

A Big Data Framework for Scalable and Cross-Dataset Capable Machine Learning in Network Intrusion Detection Systems

VINICIUS M. DE OLIVEIRA¹, HENRIQUE M. DE OLIVEIRA¹, GABRIEL M. SANTOS¹, JHONATAN GEREMIAS¹, EDUARDO K. VIEGAS¹

¹ Pontifical Catholic University of Paraná, Curitiba, Paraná, 80215-901, Brazil

Corresponding author: Eduardo K. Viegas (e-mail: eduardo.viegas@ppgia.pucpr.br).

This work was partially sponsored by the Brazilian National Council for Scientific and Technological Development (CNPq), grants n^o 407879/2023-4, 442262/2024-8, and 302937/2023-44.

ABSTRACT Network Intrusion Detection Systems (NIDS) are widely used to secure modern networks, but deploying accurate and scalable Machine Learning (ML)-based detection in high-speed environments remains challenging. Traditional approaches often fail to generalize across different network environments, leading to significant performance degradation in cross-dataset evaluations. Additionally, ensuring near real-time inference while ingesting large volumes of network events requires efficient processing pipelines. In this work, we propose a distributed ensemble-based NIDS designed to improve both accuracy and scalability in large-scale network environments. Our approach leverages a Big Data framework to decouple event ingestion from inference, ensuring high-speed processing without sacrificing detection performance. We implement our system using Apache Spark and Apache Kafka, enabling real-time event ingestion, efficient model inference, and periodic model updates through distributed storage. The ensemble classification scheme enhances generalization capabilities by combining multiple classifiers, reducing accuracy loss in cross-dataset scenarios. Experimental evaluations conducted on three benchmark datasets—UNSW-NB15, CS-CIC-IDS, and BoT-IoT—demonstrate that our proposed approach consistently outperforms traditional techniques. Our model achieves an F-Measure improvement of up to 0.46 in cross-dataset evaluations, addressing the generalization limitations of individual classifiers. Additionally, it achieves near real-time inference throughput comparable to traditional classifiers, processing up to 1.07M events per second with three workers, while our distributed training pipeline scales efficiently, reducing model training time by up to 62% in the same setup.

INDEX TERMS Network intrusion Detection, Machine Learning, Big Data, Generalization.

I. INTRODUCTION

OVER the past few years, network attacks have steadily risen. For example, a security report indicated that 2024 witnessed the most significant attack ever reported, with a Distributed Denial-of-Service (DDoS) exceeding 5.6 Terabits per second [1]. Notably, hyper-volumetric attacks surpassing one billion packets per second increased by 18× in 2024 alone. In response to the escalating volume of attacks, network operators typically rely on Network Intrusion Detection Systems (NIDSs), implemented using either *misuse-based* or *behavior-based* approaches [2]. On the one hand, *misuse-based* strategies rely on a database of well-known malicious signatures, signaling misconduct based on previously iden-

tified attack patterns. As a result, they often fall short of detecting novel or subtle variations of known attacks [3]. On the other hand, *behavior-based* strategies detect malicious activities by analyzing event behavior and identifying attacks based on deviations from an established normal baseline. Therefore, they have the potential to detect new attacks, provided these attacks exhibit behavioral patterns similar to those previously modeled [4].

In general, *behavior-based* NIDSs is implemented using pattern recognition strategies, often leveraging Machine Learning (ML) techniques [5]. To this end, a behavioral ML model is trained on a dataset containing millions of network samples, encompassing both normal and malicious activi-

ties [6]. The accuracy of the resulting model is then evaluated using a testing dataset, and its measured performance is expected to be confirmed when the system is deployed in production. Consequently, current literature assumes that the behavior in the production environment closely mirrors that observed during the testing phase [7]. However, network traffic behavior is highly variable, as evidenced by the introduction of new services or the emergence of new attacks. This situation often results in highly accurate ML-based NIDSs that struggle to maintain sufficient accuracy when deployed in environments different from those encountered during the training phase [8].

The design of a generalization-capable ML-based NIDS for high-speed networks, specifically targeting current hypervolumetric attacks, is often overlooked in the literature [9]. Current approaches typically address the detection of a broader range of attack and normal samples by increasing the complexity of the underlying classifier, often by using Deep Neural Networks (DNNs) [10]. As a result, while these systems have the potential to improve generalization, they are often impractical for high-speed networks due to the high computational costs associated with the inference phase. In high-speed networks, the inference task must be performed at scale with minimal computational costs; in contrast, current approaches often require substantial memory footprints and impose impractical computational demands [11].

Achieving a generalizable ML-based NIDS for high-speed networks necessitates its implementation as a distributed system capable of operating at scale [12]. This process involves several challenges that must be addressed to ensure the effective deployment of an ML-based NIDS in high-speed networks. First, storing and provisioning ML models for inference requires efficient versioning mechanisms and low-latency access to accommodate frequent updates and model retraining [13]. Without an optimized storage and retrieval strategy, the system may experience significant delays in threat detection. Second, designing a distributed event ingestion mechanism for near real-time processing demands high-throughput data pipelines capable of handling massive volumes of network traffic while minimizing processing overhead [14]. This ensures that network anomalies and potential threats are detected with minimal latency. Finally, enabling architectural scalability typically requires the system design as a microservice-based implementation, allowing the system to distribute workloads dynamically across multiple nodes [15]. This approach enhances fault tolerance, facilitates load balancing, and ensures that the system can adapt to fluctuations in network traffic without compromising detection accuracy or performance.

Unfortunately, existing literature on ML-based NIDSs often overlooks the challenge of generalization, assuming that models trained on specific datasets will perform reliably across diverse network environments [16]. When generalization is considered, studies typically focus on improving model robustness through more complex architectures, such as DNN, while neglecting the feasibility of deploying these

models at scale [17]. Conversely, approaches that address scalability often target a single aspect of the system, such as optimizing inference efficiency, while disregarding the broader integration challenges. In particular, many scalable implementations fail to account for essential components such as model storage, real-time inference, and distributed event ingestion, treating them as isolated tasks rather than interconnected systems. As a result, current solutions lack the necessary infrastructure to operate efficiently in high-speed network environments for NIDS.

Contribution. In light of this, this paper proposes a novel scalable Big Data architecture for cross-dataset capable ML-based NIDS, implemented in two key stages. First, we design the classification task as an ensemble of shallow classifiers, leveraging a majority voting mechanism. Our approach selects the most effective classifiers based on cross-dataset performance, ensuring improved generalization while maintaining lower computational costs. Second, we implement the system as a distributed microservice-based architecture atop a Big Data platform. The proposed architecture integrates model storage and serving, distributed event ingestion, and scalable inference to handle high-speed network traffic efficiently. As a result, our system enhances classification generalization and ensures scalability, making it suitable for real-world deployment in high-throughput environments.

In summary, the main contributions of this paper are:

- We comprehensively evaluate the generalization capabilities of widely used ML-based NIDSs in the literature. Our experiments demonstrate that existing approaches fail to effectively generalize learned behaviors from the training environment to a different dataset, resulting in a significant decline in accuracy;
- A new generalization-capable architecture implemented atop a Big Data platform. Our proposed model enhances cross-dataset generalization, increasing the F-Measure by up to 0.46. Additionally, it achieves near real-time inference throughput comparable to traditional classifiers, processing up to 1.07M events per second with three workers, while our distributed training pipeline scales efficiently, reducing model training time by up to 62% in the same setup;

Roadmap. The remainder of this paper is organized as follows. Section II further describes the operation of ML-based NIDSs and Big Data platforms. Section III overviews the current literature. Section IV describes our proposed model, Section V introduces its implementation, and Section VI evaluates its performance. Finally, Section VII concludes our work.

II. PRELIMINARIES

This section further overviews the typical implementation of ML-based NIDSs. In addition, we present the operation of current Big Data architectures in terms of their scalability.

TABLE 1: Feature descriptions for network flow analysis. Features are extracted from each client-server communication comprising a 60-second interval.

Feature	Description
PROTOCOL	IP protocol identifier
L7_PROTO	Layer 7 protocol identifier (numeric)
IN_BYTES	Total number of incoming bytes
OUT_BYTES	Total number of outgoing bytes
IN_PKTS	Total number of incoming packets
OUT_PKTS	Total number of outgoing packets
FLOW_DURATION_MILLISECONDS	Total duration of the flow in milliseconds
TCP_FLAGS	Cumulative value of all TCP flags
CLIENT_TCP_FLAGS	Cumulative value of TCP flags set by the client
SERVER_TCP_FLAGS	Cumulative value of TCP flags set by the server
DURATION_IN	Duration of the client-to-server stream (ms)
DURATION_OUT	Duration of the server-to-client stream (ms)
LONGEST_FLOW_PKT	Size of the largest packet in the flow (bytes)
SHORTEST_FLOW_PKT	Size of the smallest packet in the flow (bytes)
MIN_IP_PKT_LEN	Length of the smallest IP packet in the flow
MAX_IP_PKT_LEN	Length of the largest IP packet in the flow
SRC_TO_DST_SECOND_BYTES	Data transfer rate from source to destination (bytes/sec)
DST_TO_SRC_SECOND_BYTES	Data transfer rate from destination to source (bytes/sec)
RETRANSMITTED_IN_BYTES	Total retransmitted bytes from source to destination
RETRANSMITTED_IN_PKTS	Total retransmitted packets from source to destination
RETRANSMITTED_OUT_BYTES	Total retransmitted bytes from destination to source
RETRANSMITTED_OUT_PKTS	Total retransmitted packets from destination to source
SRC_TO_DST_AVG_THROUGHPUT	Average throughput from source to destination (bps)
DST_TO_SRC_AVG_THROUGHPUT	Average throughput from destination to source (bps)
NUM_PKTS_UP_TO_128_BYTES	Number of packets with an IP size of at most 128 bytes
NUM_PKTS_128_TO_256_BYTES	Number of packets with an IP size between 129 and 256 bytes
NUM_PKTS_256_TO_512_BYTES	Number of packets with an IP size between 257 and 512 bytes
NUM_PKTS_512_TO_1024_BYTES	Number of packets with an IP size between 513 and 1024 bytes
NUM_PKTS_1024_TO_1514_BYTES	Number of packets with an IP size between 1025 and 1514 bytes
TCP_WIN_MAX_IN	Maximum TCP window size from source to destination
TCP_WIN_MAX_OUT	Maximum TCP window size from destination to source
ICMP_TYPE	ICMP type and code (ICMP type * 256 + ICMP code)
ICMP_IPV4_TYPE	ICMP type
DNS_QUERY_ID	Unique identifier for the DNS query transaction
DNS_QUERY_TYPE	DNS query type (e.g., 1 = A, 2 = NS, etc.)
FTP_COMMAND_RET_CODE	Return code of the FTP client command
SRC_TO_DST_IAT_MIN	Minimum inter-packet arrival time from source to destination (ms)
SRC_TO_DST_IAT_MAX	Maximum inter-packet arrival time from source to destination (ms)
SRC_TO_DST_IAT_AVG	Average inter-packet arrival time from source to destination (ms)
SRC_TO_DST_IAT_STDDEV	Standard deviation of inter-packet arrival time from source to destination
DST_TO_SRC_IAT_MIN	Minimum inter-packet arrival time from destination to source (ms)
DST_TO_SRC_IAT_MAX	Maximum inter-packet arrival time from destination to source (ms)
DST_TO_SRC_IAT_AVG	Average inter-packet arrival time from destination to source (ms)
DST_TO_SRC_IAT_STDDEV	Standard deviation of inter-packet arrival time from destination to source

A. MACHINE LEARNING FOR NIDS

A typical ML-based NIDS is implemented in four stages, comprising *Data Acquisition*, *Feature Extraction*, *Classification*, and *Alert* [18]. In the *Data Acquisition* stage, raw network traffic is continuously captured from various sources, such as packet sniffers from a Network Interface Card (NIC) or network flow collectors. The *Feature Extraction* stage processes this raw data to derive meaningful characteristics that represent network behavior, typically using statistical or time-series representations [19]. Table 1 shows the set of features that can be extracted from the network traffic. The *Classification* stage applies an ML model to categorize network activity as either benign or malicious based on extracted features, leveraging techniques ranging from shallow classifiers to deep learning models. Finally, the *Alert* stage generates notifications or triggers automated responses when malicious activity is detected, enabling real-time mitigation and threat response within the network.

Typically, the classification stage in ML-based NIDSs relies on a model built using a three-phase process: *training*, *validation*, and *testing* [20]. In the *training* phase, the model learns patterns from a labeled dataset by adjusting its parameters to distinguish between normal and malicious network activity. The *validation* phase is then used to fine-tune the model by evaluating its performance on a separate dataset, helping to optimize hyperparameters and prevent overfitting. Finally, in the *testing* phase, the model's generalization capabilities are assessed using an unseen dataset, estimating how well it will perform in real-world scenarios.

This traditional approach presents a significant challenge for the application of NIDS, as network behavior is highly dynamic, changing due to new services or the emergence of novel attack patterns [21]. As a result, proposed solutions must be capable of generalizing the behavioral characteristics learned during training to different operational environments. To address this issue, some authors have explored cross-

dataset training, where models are trained and evaluated using multiple datasets to enhance their adaptability to diverse network conditions [16]. However, the predominant strategy in such approaches relies on deep learning classifiers, which, while effective in capturing complex patterns, impose substantial memory requirements and computational costs [22]. This constraint makes their deployment in high-speed networks particularly challenging, as real-time inference must be conducted at scale with minimal resource consumption.

B. BIG DATA PROCESSING

Big Data is commonly characterized by three fundamental properties, known as the three Vs: *Volume*, *Velocity*, and *Variety* [23]. *Volume* refers to the vast amount of data generated continuously, requiring scalable storage and processing solutions. *Velocity* represents the high speed at which data is produced and must be processed in real-time or near real-time to extract timely insights. *Variety* encompasses the diverse formats and structures of data, ranging from structured logs to unstructured network packets. As an example, in the context of NIDS, these characteristics are particularly relevant [24]. Network traffic monitoring generates massive *Volume* as data flows through high-speed networks, requiring efficient storage and processing architectures. The *Velocity* aspect is critical since intrusion detection must be performed in near real-time to promptly identify and mitigate potential threats. Lastly, the *Variety* of network data, including raw packets, flow-based records, and protocol-specific logs, demands adaptable feature extraction and classification techniques to ensure accurate detection across different network environments.

These characteristics necessitate the development of new architectures capable of effectively managing the complexities of Big Data characteristics in NIDS domain [25]. Traditional architectures often struggle to scale efficiently, making it imperative to design solutions that can accommodate the ever-growing *volume*, *velocity*, and *variety* of network traffic data. One key challenge lies in *storage*, as the vast amount of data generated by high-speed networks demands distributed and scalable storage systems that balance efficiency and retrieval speed. *Event ingestion* poses another challenge, as network data must be collected, processed, and forwarded in near real-time to ensure timely threat detection while minimizing bottlenecks. Finally, *processing* is a critical hurdle, as ML-based NIDS require adequate computational resources to analyze massive datasets and execute inference at scale. Addressing these challenges requires architectures that integrate distributed computing, efficient data pipelines, and optimized ML models to ensure both scalability and real-time performance.

Traditional storage solutions often rely on the Hadoop Distributed File System (HDFS), a scalable and fault-tolerant storage system designed to handle large volumes of data efficiently [26]. HDFS follows a master-slave architecture, where a central *NameNode* manages metadata and directory structures, while multiple *DataNodes* store the actual data

blocks across a distributed cluster. To ensure fault tolerance, it replicates data across multiple nodes, typically using a default replication factor of three. This design allows for high availability and resilience against hardware failures.

For data processing, designed architectures usually resort to Apache Spark, a widely used framework for fast and scalable solutions for handling large-scale datasets [27]. Unlike traditional batch-processing frameworks, Spark provides in-memory computation, enabling significantly faster data processing for ML, stream processing, and iterative workloads. Similarly, its architecture follows a master-worker model, where a central *Driver* program manages the execution and distributes tasks across multiple *Worker* nodes. Each worker runs one or more *Executors*, responsible for executing tasks in parallel and storing intermediate data in memory when possible. Spark employs a Directed Acyclic Graph (DAG) scheduler to optimize task execution and minimize data shuffling. Additionally, it supports multiple programming interfaces, including Spark SQL for structured data, MLlib for machine learning, and Spark Streaming for real-time processing, making it a versatile choice for Big Data applications such as NIDS.

In this setting, Apache Kafka is a widely adopted event ingestion mechanism for Big Data environments, designed to handle high-throughput, low-latency data streams in a distributed and fault-tolerant manner [28]. Kafka follows a publish-subscribe architecture where data producers send events to *topics*, which subscribers then consume. Each topic is divided into multiple *partitions*, enabling parallel processing and scalability. Kafka brokers manage message storage and distribution, ensuring durability by persisting data to disk and replicating it across multiple nodes. A *Zookeeper* service coordinates broker metadata, manages leader election for partitions, and maintains cluster state. Consumers can process events in real-time or batch mode, making Kafka an essential component for streaming analytics, log aggregation, and intrusion detection systems. In the context of NIDS, Kafka efficiently ingests network events at scale, ensuring reliable and sequential data processing for real-time threat analysis.

As a result, the integration of HDFS, Apache Spark, and Apache Kafka have the potential to enable the development of a Big Data processing architecture for NIDS, leveraging distributed storage, real-time event ingestion, and scalable processing [23]. In this setup, Kafka ingests high-volume network traffic data, distributing it across partitions for efficient consumption. Spark processes this data in real-time or batch mode, applying ML models to detect intrusions while leveraging HDFS as a persistent storage layer for historical data and model training. However, this integration poses several challenges, including ensuring low-latency processing while handling massive data streams, efficiently managing resource allocation across Spark and Kafka clusters, and maintaining data consistency between real-time analysis and stored historical datasets. Additionally, achieving fault tolerance and scalability requires careful coordination of data replication,

load balancing, and system monitoring to prevent bottlenecks in high-speed network environments.

III. RELATED WORKS

Over the past years, several works have proposed highly accurate ML-based NIDSs. In general, proposed schemes aim for higher detection accuracies while overlooking the generalization and processing costs of their scheme. As an example, Z. Ye *et al.* [29] proposes an ensemble of ML classifiers built through a feature selection strategy. Their proposed model increases accuracy on a single dataset while neglecting the resulting generalization capabilities and processing costs. Similarly, C. Hazman *et al.* [30] proposes a AdaBoost framework built through a feature selection strategy. Their scheme can improve accuracy while also reducing inference computational costs, however, the impact on model generalization is not addressed. M. Mohy-Eddine *et al.* [31] makes use of an ensemble of classifiers built through a feature reduction strategy. Their proposed approach improves accuracy on a single dataset while disregarding the challenges associated with model generalization. M. Ali Khan *et al.* [32] build an ensemble of classifiers through an AutoML strategy. The proposed model improves accuracy when compared to traditional approaches but neglect processing and generalization costs. A. M. Alsaffar *et al.* [33] combine a wrapper and filter feature selection strategy to build an ensemble of stacked classifiers. Their approach improves classification performance but neglects inference computational costs and model generalization capabilities.

Model generalization in NIDS is rarely considered in the literature, where authors usually assume that the accuracy measured on the *testing* dataset will reflect in real-world settings [4]. M. Cantone *et al.* [16] assessed the accuracy of widely used ML classifiers for NIDS in a cross-dataset setting. Their evaluation showed that current schemes significantly degrade their accuracy when evaluated in a different dataset than that used during training phase. To address this issue, M. Wang *et al.* [34] combines the datasets in a cross-dataset sampling strategy for DNN training task. Their approach improves accuracy when a cross-dataset setting is considered, however, the processing costs associated with the DNN is neglected. S. Wali *et al.* [35] attempt to address generalization by extracting a multimodal dataset integrating flow, payload and contextual features. Their extracted features improve accuracy in a cross-dataset validation, however, it overlooks the computational costs associated with the feature extraction costs. Z. Fan *et al.* [36] attempt to address generalization by relying on a cross-validation approach in a single dataset. Their scheme improves accuracy with parameter tuning but neglects the cross-dataset setting. N. Niknami *et al.* [37] relies on a Few-Shot Learning strategy through DNN to improve detection accuracy. Their model, when assessed in a cross-dataset setting improves accuracy when compared to traditional approaches, however, their scheme is not designed to achieve such a goal. G. Duan *et al.* [38] aim to increase model generalization by assessing massive datasets in a cross-

TABLE 2: A summary of related work and the characteristics of their NIDS implementations.

Work	Big Data Architecture	Model Update	Event Ingestion	Model Generalization	High-speed Network	Improve Accuracy
Z. Ye <i>et al.</i> [29]	×	×	×	×	×	✓
C. Hazman <i>et al.</i> [30]	×	×	×	×	×	✓
M. Mohy-Eddine <i>et al.</i> [31]	×	×	×	×	×	✓
M. Ali Khan <i>et al.</i> [32]	×	✓	×	×	×	✓
A. M. Alsaffar <i>et al.</i> [33]	×	×	×	×	×	✓
M. Cantone <i>et al.</i> [16]	×	×	×	✓	×	✓
M. Wang <i>et al.</i> [34]	×	×	×	✓	×	✓
S. Wali <i>et al.</i> [35]	×	×	×	✓	×	✓
Z. Fan <i>et al.</i> [36]	×	×	×	✓	×	✓
N. Niknami <i>et al.</i> [37]	×	×	×	✓	×	✓
G. Duan <i>et al.</i> [38]	×	×	×	✓	×	✓
A. Abid <i>et al.</i> [39]	✓	×	×	×	×	✓
F. Jemili <i>et al.</i> [40]	✓	✓	×	×	✓	✓
B. L. Pandey <i>et al.</i> [41]	✓	×	✓	×	✓	×
F. Ullah <i>et al.</i> [17]	✓	×	×	×	✓	✓
A. Kourid <i>et al.</i> [42]	✓	×	×	×	✓	✓
Ours	✓	✓	✓	✓	✓	✓

dataset setting. Their approach based on DNN improves accuracy but neglects the inference computational costs.

Addressing the high-speed network traffic for ML-based NIDS is also not easily achieved in the literature. A. Abid *et al.* [39] leverages Cloud Computing and Big Data to conduct data fusion for intrusion detection at high-speed networks. Their approach improves accuracy but overlooks model generalization. F. Jemili *et al.* [40] proposes a ensemble of ML classifiers in a Big Data environment. Their approach improves accuracy and address high-speed networks, however they overlook how model generalization can be addressed. B. L. Pandey *et al.* [41] implements an intrusion detection scheme in a Big Data setting with feature selection and data augmentation. Their approach improves accuracy but neglects the computational costs and model generalization aspects. F. Ullah *et al.* [17] implements a Big Data oriented architecture for NIDS with Apache Spark and DNN. The proposed architecture provides significantly high detection accuracies but overlooks model generalization. A. Kourid *et al.* [42] implement their NIDS on top of Apache Spark to address the high variability of network traffic behavior. Their approach provides low error rates but also neglects model generalization aspects.

A. DISCUSSION

Table 2 overviews the current literature on ML-based NIDS for high-speed networks. The current literature predominantly focuses on improving classification accuracy, often employing complex DNN models to enhance detection performance. However, a significant limitation of these approaches is their inability to generalize effectively across different network environments. Many models are trained and

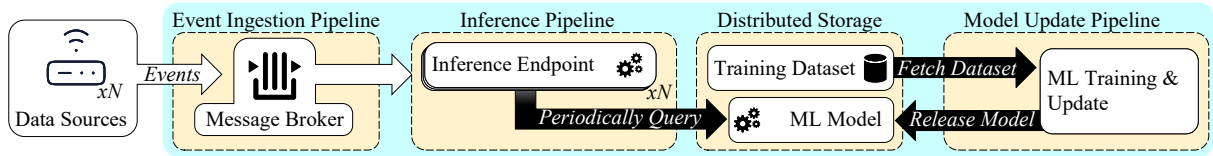


FIGURE 1: Overview of our proposed architecture for enabling ML-based NIDS in high-speed networks. *Event Ingestion Pipeline* receives at near real-time events from multiple data sources for inference. *Inference Pipeline* conducts the inference task at scale with multiple endpoints. *Distributed Storage* stores the training dataset and the ML model in a distributed manner. The *Model Update Pipeline* conducts the ML model training and periodic updates when requested.

tested within a single dataset, leading to high accuracy within controlled conditions but poor performance when deployed in real-world settings with different network behaviors. While some studies attempt to address model generalization by incorporating cross-dataset evaluation, these approaches typically introduce additional computational complexity, making them impractical for deployment in high-speed networks. The reliance on resource-intensive models further increase this issue, as real-time intrusion detection demands efficient processing capabilities to handle large-scale traffic without introducing latency.

Furthermore, existing solutions often neglect key aspects of scalable implementation, particularly the integration of Big Data architectures, event ingestion mechanisms, and efficient model updates. High-speed networks generate vast amounts of traffic that require real-time analysis, yet most studies do not incorporate distributed event ingestion frameworks such as Apache Kafka, limiting their ability to process large-scale data streams effectively. Similarly, the lack of Big Data infrastructure, such as Apache Spark and HDFS, hinders scalability and long-term storage for model retraining and adaptation. As a result, even when model updates are considered, they are typically performed offline, requiring significant manual intervention. This fragmented approach prevents existing ML-based NIDSs from achieving both high accuracy and efficient real-time deployment, highlighting the need for a more comprehensive architecture that integrates scalability, low-latency processing, and automated model adaptation.

IV. A BIG DATA ARCHITECTURE FOR CROSS-DATASET CAPABLE ML-BASED NIDS

Our proposed architecture is implemented as a Big Data processing framework to address the aforementioned challenges of near real-time ML-based NIDS in high-speed networks. In practice, it aims to address three key challenges associated with high-speed networks:

- **Model Generalization** We design an ensemble of shallow classifiers optimized for cross-dataset performance. Instead of relying on a single model that may overfit a specific dataset, we construct an ensemble that selects the best-performing classifiers across different datasets. This enhances the system's ability to detect novel threats while maintaining computational efficiency, ensuring high detection accuracy across varying network environments;

- **Near Real-time Inference at Scale** Our architecture enables the processing of high-speed network traffic with minimal latency while maintaining high throughput. By distributing the workload across multiple computing nodes, we ensure efficient inference that is capable of handling hyper-volumetric attacks in near real-time;
- **Distributed Model Training and Update** We incorporate a scalable model storage and provisioning mechanism that enables continuous retraining with updated network traffic. This ensures the system adapts to evolving attack patterns without requiring manual intervention, improving long-term reliability and maintaining detection accuracy over time;

Figure 1 illustrates the implementation of our proposed architecture. It includes the *Event Ingestion Pipeline*, *Inference Pipeline*, *Distributed Storage*, and *Model Update Pipeline*. The *Event Ingestion Pipeline* leverages a message broker to efficiently collect and process network events from multiple data sources in near real-time. It is designed for scalability and ensures that high-speed network traffic is continuously ingested and made available for analysis without delays. The *Inference Pipeline* is implemented as a distributed service in a Big Data framework that processes incoming events by applying an ensemble of shallow classifiers through a majority voting procedure. This ensemble is built by selecting classifiers that demonstrate strong performance in a cross-dataset evaluation, ensuring improved generalization across different network environments. By relying on multiple lightweight models instead of a single complex counterpart, this approach maintains computational efficiency, making it suitable for high-speed networks while reducing the risk of model overfitting. The *Distributed Storage* component is responsible for storing both the training datasets and also different versions of the ML models. Maintaining historical datasets and model snapshots enables efficient retrieval and management of data for continuous model training and evaluation. Finally, the *Model Update Pipeline* fetches training datasets from the distributed storage to periodically retrain the ML model. When a new model version is available, it is released and deployed to the inference pipeline, ensuring the system remains updated with evolving network behaviors and emerging attack patterns.

The following subsections further describe our proposed architecture, including the modules that implement it.

A. CLASSIFICATION AND MODEL TRAINING

Achieving effective ML-based NIDS classification that ensures generalization while operating in high-speed networks and near real-time presents significant challenges. Network traffic is inherently dynamic, with new services and attack patterns continuously emerging, requiring models that can generalize across diverse environments. However, ensuring this generalization often demands complex ML models, which can be computationally expensive and unsuitable for real-time processing in high-speed networks. Additionally, integrating scalable event ingestion and distributed inference mechanisms is crucial to handling large volumes of traffic efficiently while maintaining low-latency detection.

Our proposed model addresses this challenge by leveraging an ensemble of shallow classifiers, ensuring both efficiency and generalization in high-speed networks. The ensemble is constructed by selecting the best-performing models in a cross-dataset manner, allowing it to adapt to varying network behaviors while maintaining low computational costs. This approach enhances detection accuracy across different environments while enabling near real-time inference, making it suitable for deployment in large-scale, high-speed network scenarios.

Let x be a to-be-classified event feature vector comprising N features, and $\mathcal{E}$ an M -sized ensemble of h shallow classifiers, where each classifier implements a function as $h_m \rightarrow \{0, 1\}$, with $m \in \{1, \dots, M\}$, hence, mapping the feature space to a binary decision space, indicating whether the input corresponds to an attack 1 or normal traffic 0. The final classification decision $H(x)$ is obtained through a majority voting scheme, formally defined through the following equation:

$$H(x) = \mathbb{I}\left(\left(\sum_{m=1}^M h_m(x)\right) > \frac{M}{2}\right) \quad (1)$$

where $\mathbb{I}(\cdot)$ is the indicator function that returns 1 if the condition holds and 0 otherwise. The ensemble $\mathcal{E}$ consists of M shallow classifiers, each denoted as $h_m(x)$, where $m \in \{1, 2, \dots, M\}$, and each classifier produces a binary output: $h_m(x) = 1$ if the instance is classified as an attack and $h_m(x) = 0$ otherwise. Therefore, the final ensemble $\mathcal{E}$ classification decision is obtained through a majority voting rule, where the instance is classified as an attack if more than half of the classifiers predict it as such. As a result, our classification scheme can be implemented in a lightweight manner to address near real-time high-speed network traffic classification.

To construct the ensemble $\mathcal{E}$ and ensure generalization capabilities, we first train multiple classifiers on multiple datasets, ensuring diverse learning representations from different network environments. Each classifier is independently trained and evaluated on various datasets to assess its generalization performance. We then select the subset of classifiers that achieve the highest accuracy in a cross-dataset evaluation. This selection process ensures that the ensemble is composed

of models capable of detecting intrusions across different network conditions, improving overall detection reliability while maintaining computational efficiency for high-speed network environments.

Let $\mathcal{D} = \{D_1, \dots, D_N\}$ be a set of training datasets with N datasets, where each D_n contains labeled instances for both normal and malicious network traffic. For each dataset D_n , we train a set of classifiers $\{h_{n1}(x), \dots, h_{nK}(x)\}$, where each $h_{nk} : \mathbb{R}^N \rightarrow \{0, 1\}$ is a classifier that outputs either 0 (normal) or 1 (malicious) for a given feature vector x , and K represents the number of classifiers trained for each dataset. After training, each classifier is evaluated on the same dataset D_n used for training. The classifier that provides the best accuracy on the testing dataset of D_n is selected to form the ensemble. This process is repeated for all datasets in $\mathcal{D}$, and the ensemble $\mathcal{E}$ is constructed by selecting the best-performing classifier from each training dataset, ensuring that the ensemble incorporates classifiers that perform optimally for their respective datasets. The resulting ensemble then classifies network events, improving generalization and adaptability across different network environments.

B. INFERENCE PIPELINE

Implementing inference in high-speed networks presents a significant challenge because it requires not only performing classification on incoming network traffic but also ingesting events at high speed. In such environments, the volume and velocity of network data make it essential to process and classify events in near real-time without introducing delays. This requirement of high-speed event ingestion and fast inference poses difficulties, as the system must be capable of handling large amounts of data continuously while applying ML models for accurate and timely classification. Balancing both tasks efficiently in high-speed networks demands scalable architectures that can handle the throughput without compromising inference performance.

We consider an architecture that ingests network events for inference from multiple sources (*Data Sources*, Fig. 1). The events are collected by a *Message Broker*, which is implemented in a distributed manner to handle large volumes of data in near real-time. The collected events are then provided as input to the *inference Endpoint*, which applies the ensemble of classifiers (see Section IV-A) in a distributed manner to conduct this task at scale. Additionally, the inference endpoint periodically queries a distributed storage system for the latest ML model version, ensuring that the most up-to-date model is always used for classification.

We decouple the inference and event ingestion processes to ensure that each task can be scaled and optimized independently, avoiding bottlenecks and enhancing system efficiency. Separating the two tasks allows for more flexible resource allocation, enabling the ingestion pipeline to focus on handling incoming events at high speed without being hindered by the computational demands of the inference process. This design also facilitates load balancing and the ability to scale either process based on the specific needs of the network en-

environment, ensuring smooth operation even under high traffic conditions. Additionally, we utilize a *Distributed Storage* to store the ML model, which further enhances the efficiency of our proposed inference pipeline by allowing for fast and scalable retrieval of the latest model version for inference.

C. MODEL UPDATE PIPELINE

We implement the model training pipeline in a distributed manner to efficiently handle massive amounts of training data. The training dataset, stored in a distributed storage system, is fetched and processed in parallel across multiple nodes to accelerate the training process. The model training is conducted distributedly, utilizing the computational power of the system to handle large-scale datasets. Additionally, we can conduct periodic model updates to ensure that the classifier adapts to evolving network behaviors and new attack patterns. When a new model is trained, it is released and stored in the distributed storage system, making it available for querying by the inference endpoint and ensuring that the most up-to-date version of the model is always used for classification tasks. This approach enables continuous improvement and scalability of the system without compromising performance.

D. DISCUSSION

Our model offers several key insights that address the challenges of traditional ML-based NIDSs in high-speed networks. First, it enables model generalization by selecting classifiers based on their performance across multiple datasets, ensuring that the system can effectively handle diverse network environments and adapt to new attack patterns. Second, it supports near real-time inference at scale, leveraging a distributed architecture to quickly process and classify network events from multiple sources, even in high-speed networks. This is achieved by decoupling event ingestion from inference, allowing each task to be handled independently at scale. Finally, our model incorporates distributed model training and updates, enabling the continuous adaptation of the system by periodically updating the model with the latest training data. This ensures the system stays current with evolving network behavior, allowing for sustained detection accuracy and robustness in dynamic environments.

V. PROTOTYPE

We implemented a proposal prototype in a distributed environment as illustrated in Figure 2. It considers the implementation of a distributed Big Data processing architecture executing our proposed scheme (see Section IV). To achieve this, the hardware and software components of the prototype are designed to create a commonly available Big Data processing architecture that ingests network events at scale for inference purposes. Each architecture node is executed through an Ubuntu v.24.04 equipped with an Intel i7 with 4 CPU cores and 16GB of memory. The infrastructure elements are deployed as an isolated container through Docker v24.0.

The *Distributed Storage* (Fig. 1) is implemented through a HDFS cluster composed of three DataNodes and a single

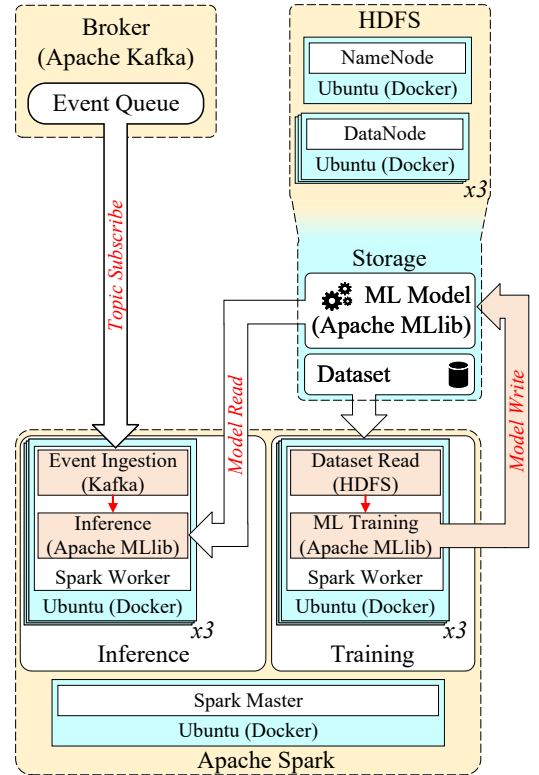


FIGURE 2: Prototype overview of our proposed model. *Distributed Storage* is implemented through HDFS, *Message Broker* makes use of Apache Kafka, and *Inference Pipeline* and *Model Update Pipeline* is implemented through Apache Spark.

NameNode. The deployed DataNodes stores the *Training Dataset* when required for model training and update purposes and the ML model that will be used for model inference. We implement the *Event Ingestion Pipeline* through an Apache Kafka message broker v.3.7.0. The events used for model inference are published as a Kafka topic and later read by the inference pipeline in near real-time.

The *Inference Pipeline* is executed on top of Apache Spark v.3.5.4. The pipeline is executed as an Apache Spark job with up to 3 Apache Workers simultaneously, continuously ingesting events for inference through the Apache Kafka broker (*Topic Subscribe*, Fig. 2) The ingested events are used for inference by applying the previously trained ML model implemented with the Apache Spark Machine Learning Library (MLlib). To achieve such a goal, at the job deployment phase, the model available at the *Distributed Storage* module is read and used for model inference (*Model Read*, Fig. 2).

The *Model Update Pipeline* is also implemented on top of Apache Spark as a job with up to 3 workers. At model updates, it reads a previously stored dataset from the HDFS and builds a new ML model through Apache Spark MLlib API. The resulting model is then stored back on HDFS (*Model Write*, Fig. 2).

Our prototype enables the implementation of our scheme in a fully distributed manner by leveraging a Big Data frame-

work that ensures scalability, fault tolerance, and efficient processing of network events. The architecture integrates Apache Kafka for high-speed event ingestion, Apache Spark for distributed inference and model updates, and HDFS for scalable storage, creating a seamless pipeline for handling large-scale network traffic. The prototype ensures near real-time inference by distributing computation across multiple worker nodes, efficiently processing high-throughput network events while maintaining high availability. Furthermore, the distributed model training and update mechanisms allow the system to periodically refine the ML model without disrupting ongoing inference tasks, ensuring adaptability to evolving network threats. This architecture enables efficient parallel processing and enhances the robustness and generalization capabilities of our proposed scheme, making it suitable for real-world, high-speed network environments.

VI. EVALUATION

Our conducted experiments aim to answer the following Research Questions (RQs):

- **RQ1:** What are the generalization capabilities of traditional ML-based NIDSs?
- **RQ2:** Does our proposed classification scheme improves generalization?
- **RQ3:** What are the scaling capabilities of our scheme?

The next subsections describe the performance of our scheme, including the model-building aspects.

A. MODEL BUILDING

We assessed our proposed scheme on top of our previously described prototype (see Section V). To achieve such a goal, we evaluated four classifiers, namely Decision Tree (DT), Gradient Boosting (GBT), Multilayer Perceptron (MLP), and Random Forest (RF). The DT classifier utilizes the Gini impurity criterion for node splitting. The GBT classifier is configured with a learning rate of 0.1, employs the deviance loss function, and consists of an ensemble of 10 decision trees as its base learners. The MLP classifier is trained for 100 iterations with a batch size of 128 and consists of four hidden layers, each containing 128 neurons, using the logistic sigmoid activation function. The RF classifier is constructed with an ensemble of 10 decision trees as its base learners, with predictions aggregated through majority voting. The selected classifiers were implemented using Apache Spark MLlib to enable efficient distributed training and inference. By leveraging Apache Spark MLlib, we ensure that these classifiers are efficiently trained and applied in a distributed manner, enhancing scalability for high-speed network environments.

We evaluated the performance of the selected algorithms using three benchmark datasets: UNSW-NB15, CS-CIC-IDS, and BoT-IoT. These datasets provide diverse network traffic characteristics, enabling a comprehensive assessment of our model's generalization capabilities across different environments as follows:

- **UNSW-NB15** [43]. This dataset contains a mix of normal and malicious network traffic generated using a

cyber range testbed. It includes a variety of modern attack types, such as DoS, backdoors, and exploits;

- **CS-CIC-IDS** [44]. Developed by the Canadian Institute for Cybersecurity, this dataset captures realistic attack scenarios and normal traffic patterns. It features up-to-date cyber threats, including botnets, brute force attacks, and web-based intrusions;
- **BoT-IoT** [45]. Specifically designed to represent IoT-based attacks, this dataset includes extensive IoT traffic alongside various attack types, such as DDoS, data exfiltration, and reconnaissance;

We utilized three datasets to enable a comprehensive cross-validation evaluation of our approach. By leveraging multiple datasets, we assess the generalization capabilities of our model across different network environments and attack scenarios. This cross-validation strategy ensures that our ensemble selection process identifies classifiers that perform consistently well in diverse settings, enhancing the robustness and reliability of our scheme (see Section IV-A). Each dataset is randomly divided without replacement into *training*, *validation*, and *testing* datasets, each composed of 40%, 30%, and 30% of samples, respectively. The *training* dataset is used for model training purposes. The *validation* dataset is used for model fine-tuning. Finally, the *testing* dataset is used to measure the resulting model accuracy. The reported accuracy performance in the paper is measured on the *testing* dataset. The behavior of the selected datasets is represented by their flow-based features as shown in Table 1.

We evaluate the selected classifiers using the following classification performance metrics:

- **True Positive (TP)**: number of attack samples correctly classified as an attack.
- **True Negative (TN)**: number of normal samples correctly classified as normal.
- **False Positive (FP)**: number of normal samples incorrectly classified as an attack.
- **False Negative (FN)**: number of attack samples incorrectly classified as normal.

Further, we measure the F-Measure according to the harmonic mean of precision and recall values while considering attack samples as positive and normal samples as negative, as shown in Eq. 4.

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

$$F-Measure = 2 \times \frac{Precision \cdot Recall}{Precision + Recall} \quad (4)$$

B. THE GENERALIZATION CHALLENGE

Our first experiment aims at answering RQ1 and investigates the generalization capabilities of traditional ML-based NIDSs. To achieve such a goal, we evaluate the classification performance of the selected classifiers on multiple datasets

TABLE 3: Classification performance of the selected classifiers in a cross-dataset setting as measured by the obtained F-Measure on the *testing* dataset.

Training Env.	Class.	Testing Environment		
		UNSW-NB15	CS-CIC-IDS	BoT-IoT
UNSW-NB15	DT	0.97	0.02	0.68
	GBT	0.96	0.04	0.65
	MLP	0.92	0.66	0.10
	RF	0.96	0.10	0.16
CS-CIC-IDS	DT	0.32	0.98	0.08
	GBT	0.03	0.97	0.65
	MLP	0.03	0.92	0.06
	RF	0.02	0.97	0.08
BoT-IoT	DT	0.07	0.23	1.00
	GBT	0.03	0.30	1.00
	MLP	0.08	0.21	0.99
	RF	0.25	0.54	1.00
Ours		0.77	0.56	1.00

(see Section VI-A). Specifically, we assess whether the accuracy of the selected techniques can be maintained when applied to a different dataset in a cross-dataset evaluation setting. For example, we build a classifier using the UNSW-NB15 dataset and evaluate the resulting model accuracy on the BoT-IoT dataset. This approach allows us to determine how well a model trained on one network environment performs when exposed to a different dataset. It reflects real-world deployment scenarios where an intrusion detection system must adapt to diverse network conditions, hence measuring its generalization capabilities.

Table 3 shows the classification performance as measured by the F-Measure of the selected classifiers in a cross-dataset setting. It is possible to observe that all selected classifiers achieve significantly high detection accuracies when evaluated in the same environment in which they were trained. For example, the classifiers reached an average F-Measure of 0.95, 0.96, and 1.00 on the UNSW-NB15, CS-CIC-IDS, and BoT-IoT datasets, respectively. However, when these models are evaluated in a different environment, their performance deteriorates significantly, highlighting the challenge of generalization in ML-based NIDSs. This phenomenon is particularly evident in the case of the RF classifier. As an example, when trained on the UNSW-NB15 dataset, the RF classifier achieves strong classification performance within the same dataset, but when evaluated on the CS-CIC-IDS and BoT-IoT datasets, its F-Measure drops to only 0.10 and 0.16 respectively. This result indicates that features learned from one dataset may not generalize well to another, as different datasets capture distinct network traffic characteristics, attack patterns, and underlying distributions. This significant performance gap in cross-dataset evaluations suggests that traditional ML models struggle to adapt to unseen network environments, reinforcing the need for approaches that enhance model generalization.

Our second experiment aims to answer RQ2 and investigates how our proposed classification scheme can improve the generalization capabilities of ML-based NIDS. To achieve such a goal, we implement our proposed ensemble-based

classification scheme (see Section IV-A) on top of our proposed prototype. Given that we use three datasets, we build our ensemble $\mathcal{E}$ composed of three classifiers. In this case, the ensemble is composed of the DT, RF, and MLP classifiers built from the UNSW-NB15, CSE-CIC-IDS, and BoT-IoT datasets, respectively. The classifiers were chosen according to our ensemble-building strategy based on their accuracy on each dataset. The classification combination of the resulting ensemble is conducted through a majority voting strategy (see Eq. 1).

Table 3 presents the classification performance of our proposed model across different testing datasets. It is evident that our approach substantially enhances the F-Measure in a cross-dataset setting, demonstrating improved generalization capabilities compared to traditional ML-based NIDSs. While the F-Measure of our model remains slightly lower than that of classifiers trained and tested within the same dataset, it consistently outperforms models that are evaluated in a different environment from the one they were trained on. As an example, a direct comparison with the RF classifier further highlights the advantages of our approach. As previously discussed, the RF classifier exhibits a drastic decline in F-Measure when applied to datasets other than the one it was trained on, reaching only 0.10 when trained on CS-CIC-IDS and evaluated on BoT-IoT. In contrast, our proposed model maintains a significantly higher F-Measure under the same conditions, with an F-Measure of 0.56, reinforcing its ability to generalize effectively across diverse network environments. This result shows the limitations of traditional classifiers in handling real-world variability and further validates the effectiveness of our ensemble-based approach in improving the generalization in NIDSs.

Figure 3 directly compares the F-Measure of our proposal vs. the selected classifiers. It is possible to observe that our approach consistently improves the F-Measure in most cases, indicating that the model effectively learns generalizable patterns rather than overfitting specific dataset characteristics. This improvement translates into better detection capabilities in real-world scenarios, where variations in network traffic and attack patterns are common. The ability to sustain a higher F-Measure across multiple datasets further confirms the robustness of our model, making it a more reliable solution for intrusion detection in dynamic and evolving network environments.

C. SCALING ML-BASED NIDS DETECTION

Finally, we answer RQ3 and investigate the scaling capabilities of our proposed model when operating with our proposed ensemble-based classification model. To achieve such a goal, we investigate how our proposed ensemble can scale inference and model training as implemented through our proposal prototype (see Section V). In this case, we assess the scaling performance of our scheme vs. the traditional approach when varying the number of deployed Apache Spark workers (see Fig. 2). The goal is further investigate the scaling capabilities of our scheme and also if our proposed ensemble approach

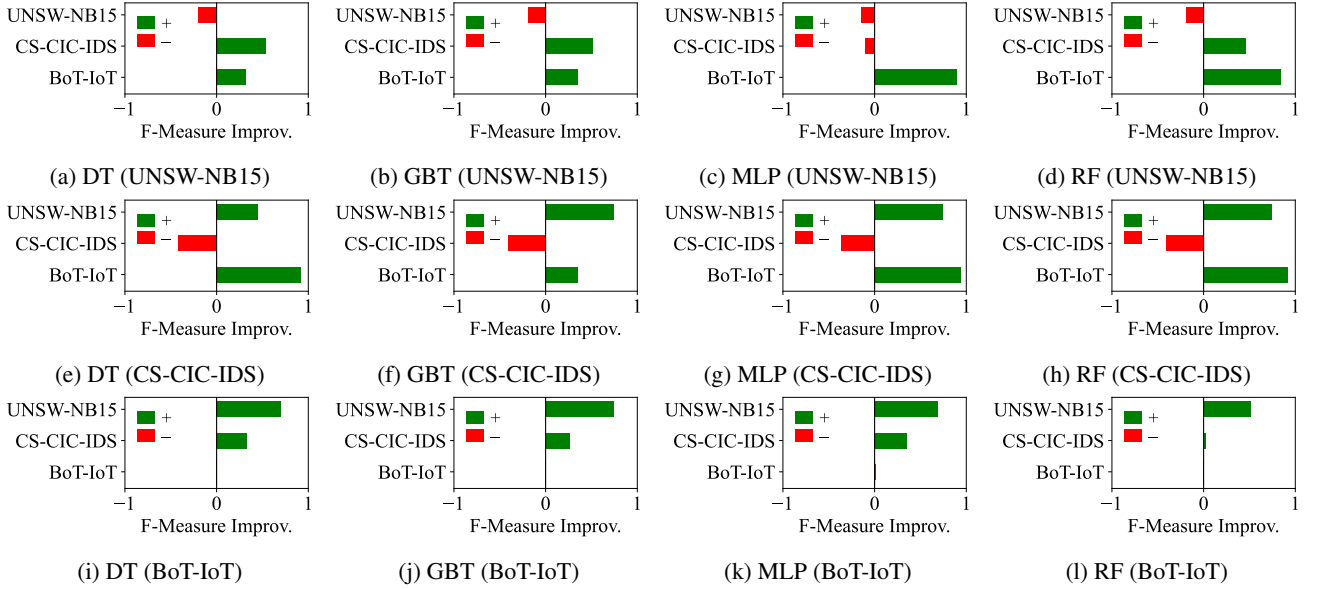


FIGURE 3: F-Measure comparison of our proposed scheme vs. traditional single classifier on the selected datasets. The caption denotes the classifier used and the training environment. Positive (+) values denote the F-Measure improvement of our proposal, while negative (−) values denote the decrease in F-Measure.

significantly degrade inference scaling.

Figure 4 illustrates the inference scaling of our proposed model as the number of deployed workers increases. The results demonstrate that our approach can be effectively scaled in a distributed environment, allowing for efficient processing of network events while maintaining high classification performance. This scalability ensures that the model can handle increasing traffic loads without a significant drop in inference throughput, making it well-suited for real-time intrusion detection in high-speed networks.

Additionally, our proposed ensemble-based classification scheme achieves comparable inference throughput to traditional ML techniques. Despite incorporating multiple classifiers, the ensemble does not introduce excessive computational overhead, enabling near real-time classification at scale. This is particularly evident when comparing our model to the RF classifier. For instance, when deployed with three workers, the RF classifier achieves an inference throughput of $\approx 1.27\text{M}$ events per second, whereas our proposed model reaches $\approx 1.07\text{M}$, demonstrating that our approach can deliver similar throughput with significantly better generalization capabilities. Overall, these results highlight the effectiveness of our distributed implementation. By leveraging a scalable architecture and an efficient ensemble classification scheme, our approach ensures both high detection accuracy and real-time inference capabilities, addressing the key challenges of deploying ML-based NIDSs in large-scale and dynamic network environments.

The second evaluation aims to investigate our proposed model performance during the training phase. To achieve such a goal, we assess the model training time of our scheme according to the number of workers and the time required by

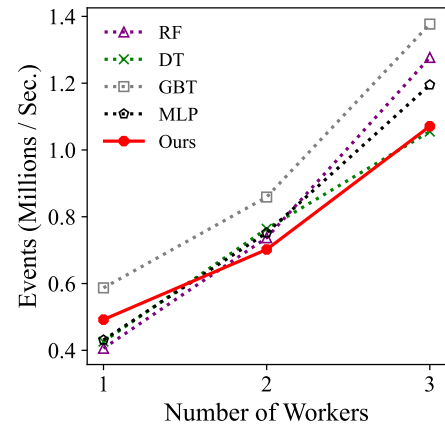


FIGURE 4: Scaling of inference task of our proposed model vs. traditional single classifiers approach.

each classifier forming the ensemble. The goal is to assess how the training process of our proposed ensemble can be effectively scaled as the number of Apache Workers increase.

Figure 5 shows the model training time of our proposed ensemble according to the number of deployed Apache Workers. The results demonstrate that our proposed model training procedure can be effectively scaled, allowing it to handle large volumes of training data in a distributed manner. Furthermore, the proposed ensemble approach scales according to the underlying classifiers used in the ensemble. Since each classifier in the ensemble can be trained in parallel, the overall training process benefits from the distributed nature of the implementation. This enables efficient utilization of computational resources, ensuring that even complex models can be

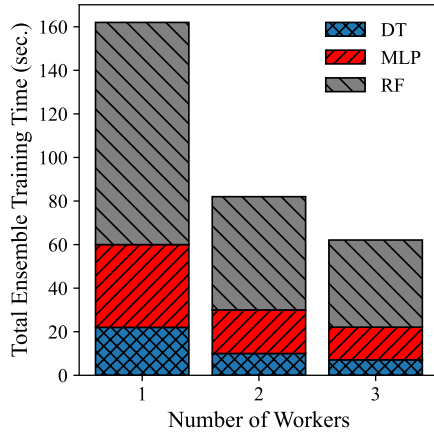


FIGURE 5: Scaling of model training task of our proposed model.

trained within a reasonable time frame. As the number of deployed Apache Workers increases, the training time decreases accordingly. Specifically, our results indicate that an increase in the number of workers results in a decrease in training time, for instance from 162 seconds with one worker to only 62 seconds with three workers. This reduction highlights the efficiency of our distributed training pipeline, allowing faster model updates while maintaining high classification performance.

VII. CONCLUSION

Network intrusion detection in high-speed environments presents significant challenges due to the need for real-time event ingestion and accurate inference at scale. Traditional ML-based NIDS struggle with generalization across different network environments, leading to a significant drop in detection accuracy when applied to unseen datasets. Additionally, high computational demands for model training and inference hinder their deployment in large-scale distributed settings. Addressing these challenges requires a scalable and efficient approach that ensures both high accuracy and adaptability to evolving network conditions.

To tackle these issues, we proposed a distributed ensemble-based NIDS that leverages a scalable Big Data framework for efficient model training, inference, and updating. Our architecture decouples event ingestion and inference, ensuring high-speed processing while maintaining accurate classification performance. The proposed system dynamically retrieves the latest model versions from a distributed storage, enabling continuous updates to adapt to emerging threats. Furthermore, by implementing inference and model updates on Apache Spark, our approach ensures that both tasks scale efficiently with the number of deployed computational resources. Experimental results demonstrated that our approach significantly improves generalization capabilities compared to traditional classifiers, reducing performance degradation in cross-dataset evaluations. Our proposed ensemble-based method achieves competitive inference throughput, main-

taining efficiency while improving classification accuracy across multiple datasets. Additionally, the distributed training pipeline scales effectively, reducing model update times as more workers are added and ensuring timely and reliable model deployment in large-scale network environments.

Future works include addressing near real-time model updates in an incremental manner and the integration of DNN-based classification.

ACKNOWLEDGMENT

This work was partially sponsored by the Brazilian National Council for Scientific and Technological Development (CNPq), grants n^o 407879/2023-4, 442262/2024-8, and 302937/2023-44.

REFERENCES

- [1] J. P. Omer Yoachimik, "Record-breaking 5.6 tbps ddos attack and global ddos trends for 2024 q4." 2025. [Online]. Available: <https://blog.cloudflare.com/ddos-threat-report-for-2024-q4/>
- [2] S. Gamage and J. Samarabandu, "Deep learning methods in network intrusion detection: A survey and an objective comparison," *Journal of Network and Computer Applications*, vol. 169, p. 102767, Nov. 2020. [Online]. Available: <http://dx.doi.org/10.1016/j.jnca.2020.102767>
- [3] L. Le Jeune, T. Goedeme, and N. Mentens, "Machine learning for misuse-based network intrusion detection: Overview, unified evaluation and feature choice comparison framework," *IEEE Access*, vol. 9, p. 63995–64015, 2021. [Online]. Available: <http://dx.doi.org/10.1109/ACCESS.2021.3075066>
- [4] E. K. Viegas, A. O. Santin, and L. S. Oliveira, "Toward a reliable anomaly-based intrusion detection in real-world environments," *Computer Networks*, vol. 127, p. 200–216, Nov. 2017. [Online]. Available: <http://dx.doi.org/10.1016/j.comnet.2017.08.013>
- [5] M. A. R. Putra, T. Ahmad, and D. P. Hostiadi, "B-cat: a model for detecting botnet attacks using deep attack behavior analysis on network traffic flows," *Journal of Big Data*, vol. 11, no. 1, Apr. 2024. [Online]. Available: <http://dx.doi.org/10.1186/s40537-024-00900-1>
- [6] A. Shiravani, M. H. Sadreddini, and H. N. Nahook, "Network intrusion detection using data dimensions reduction techniques," *Journal of Big Data*, vol. 10, no. 1, Mar. 2023. [Online]. Available: <http://dx.doi.org/10.1186/s40537-023-00697-5>
- [7] W. Wei, Y. Chen, Q. Lin, J. Ji, K.-C. Wong, and J. Li, "Multi-objective evolving long-short term memory networks with attention for network intrusion detection," *Applied Soft Computing*, vol. 139, p. 110216, May 2023. [Online]. Available: <http://dx.doi.org/10.1016/j.asoc.2023.110216>
- [8] M. Wang, N. Yang, D. H. Gunasinghe, and N. Weng, "On the robustness of ml-based network intrusion detection systems: An adversarial and distribution shift perspective," *Computers*, vol. 12, no. 10, p. 209, Oct. 2023. [Online]. Available: <http://dx.doi.org/10.3390/computers12100209>
- [9] J. Ni, W. Chen, J. Tong, H. Wang, and L. Wu, "High-speed anomaly traffic detection based on staged frequency domain features," *Journal of Information Security and Applications*, vol. 77, p. 103575, Sep. 2023. [Online]. Available: <http://dx.doi.org/10.1016/j.jisa.2023.103575>
- [10] M. Mahdavisarif, S. Jamali, and R. Fotuhi, "Big data-aware intrusion detection system in communication networks: a deep learning approach," *Journal of Grid Computing*, vol. 19, no. 4, Oct. 2021. [Online]. Available: <http://dx.doi.org/10.1007/s10723-021-09581-z>
- [11] N. Hussen, S. M. Elghamrawy, M. Salem, and A. I. El-Desouky, "A fully streaming big data framework for cyber security based on optimized deep learning algorithm," *IEEE Access*, vol. 11, p. 65675–65688, 2023. [Online]. Available: <http://dx.doi.org/10.1109/ACCESS.2023.3281893>
- [12] F. Jemili, "Intelligent intrusion detection based on fuzzy big data classification," *Cluster Computing*, vol. 26, no. 6, p. 3719–3736, Oct. 2022. [Online]. Available: <http://dx.doi.org/10.1007/s10586-022-03769-y>
- [13] D. Jin, S. Chen, H. He, X. Jiang, S. Cheng, and J. Yang, "Federated incremental learning based evolvable intrusion detection system for zero-day attacks," *IEEE Network*, vol. 37, no. 1, p. 125–132, Jan. 2023. [Online]. Available: <http://dx.doi.org/10.1109/MNET.018.2200349>

- [14] S. Akili, S. Purtzel, and M. Weidlich, "Decopa: Query decomposition for parallel complex event processing," *Proceedings of the ACM on Management of Data*, vol. 2, no. 3, p. 1–26, May 2024. [Online]. Available: <http://dx.doi.org/10.1145/3654935>
- [15] M. G. Rodrigues, E. K. Viegas, A. O. Santin, and F. Enembreck, "A mlops architecture for near real-time distributed stream learning operation deployment," *Journal of Network and Computer Applications*, vol. 238, p. 104169, Jun. 2025. [Online]. Available: <http://dx.doi.org/10.1016/j.jnca.2025.104169>
- [16] M. Cantone, C. Marrocco, and A. Bria, "Machine learning in network intrusion detection: A cross-dataset generalization study," *IEEE Access*, vol. 12, p. 144489–144508, 2024. [Online]. Available: <http://dx.doi.org/10.1109/ACCESS.2024.3472907>
- [17] F. Ullah, G. Srivastava, S. Ullah, K. Yoshigoe, and Y. Zhao, "Nids-vs-b: Network intrusion detection system for vanet using spark-based big data optimization and transfer learning," *IEEE Transactions on Consumer Electronics*, vol. 70, no. 1, p. 1798–1809, Feb. 2024. [Online]. Available: <http://dx.doi.org/10.1109/TCE.2023.3328320>
- [18] X. Song and Q. Ma, "Intrusion detection using federated attention neural network for edge enabled internet of things," *Journal of Grid Computing*, vol. 22, no. 1, Jan. 2024. [Online]. Available: <http://dx.doi.org/10.1007/s10723-023-09725-3>
- [19] M. Sarhan, S. Layeghy, and M. Portmann, "Towards a standard feature set for network intrusion detection system datasets," *Mobile Networks and Applications*, vol. 27, no. 1, p. 357–370, Nov. 2021. [Online]. Available: <http://dx.doi.org/10.1007/s11036-021-01843-0>
- [20] W. Jiang, H. Han, Y. Zhang, J. Mu, and A. Shankar, "Intrusion detection with federated learning and conditional generative adversarial network in satellite-terrestrial integrated networks," *Mobile Networks and Applications*, Oct. 2024. [Online]. Available: <http://dx.doi.org/10.1007/s11036-024-02435-4>
- [21] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and don'ts of machine learning in computer security," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 3971–3988. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity22/presentation/arp>
- [22] J. A. Simioni, E. K. Viegas, A. O. Santin, and E. de Matos, "An energy-efficient intrusion detection offloading based on dnn for edge computing," *IEEE Internet of Things Journal*, p. 1–1, 2025. [Online]. Available: <http://dx.doi.org/10.1109/IJOT.2025.3544060>
- [23] E. Viegas, A. O. Santin, and V. Abreu Jr, "Machine learning intrusion detection in big data era: A multi-objective approach for longer model lifespans," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 1, p. 366–376, Jan. 2021. [Online]. Available: <http://dx.doi.org/10.1109/TNSE.2020.3038618>
- [24] E. Viegas, A. Santin, A. Bessani, and N. Neves, "Bigflow: Real-time and reliable anomaly-based intrusion detection for high-speed networks," *Future Generation Computer Systems*, vol. 93, p. 473–485, Apr. 2019. [Online]. Available: <http://dx.doi.org/10.1016/j.future.2018.09.051>
- [25] D. Suja Mary, L. Jaya Singh Dhas, A. Deepa, M. A. Chaurasia, and C. Jaspin Jeba Sheela, "Network intrusion detection: An optimized deep learning approach using big data analytics," *Expert Systems with Applications*, vol. 251, p. 123919, Oct. 2024. [Online]. Available: <http://dx.doi.org/10.1016/j.eswa.2024.123919>
- [26] Y. Liu, X. Zhang, B. Liu, and X. Zhao, "The research and analysis of efficiency of hardware usage base on hdfs," *Cluster Computing*, vol. 25, no. 5, p. 3719–3732, May 2022. [Online]. Available: <http://dx.doi.org/10.1007/s10586-022-03597-0>
- [27] G. Cheng, S. Ying, B. Wang, and Y. Li, "Efficient performance prediction for apache spark," *Journal of Parallel and Distributed Computing*, vol. 149, p. 40–51, Mar. 2021. [Online]. Available: <http://dx.doi.org/10.1016/j.jpdc.2020.10.010>
- [28] T. P. Raptis and A. Passarella, "A survey on networked data streaming with apache kafka," *IEEE Access*, vol. 11, p. 85333–85350, 2023. [Online]. Available: <http://dx.doi.org/10.1109/ACCESS.2023.3303810>
- [29] Z. Ye, J. Luo, W. Zhou, M. Wang, and Q. He, "An ensemble framework with improved hybrid breeding optimization-based feature selection for intrusion detection," *Future Generation Computer Systems*, vol. 151, p. 124–136, Feb. 2024. [Online]. Available: <http://dx.doi.org/10.1016/j.future.2023.09.035>
- [30] C. Hazman, A. Guezzaz, S. Benkirane, and M. Azrou, "lids-sioel: intrusion detection framework for iot-based smart environments security using ensemble learning," *Cluster Computing*, vol. 26, no. 6, p. 4069–4083, Nov. 2022. [Online]. Available: <http://dx.doi.org/10.1007/s10586-022-03810-0>
- [31] M. Mohy-Eddine, A. Guezzaz, S. Benkirane, M. Azrou, and Y. Farhaoui, "An ensemble learning based intrusion detection model for industrial iot security," *Big Data Mining and Analytics*, vol. 6, no. 3, p. 273–287, Sep. 2023. [Online]. Available: <http://dx.doi.org/10.26599/BDMA.2022.9020032>
- [32] M. A. Khan, N. Iqbal, Imran, H. Jamil, and D.-H. Kim, "An optimized ensemble prediction model using automl based on soft voting classifier for network intrusion detection," *Journal of Network and Computer Applications*, vol. 212, p. 103560, Mar. 2023. [Online]. Available: <http://dx.doi.org/10.1016/j.jnca.2022.103560>
- [33] A. M. Alsaffar, M. Nouri-Baygi, and H. M. Zolbanin, "Shielding networks: enhancing intrusion detection with hybrid feature selection and stack ensemble learning," *Journal of Big Data*, vol. 11, no. 1, Sep. 2024. [Online]. Available: <http://dx.doi.org/10.1186/s40537-024-00994-7>
- [34] M. Wang, N. Yang, Y. Guo, and N. Weng, "Learn-ids: Bridging gaps between datasets and learning-based network intrusion detection," *Electronics*, vol. 13, no. 6, p. 1072, Mar. 2024. [Online]. Available: <http://dx.doi.org/10.3390/electronics13061072>
- [35] S. Wali, Y. A. Farrukh, I. Khan, and N. D. Bastian, "Meta: Toward a unified, multimodal dataset for network intrusion detection systems," *IEEE Data Descriptions*, vol. 1, p. 50–57, 2024. [Online]. Available: <http://dx.doi.org/10.1109/IEEEDATA.2024.3482286>
- [36] Z. Fan, S. Sohail, F. Sabrina, and X. Gu, "Sampling-based machine learning models for intrusion detection in imbalanced dataset," *Electronics*, vol. 13, no. 10, p. 1878, May 2024. [Online]. Available: <http://dx.doi.org/10.3390/electronics13101878>
- [37] N. Niknami, V. Mahzoon, and J. Wu, "Ptn-ids: Prototypical network solution for the few-shot detection in intrusion detection systems," in *2024 IEEE 49th Conference on Local Computer Networks (LCN)*. IEEE, Oct. 2024, p. 1–9. [Online]. Available: <http://dx.doi.org/10.1109/LCN60385.2024.10639652>
- [38] G. Duan, Y. Fu, M. Cai, H. Chen, and J. Sun, "Dongting: A large-scale dataset for anomaly detection of the linux kernel," *Journal of Systems and Software*, vol. 203, p. 111745, Sep. 2023. [Online]. Available: <http://dx.doi.org/10.1016/j.jss.2023.111745>
- [39] A. Abid, F. Jemili, and O. Korbaa, "Real-time data fusion for intrusion detection in industrial control systems based on cloud computing and big data techniques," *Cluster Computing*, vol. 27, no. 2, p. 2217–2238, Jun. 2023. [Online]. Available: <http://dx.doi.org/10.1007/s10586-023-04087-7>
- [40] F. Jemili, R. Meddeb, and O. Korbaa, "Intrusion detection based on ensemble learning for big data classification," *Cluster Computing*, vol. 27, no. 3, p. 3771–3798, Nov. 2023. [Online]. Available: <http://dx.doi.org/10.1007/s10586-023-04168-7>
- [41] B. K. Pandey, V. M.R.M., S. Ahmad, C. Rodriguez, and D. Esenarro, "Expssoa-deep maxout: Exponential shuffled shepherd optimization based deep maxout network for intrusion detection using big data in cloud computing framework," *Computers and Security*, vol. 124, p. 102975, Jan. 2023. [Online]. Available: <http://dx.doi.org/10.1016/j.cose.2022.102975>
- [42] A. Kourid, S. Chikhi, and D. R. Recupero, "Fuzzy optimized v-detector algorithm on apache spark for class imbalance issue of intrusion detection in big data," *Neural Computing and Applications*, vol. 35, no. 27, p. 19821–19845, Jul. 2023. [Online]. Available: <http://dx.doi.org/10.1007/s00521-023-08783-8>
- [43] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*. IEEE, Nov. 2015, p. 1–6. [Online]. Available: <http://dx.doi.org/10.1109/MilCIS.2015.7348942>
- [44] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy*. SCITEPRESS - Science and Technology Publications, 2018. [Online]. Available: <http://dx.doi.org/10.5220/0006639801080116>
- [45] N. Moustafa, M. Keshk, K.-K. R. Choo, T. Lynar, S. Camtepe, and M. Whitty, "Dad: A distributed anomaly detection system using ensemble one-class statistical learning in edge networks," *Future Generation Computer Systems*, vol. 118, p. 240–251, May 2021. [Online]. Available: <http://dx.doi.org/10.1016/j.future.2021.01.011>

VINICIUS M. DE OLIVEIRA received the B.S. degree in Computer Science from Pontifical Catholic University of Paraná (PUCPR) in 2024. His research interests include machine learning, network analytics and computer security.

HENRIQUE M. DE OLIVEIRA received the B.S. degree in Computer Science from Pontifical Catholic University of Paraná (PUCPR) in 2024. His research interests include machine learning, network analytics and computer security.

GABRIEL M. DE OLIVEIRA received the B.S. degree in Computer Science from Pontifical Catholic University of Paraná (PUCPR) in 2024. His research interests include machine learning, network analytics and computer security.

JHONATAN GEREMIAS received the BS degree in computer science in 2016 and the MSC degree in computer science in 2020 from PUCPR. He is currently working toward his PhD degree in computer science at PUCPR. His research interests include machine learning, pattern recognition, image analysis and computer security.

EDUARDO K. VIEGAS received my BS in computer science in 2013, my MSC in computer science in 2016 from PUCPR, and my Ph.D. degree from PUCPR in 2018. He is an associate professor of the Graduate Program in Computer Science (PPGIa). His research interests include ML, network analytics, and computer security.