud networks using Time-Aware and deep learning techniques. *Computers*, *14*(7), 282. https://doi.org/10.3390/computers14070282

[31] Waleed, A., Jamali, A. F., & Masood, A. (2022). Which open-source ids? snort, suricata or zeek. *Computer Networks*, *213*, 109116. https://doi.org/10.1016/j.comnet.2022.109116

[32] Zarzoor, A. R., Al-Jamali, N. A. S., & Qader, D. A. A. (2023). Intrusion detection method for internet of things based on the spiking neural network and decision tree method. *International Journal of Electrical and Computer Engineering*, *13*(2), 2278. https://doi.org/10.11591/ijece.v13i2

Authors Biography

Ammar Ahmed Abdullah Director of the Department of Total Quality Management and Institutional Development at the General Directorate of Education in Nineveh. He holds a Bachelor's and Higher Diploma in Computer Science from the University of Mosul, specializing in Real-Time Distributed Database Systems, and a master's degree in data mining in Artificial Intelligence from the Applied Science Private University in Jordan. Currently pursuing a PhD at the University of Mosul, his research focuses on advanced topics in computer science. He has published multiple studies in reputable national, regional, and international journals.

Prof. Dr. Dhuha Basheer Abdullah She is the Dean of the College of Computer Science and Mathematics at the University of Mosul. She also serves as Head of the Committee of Deans of IT Colleges and Chief Editor of the Al-Rafidain Journal of Computer Sciences and Mathematics. She earned her PhD in Computer Science in 2004, specializing in computer architecture and operating systems. She has supervised numerous MSc and PhD theses in areas such as parallel processing, real-time systems, IoT, AIoT, cloud computing, and big data. Dr. Dhuha actively leads and teaches courses at the BSc, MSc, and PhD levels.