Journal of the Brazilian Computer Society, 2026, 32:1, doi: 10.5753/jbcs.2026.5608

A Reliable Stream Learning Model for Network Intrusion Detection Systems

Pedro Horchulhack [**Pontifícia Universidade Católica do Paraná** | ***pedro.horchulhack@ppgia.pucpr.br***]
Eduardo Kugler Viegas [**Pontifícia Universidade Católica do Paraná** | ***eduardo.viegas@ppgia.pucpr.br***]
Altair Olivo Santin [**Pontifícia Universidade Católica do Paraná** | ***santin@ppgia.pucpr.br***]

Graduate Program in Informatics (PPGIa), Pontifical Catholic University of Paraná, Bloco 8 – Parque Tecnológico – 2º andar, Rua Imaculada Conceição, 1155 - Prado Velho, 80215-901 - Curitiba - PR, Brazil.

Received: 24 February 2025 • **Accepted:** 23 June 2025 • **Published:** 02 March 2026

Abstract. Developing a reliable Network Intrusion Detection System (NIDS) remains a complex task due to the non-stationary nature of network traffic and the need for frequent updates to maintain high classification performance. Many existing approaches assume a stationary network environment, which overlooks the challenges associated with periodic model updates, such as the need for large amounts of properly labeled data and significant computational resources. This issue is particularly challenging for real-time applications, where minimizing delays and ensuring accuracy is crucial. This paper proposes an analysis of how changes in the network behavior negatively affects the long-term of ML-Based NIDS. For such a problem, it is proposed a new NIDS approach integrating stream learning with a reject option technique to simplify the model update process while ensuring consistent classification accuracy over time. The proposal uses stream learning classifiers to incrementally incorporate new data, while the reject option allows the system to evaluate the reliability of classifications before they are used for updates. The scheme operates with minimal intervention, with rejected instances stored for future updates and used to fine-tune the model over time, ensuring adaptation to evolving network conditions. Experimental results demonstrate that the proposed approach maintains high classification accuracy over a year, even without recurrent updates, and achieves significant improvements in true positive rates compared to traditional methods. The system can operate for up to three months without updates, with no significant degradation in performance.

Keywords: Machine Learning, Stream Learning, Network-based Intrusion Detection System, Reject Option

1 Introduction

Cyberattacks have steadily increased in recent years, with many of these threats leveraging network traffic to infiltrate systems and compromise sensitive data. As attackers employ more sophisticated techniques to evade traditional security measures, organizations must adopt robust defense mechanisms to safeguard their networks. Network-based Intrusion Detection System (NIDS) have become essential in this context, enabling real-time monitoring and analysis of network traffic to identify and mitigate potential threats before they cause significant damage.

Companies often rely on NIDSs to detect potential network attacks by employing two main detection approaches: *rule-based* or *behavior-based* [Liao *et al.*, 2013]. The former defines signatures corresponding to known attack patterns, enabling precise identification of previously known threats. In contrast, the latter analyzes statistical and behavioral patterns in network traffic to detect deviations from normal activity. Rule-based NIDSs relies on well-known attack patterns; hence, they usually achieve high true-positive rates in stationary environments but require continuous updates to their attack database and are incapable of detecting previously unknown threats [Nisioti *et al.*, 2018; Molina-Coronado *et al.*, 2020]. As a result, they are usually unsuitable for networks with highly dynamic behavior. In contrast, behavior-based techniques are more effective in handling non-stationary environments. This is because it learns from labeled network samples, enabling them to classify new events as malicious by recognizing patterns in the provided labels [Kilincer *et al.*, 2021]. Traditional behavior-based NIDSs suffer from high false-positive and false-negative rates due to the overlapping characteristics of normal and malicious behavior, significantly impacting their performance in real-world scenarios [Pereira and Maia, 2022].

Learning new patterns from network communications is a complex challenge. In real-world scenarios, the vast volume of network data makes the analysis required to distinguish between malicious and benign events both computationally intensive and resource-demanding. These difficulties arise from network traffic's inherent diversity and complexity, varying across protocols, traffic types, and user behaviors. In this context, Machine Learning (ML) techniques have emerged as a promising approach for detecting novel attacks, as they can identify subtle changes in network activity. Their ability to recognize anomalous patterns, even when they are sparse or highly similar, enables machine learning models to detect previously unknown attacks, representing a significant advancement over traditional detection methods [Apruzzese *et al.*, 2023].

Despite their sensitivity to new attacks, previous studies have often assumed that labeled data is always available.

However, in real-world scenarios, this is rarely the case. Human intervention is required to analyze and classify events as benign or malicious before retraining the ML model. Additionally, the training time and reliability of ML-based NIDSs become significant challenges when failing to account for the highly dynamic and non-stationary nature of real-world network environments [Kalita *et al.*, 2023]. Some studies have proposed updating ML models with newly collected data to address these issues. However, the sheer volume of network traffic data introduces additional challenges, particularly regarding the computational costs associated with training and storage. A promising solution lies in *stream learning* techniques, which allow ML models to incorporate new instances incrementally. This approach enables fast and lightweight updates, making it particularly suited for critical applications such as NIDSs. Nevertheless, most existing works overlook the complex nature of networked environments, leading to unrealistic solutions that fail to ensure robust security.

An outdated ML model in a NIDS significantly degrades the system's ability to detect emerging threats. As network environments evolve, new attack patterns and variations of existing threats emerge, often bypassing outdated detection mechanisms. Without regular updates, the ML model becomes increasingly ineffective, leading to a rise in false negatives, where malicious activities go undetected, and false positives, which generate unnecessary alerts and increase operational overhead. Additionally, outdated models may fail to adapt to legitimate changes in network behavior, causing disruptions in normal traffic classification. This lack of adaptability weakens security and diminishes trust in the system, making continuous model updates essential for maintaining effective intrusion detection.

Updating a traditional ML model in NIDS presents several challenges, primarily due to the need to retrain the model from scratch. This process is computationally expensive, requiring significant processing power and storage to handle the vast amounts of network traffic data. Moreover, identifying new behavior patterns is inherently difficult, as emerging threats often exhibit subtle deviations from normal traffic, making them hard to distinguish without extensive analysis. Constructing an updated dataset that accurately represents these evolving patterns is a critical challenge, as it requires continuous data collection, expert labeling, and validation. These constraints increase the time and resources required for model updates and introduce delays in deploying improved detection mechanisms, leaving networks vulnerable to emerging threats.

Contribution. To address these challenges, this paper proposes a novel intrusion detection strategy based on stream learning, enabling model updates with significantly lower computational costs. Proposed experiments evaluates how changes in network traffic behavior jeopardize traditional ML-based techniques. The hypothesis is that outdated models shows poorer True-Positive (TP) and True-Negative (TN) rates on the long-term. The core insight is to reject outdated events based on a confidence threshold and selectively use them for model updates. By filtering unreliable events, experts can analyze and accurately label them, ensuring a more reliable learning process while minimizing unnecessary retraining. Additionally, the proposed method maintains high accuracy, making it well-suited for real-world deployment where both efficiency and reliability are critical. The proposed approach is implemented in three steps: (i) stream learning pool, (ii) classification assessment, and (iii) model updates. In the first step, a pool of stream learning classifiers evaluates network flows individually, enhancing classification confidence and reducing the computational costs associated with future updates. The second step focuses on assessing classification confidence through a reject option mechanism. Events that fall below a predefined confidence threshold are rejected and set aside for further review. This allows for human intervention in a subset of uncertain events, enabling experts to label them accurately. In the final step, the rejected events are periodically used to update the stream learning pool. This update can occur after a specific period, such as one month, once the correct labels become available following an incident.

In summary, the main contributions of this paper are:

- An evaluation of the classification reliability throughout a one-year interval of widely used ML-based NIDSs. Our experiments suggests that current approaches are unreliable for long-interval classifications, demanding unfeasible periodic model updates to be conducted;
- A new NIDS implemented through stream learning algorithms coped with a reject option classification. Our proposed scheme can identify unreliable events for classification without expert supervision;

Roadmap. The remainder of this paper is organized as follows. Section 2 introduces the NIDS and ML topics and how both relate. Section 3 describes the current state of the art regarding attempts to solve the problem within the paper. Section 4 presents the problem, the materials and methods, and show evidence that changes in the network traffic may jeopardize ML-based NIDS. Section 5 describes the paper proposal, specifically the materials and methods. Section 6 shows the results and discussions compassing them. Finally, Section 7 concludes the paper.

2 Preliminaries

This section overviews the current implementation of ML-based NIDSs and its challenges.

2.1 Network-based Intrusion Detection

NIDS schemes are designed to detect attacks within a network and can be implemented through four sequential stages: *Data Acquisition*, *Feature Extraction*, *Classification*, and *Alert* [Molina-Coronado *et al.*, 2020]. Data is collected from the monitored environment in the *Data Acquisition* stage, typically consisting of network packets gathered from a Network Interface Card (NIC). These collected data are then passed to the *Feature Extraction* stage, where behavioral features—usually represented as network flows summarizing communication between two endpoints within a given time window—are extracted. Table 1 shows an example of features that can be extracted from a given network communication flow. After feature extraction, the *Classification* module classifies each network flow as either *normal* or *attack*, typically using ML

Table 1. Network flow features extracted for network traffic analysis.

Group	Feature Name	Description
Flow	minimumInterArrivalTime	Minimum inter-arrival time between packets in the flow.
	quartileFirstInterArrivalTime	First quartile of inter-arrival times between packets in the flow.
	medianInterArrivalTime	Median inter-arrival time between packets in the flow.
	avgInterArrivalTime	Average inter-arrival time between packets in the flow.
	quartileThirdInterArrivalTime	Third quartile of inter-arrival times between packets in the flow.
	maximumInterArrivalTime	Maximum inter-arrival time between packets in the flow.
	varianceInterArrivalTime	Variance of inter-arrival times between packets in the flow.
	minimumDataWire	Minimum data wire size in the flow.
	quartileFirstDataWire	First quartile of data wire sizes in the flow.
	medianDataWire	Median data wire size in the flow.
	avgDataWire	Average data wire size in the flow.
	quartileThirdDataWire	Third quartile of data wire sizes in the flow.
	maximumDataWire	Maximum data wire size in the flow.
	varianceDataWire	Variance of data wire sizes in the flow.
	total_packets_flow	Total number of packets exchanged in the flow.
	ack_pkts_sent_flow	Number of acknowledgment packets exchanged in the flow.
	pure_acks_sent_flow	Number of pure acknowledgment packets exchanged in the flow.
	pushed_pkts_sent_flow	Number of pushed packets exchanged in the flow.
	syn_pkts_sent_flow	Number of SYN packets exchanged in the flow.
	fin_pkts_sent_flow	Number of FIN packets exchanged in the flow.
	urgent_pkts_sent_flow	Number of urgent packets exchanged in the flow.
	throughput_flow	Throughput of packets in the flow.
Source to Destination / Destination to Source	minimumInterArrivalTime_dir	Minimum inter-arrival time for packets.
	quartileFirstInterArrivalTime_dir	First quartile of inter-arrival times for packets.
	medianInterArrivalTime_dir	Median inter-arrival time for packets.
	avgInterArrivalTime_dir	Average inter-arrival time for packets.
	quartileThirdInterArrivalTime_dir	Third quartile of inter-arrival times for packets.
	maximumInterArrivalTime_dir	Maximum inter-arrival time for packets.
	varianceInterArrivalTime_dir	Variance of inter-arrival times for packets.
	minimumDataWire_dir	Minimum data wire size for packets.
	quartileFirstDataWire_dir	First quartile of data wire sizes for packets.
	medianDataWire_dir	Median data wire size for packets.
	avgDataWire_dir	Average data wire size for packets.
	quartileThirdDataWire_dir	Third quartile of data wire sizes for packets.
	maximumDataWire_dir	Maximum data wire size for packets.
	varianceDataWire_dir	Variance of data wire sizes for packets.
	total_packets_dir	Total number of packets sent.
	ack_pkts_sent_dir	Number of acknowledgment packets sent.
	pure_acks_sent_dir	Number of pure acknowledgment packets sent.
	pushed_pkts_sent_dir	Number of pushed packets sent.
	syn_pkts_sent_dir	Number of SYN packets sent.
	fin_pkts_sent_dir	Number of FIN packets sent.
	urgent_pkts_sent_dir	Number of urgent packets sent.
	throughput_dir	Throughput of packets.

models to perform this task. Finally, the *Alert* module generates an alert if a network flow is classified as an *attack*, notifying the network administrator.

Several techniques have been proposed for the classification of network flows, where the authors often resort to ML-based schemes, typically through pattern recognition (*batch-based*) approaches [Molina-Coronado *et al.*, 2020]. The training procedure of a traditional batch-based ML NIDS typically follows a three-step process: *training*, *testing*, and *validation*. In the first step, the model is trained on a large dataset containing normal and attack network flows. This dataset is used to adjust the model's parameters, allowing it to learn patterns and distinguish between benign and malicious traffic. Once the model has been trained, it is evaluated in the second step using a separate testing dataset, which the model has not seen during training. This step assesses the model's ability to generalize and correctly classify unseen data. Finally, the validation step involves evaluating the model's performance on a validation set distinct from the training and testing datasets. The validation process ensures that the model is not overfitting to the training data and helps fine-tune hyperparameters to optimize performance. While effective in controlled environments, this batch-based approach often struggles to adapt to dynamic, real-time network conditions.

2.2 Behavior-based NIDS

ML is a subfield of artificial intelligence focused on learning patterns from a given dataset. The learning task can occur through two distinct approaches: supervised and unsupervised learning. In supervised learning, a classification algorithm is trained using a dataset that includes features and corresponding labels to differentiate between classes [Ahmad *et al.*, 2021]. The algorithm searches for statistical patterns within the feature set that enable it to identify the class associated with each label correctly. For example, a network flow characterized by a finite set of features (see Table 1) can be classified as either an attack or benign based on training data with known attack and non-attack flow labels. In contrast, unsupervised learning does not rely on labeled data. Instead, the focus is on identifying patterns or groups within the feature set. This approach is particularly useful for labeling network events, as demonstrated in [MAWI, 2021], where the goal is to detect anomalies or cluster similar instances without predefined labels. Finally, all features in Table 1 may be used for classification tasks.

Both approaches rely on batch processing, meaning that a pre-collected dataset is used for training. This method incurs high computational costs, as it requires substantial storage capacity and processing of all instances simultaneously. In contrast, stream learning techniques are based on incrementally streamed process data. Each incoming data point, or stream instance, is incorporated into the model in real time. This approach significantly reduces computational overhead by eliminating the need to process large batches, enabling more efficient and adaptive learning on a per-event basis.

To deploy an effective NIDS with a reliable ML model, a three-step process—*training*, *validation*, and *testing*—is typically followed [Viegas *et al.*, 2017]. The *training* phase involves constructing an ML model by analyzing labeled network flows from a training dataset. The goal is to develop a behavioral model capable of distinguishing between *attack* and *normal* events. Consequently, the training dataset must be extensive and well-labeled to ensure effectiveness. Next, the *validation* phase focuses on refining the model through parameter optimization and feature selection, enhancing its ability to generalize to unseen data. To maintain reliability, the datasets used in the *training* and *validation* phases are distinct, preventing event overlap [Bouke and Abdullah, 2023]. Finally, using a separate test dataset, the *testing* phase evaluates the optimized model's accuracy. Upon achieving satisfactory performance, the model is deployed for real-time classification of network traffic [Apruzzese *et al.*, 2023].

In recent years, several studies have explored the application of *stream* learning techniques to scenarios where behavior evolves over time [Abbasi *et al.*, 2021; Wang *et al.*, 2024]. Unlike *batch* learning approaches, which require the processing of entire datasets, *stream* learning allows for incremental model updates, where each new instance is integrated into the model. This approach significantly reduces computational costs, as it enables the use of the current model for updating rather than retraining from scratch. Stream learning has widespread use in diverse fields, including finance, telemetry, and industrial applications. Stream learning is useful for scenarios where data continuously evolves, allowing models to update incrementally as new instances are introduced. Instead of retraining the entire model from scratch, stream learning enables real-time updates, where each incoming data point is processed and incorporated into the existing model. This approach reduces computational overhead, as it minimizes the need for large-scale retraining, making it suitable for applications with large, constantly changing datasets [Bahri *et al.*, 2021]. Incremental model updates are particularly beneficial in dynamic environments, such as network intrusion detection systems, where timely adaptation to new patterns and behaviors is crucial [Viegas *et al.*, 2019]. By leveraging stream learning, models can remain up-to-date without the resource-intensive process of batch retraining, ensuring that they accurately reflect the latest trends in the data.

While stream learning offers the advantage of incremental model updates, its application in NIDSs presents significant challenges due to the dynamic nature of network environments. Networks constantly evolve, with traffic patterns, user behaviors, and attack strategies changing. This volatility makes it difficult for an intrusion detection system to reliably capture new attack patterns and adapt to shifting network dynamics. Additionally, the diverse and often unpredictable nature of network data can result in noise, outliers, and overlapping characteristics between normal and malicious behaviors, further complicating the task of incremental learning. Consequently, effectively updating models in real time without sacrificing accuracy or performance requires careful management of model adaptability and incoming data quality. Achieving reliable and efficient incremental updates in such a dynamic environment demands adequate techniques to handle the complexities and variability of network traffic.

3 Related Works

Over the past years, several works have proposed new ML-based NIDSs in the literature able to reach significantly high detection accuracies. As an example M. A. Talukder *et al.* [Talukder *et al.*, 2023] proposes a dependable intrusion detection model implemented with oversampling techniques. Their proposed model conducts intrusion detection through an XGBoost classifier that copes with SMOTE for data balancing, yielding higher accuracies than traditional unbalanced approaches. However, classification reliability in evolving network traffic behavior is overlooked. Similarly, T. Saba *et al.* [Saba *et al.*, 2022] proposes a deep learning approach for intrusion detection in resource-constrained devices. Their scheme improves accuracy compared to traditional shallow-based techniques but overlooks the challenges associated with model updates and classification reliability. G. Apruzzese *et al.* [Apruzzese *et al.*, 2022] addresses the challenges associated with classification generalization through a cross-dataset evaluation approach. Their approach conducts model training considering multiple dataset performance. Unfortunately, although their approach tackles the challenges associated with classification reliability, the evolving behavior of network traffic is neglected. Another deep learning approach was proposed by B. Sharma *et al.* [Sharma *et al.*, 2023] for intrusion detection on resource-constrained devices. Their scheme relies on feature selection to reduce computational costs with-

Table 2. A summary of related work and the characteristics of their intrusion detection strategy.

Work	**Model Updates**	**Detection Reliability**	**Dynamic Network**	**Incremental Update**	**Realistic Dataset**
M. A. Talukder *et al.*	×	×	×	×	✓
T. Saba *et al.*	×	×	×	×	×
G. Apruzzese *et al.*	×	×	×	×	✓
B. Sharma *et al.*	×	×	×	×	✓
J. Du *et al.*	×	×	×	×	×
J. Luxemburk *et al.*	×	✓	×	×	✓
M. Verkerken *et al.*	×	✓	×	×	×
A. Paya *et al.*	×	✓	✓	×	×
B. O. George *et al.*	✓	×	×	×	×
M. Sarhan *et al.*	✓	✓	×	✓	×
I. A. Kandhro *et al.*	×	×	×	✓	✓
A. Abdulboriy *et al.*	×	×	×	✓	×
Z. Nie *et al.*	×	×	✓	✓	×
X. Xu *et al.*	×	×	×	✓	×
Ours	✓	✓	✓	✓	✓

out affecting detection accuracy. Similarly, model updates and classification reliability were overlooked when subject to network traffic behavior changes. J. Du *et al* [Du *et al.*, 2023] also relies on increasing model complexity to improve detection accuracy. Their approach increases the detection rate on multiple datasets but overlooks the challenges associated with classification unreliability and model updates.

The classification reliability in intrusion detection is rarely considered in the literature. J. Luxemburk *et al.* [Luxemburk and Čejka, 2023] proposes a service identification approach through a neural network implementation. Their scheme yields high accuracy by rejecting unknown services during the training phase. Unfortunately, the application in the intrusion detection domain is overlooked. M. Verkerken *et al.* [Verkerken *et al.*, 2021] use unsupervised machine learning techniques to improve the generalization capabilities of intrusion detection. Their approach can operate without expert supervision but overlooks the challenges associated with new network traffic behavior. A. Paya *et al.* [Paya *et al.*, 2024] addresses the reliability challenge by actively selecting the classifiers for intrusion detection from an ensemble. Their approach can detect intrusion attempts but assumes a static network traffic behavior. To ease model updates B. O. George *et al.* [Olanrewaju-George and Pranggono, 2025] proposes a federated learning training procedure for unsupervised intrusion detection. Their approach eases model update costs but overlooks how new training events can be obtained and how periodic model updates can be conducted.

Usually, in the literature, intrusion detection research is conducted by assuming a static network traffic behavior. In practice, proposed schemes overlook the challenges associated with evolving network traffic behavior, such as the need for periodic model retraining, updated dataset building, and the identification of unreliable classifications. M. Sarhan *et al.* [Sarhan *et al.*, 2021] noted the classification reliability challenges associated with the evolving behavior of network traffic. The authors showed that proposed ML-based NIDSs fail to account for the dynamic behavior of network traffic, usually only achieving high detection accuracies in the dataset utilized for training purposes. I. A. Kandhro *et al.* [Kandhro *et al.*, 2023] uses feature selection to improve the generalization capabilities of a deep learning model in intrusion detection. Their approach improved generalization but assumed a static behavior of network traffic. A. Abdulboriy *et al.* [Abdulboriy and Shin, 2024] proposes an incremental learning approach to address the evolving behavior of network traffic. The authors rely on an incremental majority voting NIDSs to pave the way for addressing model updates. Unfortunately, how unknown network traffic behavior can be identified is overlooked. Z. Nie *et al.* [Nie *et al.*, 2024] proposes an incremental NIDSs implemented with a neural network with adaptive moment estimation. Their approach can incorporate new network traffic behavior into the model; however, it neglects the identification of unreliable classifications. X. Xu *et al.* [Xu *et al.*, 2024] relies on a self-paced class incremental learning to incorporate new network traffic behavior. Their approach improves classification accuracy and reduces the costs associated with model updates. Similarly, how new network traffic behavior can be identified is overlooked.

Table 2 overviews the literature on intrusion detection. Many existing approaches fail to implement realistic model update mechanisms, often relying on static, batch-based retraining methods that do not account for the continuous evolution of network environments. This reliance on outdated update strategies challenges adapting to emerging threats and evolving traffic patterns as models become increasingly obsolete over time. Additionally, current methods frequently overlook the need to detect model unreliability. Without robust performance monitoring, these systems risk deploying models that have degraded accuracy, undermining network security and leaving vulnerabilities unaddressed.

Furthermore, many studies rarely consider the dynamic nature of real-world networks, where variability in traffic volume, protocol usage, and attack strategies demands a flexible, responsive approach. The lack of attention to network dynamism means that even well-performing models can quickly become ineffective in the face of unforeseen behavioral shifts. Incremental model updates, which can be used for maintaining relevance and responsiveness in such fluctuating environments, are also frequently neglected. Traditional approaches often require complete retraining of the model, a computationally intensive and time-consuming process, making it impractical for timely adaptation in live settings.

Finally, many of these studies utilize unrealistic datasets that fail to capture the complexity and variability of actual network traffic. This reliance on oversimplified or simulated data limits the generalizability of the findings and hinders the development of solutions that can operate effectively under real-world conditions. Each of these challenges – from impractical update mechanisms and the absence of reliability detection to the disregard for dynamic network behavior, inefficient retraining protocols, and the use of unrealistic datasets – highlights critical gaps in current research.

4 Problem statement

This section further explores the impact of changes in network behavior on ML-based NIDSs and investigates how model updates can help mitigate the resulting challenges.

4.1 Dataset

Many intrusion detection techniques have been developed using unrealistic datasets. Kilincer *et al.* examined the most commonly used datasets for NIDS [Kilincer *et al.*, 2021], highlighting a critical issue: most datasets consist of simulated, short-term, and stationary examples. An ideal dataset should be highly diverse, incorporating real and valid traffic from production environments. To leverage ML effectively, events must be properly labeled as either *attack* or *normal*, which can be achieved through expert annotation or automated methods such as unsupervised ML. Addressing the stationarity of existing datasets requires data collection over extended periods at regular intervals. This approach facilitates more realistic analysis and training of NIDS, capturing the natural evolution of network traffic. Furthermore, the dataset should be publicly accessible to the scientific literature to enable broader evaluation, analysis, and knowledge extraction. A real network traffic intrusion detection dataset is essential for developing effective NIDS. Two primary approaches can be used to construct such a dataset: collecting network packets from a production environment or generating traffic within a controlled testbed. The first approach provides a more realistic representation of real-world conditions but poses significant challenges related to privacy, data labeling, and data collection [Liu *et al.*, 2022]. In contrast, a controlled testbed data collection simplifies labeling and acquisition processes but often results in datasets that lack diversity and fail to reflect real network traffic dynamics accurately.

To address these challenges, this study leverages the MAWIFlow dataset [Viegas *et al.*, 2019], which provides a more realistic behavior for intrusion detection research. The dataset is sourced from network traffic captured at Samplepoint-F within the MAWI archive [MAWI, 2021], ensuring that it consists of real, valid, and highly diverse traffic patterns. Network traffic data is collected daily in 15-minute intervals from a transit link between Japan and the USA, covering a wide range of network activities and potential anomalies. This work utilizes network traffic from 2014, comprising over 2.6 TB of data and approximately 40 billion packets for evaluation. It consists of $81,186,253$ of samples, having two classes: *attack* and *normal*. The proportion of the classes are, respectively, 4.33% and 95.66%. The number of features used was 58, without feature selection. The sheer volume of this dataset presents challenges in manual labeling; thus, an automatic labeling process is employed using an unsupervised ML technique from MAWILab. This method systematically classifies records as either normal or attack events, with anomalies labeled as attacks, while all other traffic is categorized as normal. This automated approach enhances scalability and reduces human bias in the labeling process. To facilitate effective analysis, feature extraction is performed using BigFlow [Viegas *et al.*, 2019], a tool designed for large-scale traffic processing. BigFlow groups events into 15-second intervals and extracts the flow-based features following Moore's approach [Moore and Zuev, 2005] (see Table 1). These features capture essential traffic characteristics, enabling robust anomaly detection and classification. By employing this dataset and processing pipeline, this study ensures a more comprehensive and realistic evaluation of intrusion detection methods.

4.2 Model Construction

The non-stationary scenarios for ML-based NIDSs impose several challenges. For evaluation, two ML strategies were considered: *batch* learning and *stream* learning. The former follows the traditional approach of training models on *batches* of data, which, while effective, incurs higher computational costs due to the need for retraining on large datasets. In contrast, stream learning focuses on continuously incorporating new instances into the existing model, allowing for incremental updates. This enables the model to adapt dynamically to evolving patterns in network traffic without requiring complete retraining, making it more suitable for real-time intrusion detection scenarios.

An ensemble of classifiers was used for the *batch learning* evaluation. Three widely used classifiers were selected as base learners: *Random Forest (RF)*, *Extremely Randomized Trees (ExT)*, and *Gaussian Naïve Bayes (GNB)*. The RF model was implemented with 100 decision trees, using the *gini* split criterion and a minimum sample split of 2. The ExT classifier was configured with the same parameters as RF to ensure consistency in tree-based learning approaches. For GNB, the model was implemented with a variance smoothing of $1\mathrm{e}{-9}$. The final ensemble combined these three classifiers using a majority-voting mechanism to enhance classification robustness. All models were implemented and evaluated using the *scikit-learn v1.6.1* API, ensuring a standardized experimental setup.

For the *stream learning* evaluation, an ensemble of three classifiers was also employed. The base learners consisted of *Hoeffding Trees (HT)*, *Adaptive Random Forests (ARF)*, and *Oza Bagging (Oza)*. The HT model was configured using information gain as the criterion for node splitting, a grace period of 200, and Naïve Bayes adaptive for leaf node prediction. The ARF classifier was evaluated with three HT as base learners, with a delta of 0.001 on its ADWIN drift detector, an information gain as split criterion, and grace period of 50. The Oza classifier was implemented using an ensemble of three HT models as base estimators. Similarly to the *batch learning* evaluation, the three classifiers were combined into an ensemble using a majority-voting mechanism to enhance predictive performance. All stream learning classifiers were implemented and evaluated using the *scikit-multiflow* $v0.5.3$ library.

The dataset exhibits an inherent class imbalance, with a higher proportion of *normal* instances than attack instances. To mitigate this issue, a replacement-based random undersampling technique was applied to each day's data, ensuring a more balanced distribution of classes. This approach helps prevent classifier bias toward the majority class and improves the model's ability to detect attacks effectively.

4.3 Changes In Network Behavior

This section focus answering three Research Question (RQ) as follows:

- **RQ1**: *How do changes in network traffic behavior affects traditional ML-based techniques?*
- **RQ2**: *How do periodic updates impact the traditional batch learning and stream learning techniques?*
- **RQ3**: *What are the computational costs involved in model updates?*

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

- *TP*: number of attack samples correctly classified as an attack.
- *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. 3.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$F\text{-}Measure = 2 \times \frac{Precision \cdot Recall}{Precision + Recall} \tag{3}$$

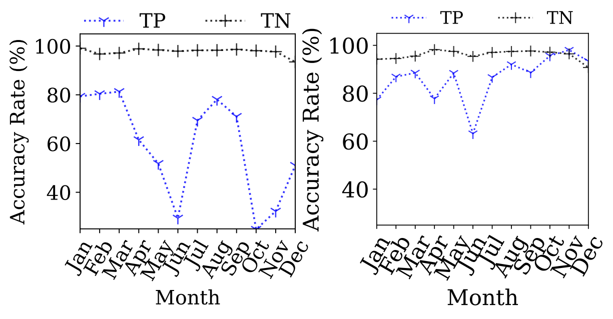
We also assess the accuracy of the selected classifiers by their Area Under the Curve (AUC) values.

$$\text{AUC} = \int_0^1 \text{TPR}(t)\, d\text{FPR}(t) \tag{4}$$

Finally, we assess the accuracy by the quotient between all the correctly predicted samples and all classified samples.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

To address the *RQ1*, an experiment was conducted to assess the accuracy performance of batch and stream learning approaches without model updates. The models were initially trained using network traffic data from *January* and subsequently tested on data from the year's remaining months. This experiment aimed to characterize the extent to which model accuracy degrades over time in the absence of updates. By exposing the models to evolving network traffic patterns without retraining, the study sought to highlight the challenges associated with deploying static intrusion detection models in dynamic environments. In particular, the experiment evaluated how shifts in traffic distribution, changes in attack strategies, and variations in normal network behavior impact the classification performance of ML-based NIDS. The findings provide valuable insights into the limitations of traditional batch learning methods and emphasize the necessity of adaptive strategies to maintain high detection performance over extended periods.



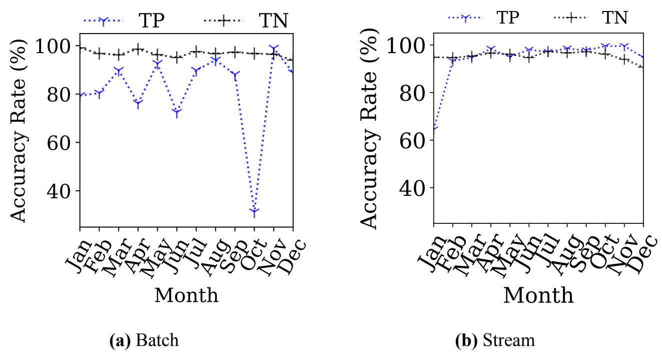
(a) Batch **(b)** Stream

Figure 1. Accuracy performance between *batch* and *stream* learning ensembles **without** recurring updates.

Figure 1 presents a comparative analysis of accuracy performance between the *batch* and *stream* learning approaches, considering the classifier ensemble. Over time, classification accuracy deteriorated, negatively impacting the models' effectiveness. As shown in Figure 1a, the *batch* ensemble experienced changes in both TP and TN rates, increasing by 0.88% and decreasing by 2.36%, respectively. In addition, the average TP and TN was 59.21% and 97.71%, respectively. The most significant drop occurred in October, where the TP rate fell to 24.77%, representing a total decline of 68.77% compared to its performance in January. Conversely, Figure 1b illustrates the accuracy trends for the *stream* ensemble. Although this approach also exhibited performance degradation over time, with the average TP and TN rates of 86.37% and 95.98%, respectively, the impact was relatively less severe than the batch approach. Regardless of the classification scheme used, the results highlight the crucial need for recurring model updates. These findings emphasize how changes in network behavior over time significantly affect the performance of ML-based NIDS, reinforcing the necessity of adaptive learning strategies for maintaining reliable intrusion detection.

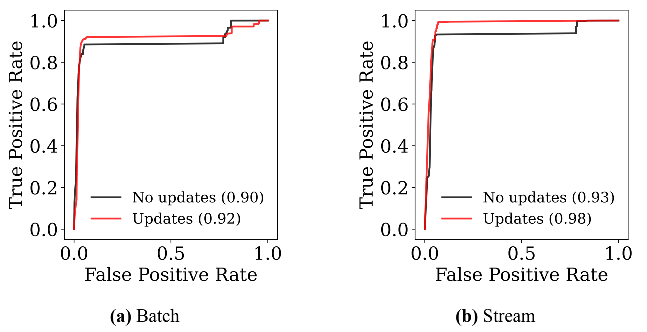
To address *RQ2*, an experiment was conducted to assess the accuracy performance of the batch and stream learning schemes, particularly when model updates are recurrent. The models were initially trained using data from January, and throughout the year, they were updated with the most recent 30 days of data. The batch and stream learning ensembles were trained from scratch for each evaluation cycle without any interference from the previous experiment. It is important to note that, in this experiment, the stream learning ensemble did not follow the typical incremental learning approach during updates, as commonly assumed in related studies. Instead, each update involved retraining the model with the new data, similar to the batch learning approach, allowing for a direct comparison between the two schemes under conditions where regular updates are applied. This setup was designed to better understand the effects of recurrent model updates on the performance of both learning paradigms and to evaluate whether stream learning can still outperform batch learning in dynamic network environments when updates are frequent.

Figure 2 illustrates the accuracy performance of both the batch and stream learning approaches, respectively. It is evi-



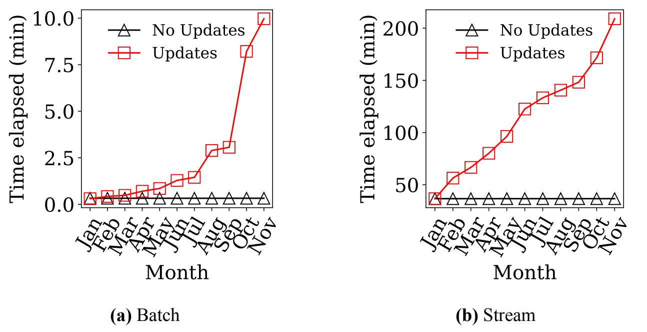
(a) Batch **(b)** Stream

Figure 2. Accuracy performance between *batch* and *stream* learning ensembles **with** recurring updates.



(a) Batch **(b)** Stream

Figure 3. ROC curve comparing performance between the *batch* and *stream* learning ensembles, both when no updates are made and when recurring updates are applied.



(a) Batch **(b)** Stream

Figure 4. Comparison of the cumulative necessary training time for *batch* and *stream* learning ensembles, both when no updates are made and when recurring updates are applied.

dent that the models' lifespan is extended throughout the year, with monthly updates, while maintaining higher accuracy levels. This trend provides strong evidence that recurrent model updates effectively adapt to the evolving network behavior. The results highlight the importance of continuous learning and model adaptation to ensure the sustained effectiveness of ML-based intrusion detection systems in dynamic environments. Specifically, Figure 2a shows the performance of the *batch* learning approach over time. The approach's average TP and TN rates were 81.81% and 96.73%, respectively. When compared to its *non-updated* counterpart, the difference in the average TP and TN rates was 27.62% and -1.00%, respectively, indicating a noticeable improvement in performance with recurrent updates mainly on the TP rate. Similarly, Figure 2b presents the accuracy of the *stream* learning approach over time. The average TP and TN rates for this approach were 94.22% and 95.33%, with differences of 9.08% and 0%, respectively, compared to the *non-updated* counterpart. Despite the worst-performing month for both approaches, the ensembles in the updated models maintained higher accuracy levels over time when compared to their respective counterparts in the previous evaluation. This further substantiates the conclusion that regular model updates are essential for adapting to changes in network behavior, enhancing the resilience and effectiveness of NIDS in dynamic environments.

To compare the results from both previous experiments, Figure 3 presents their respective Receiver Operating Characteristic (ROC) curves and AUC scores. These metrics offer a comprehensive view of the model's ability to generalize and perform effectively under the dynamic conditions of a network. The plots illustrate the performance of the *non-updated* and *updated* versions for both the batch and stream learning approaches. It is important to note that the models were trained using January data and evaluated on March data. The rationale behind this setup is that, given a one-month update delay, evaluation cannot be done in February, as there would be no available data for this purpose. The AUC score increases significantly when updates are applied. Specifically, the AUC score for the *batch* learning approach rose by 0.02 from its *non-updated* to *updated* version. Similarly, for the *stream* learning approach, the AUC score improved by 0.05. These increases further support the conclusion that model updates are crucial in effectively adapting to changes in network behavior, enhancing the overall performance of the intrusion detection system.

To address *RQ3*, Figure 4 presents the computational costs associated with training the batch and stream learning classifiers, respectively. The cumulative time (in minutes) spent training a model was compared under two conditions: *without updates* and *with updates*. Figure 4a illustrates the behavior of the batch ensemble, with training times reaching up to 0.32min when no updates were applied, while the updated counterpart had an average training time of 0.9min. Similarly, Figure 4b shows the performance of the stream ensemble, which took up to 42.56min to train a model without updates, compared to an average of 19min when updates were incorporated.

Finally, despite the extensive training, it is important to note that performance may vary depending on the underlying API implementations.

4.4 Discussion

The degradation of NIDS accuracy over time, as a result of changes in network behavior, is a well-recognized challenge in the literature. As networks evolve, the patterns of normal and malicious activities shift, causing models trained on historical data to become less effective. This section explored this issue by evaluating two prominent machine learning techniques: batch learning and stream learning. Through extensive experimentation, it was confirmed that the continuous changes in network behavior adversely impact the classification accuracy of models over time, with both techniques exhibiting a decline in performance.

To address this degradation, recurring monthly updates were introduced to the models. These updates, based on the most recent network data, proved to be an effective mechanism for mitigating the impact of shifting network behavior and maintaining the models' accuracy. By updating the models periodically, they could adapt to the evolving characteristics of the network, ensuring that they remained reliable over time. However, in practical, real-world environments, the feasibility of regular updates faces significant barriers. One of the primary challenges is the difficulty in obtaining properly labeled events, which are essential for supervised machine learning techniques. In many operational networks, event labeling can be time-consuming, error-prone, or even infeasible. Moreover, the volume of network traffic data to be analyzed and processed can be overwhelming, complicating the task of updating models regularly.

Even when updates are possible, they often come with considerable computational costs. Training a model on vast amounts of data can require significant processing power, especially when data is continually added to the system. This computational burden could become a bottleneck if updates are performed too frequently, potentially leading to detection delays and hindering the NIDS's real-time capabilities. Thus, while model updates have proven to be effective in improving accuracy, their practical implementation in production environments must account for these challenges. The balance between maintaining high accuracy through regular updates and managing the associated computational and operational costs is a key consideration for deploying ML-based intrusion detection systems in dynamic, large-scale networks.

5 Proposal

We propose a technique to mitigate the impact of network behavior changes while preserving system accuracy over time, simplifying the update process. The technique consists of two key components: *Intrusion Detection* and *Offline Updates*, as shown in Figure 5.

The Intrusion Detection component primarily focuses on classifying network events using a stream learning approach. We hypothesis that it may maintain accuracy while operating with lower computational costs, making it suitable for real-time detection in dynamic environments. Given that updates in production environments may be significantly delayed or even absent, the approach is designed to remain effective under such constraints. We introduce a confidence-based rejection mechanism to counteract accuracy degradation caused by the lack of frequent updates. Instead of blindly classifying every event, the system evaluates classification confidence using a reject option approach. Events are rejected only when the confidence value is highly uncertain, ensuring low-confidence classifications do not contribute to performance degradation. This mechanism allows the model to sustain reliable detection over time, even in scenarios where updates are sparse or impractical.

The goal of *Offline Update* is to streamline the model update process while minimizing computational overhead. This approach updates the model incrementally by leveraging the rejected instances from the previous classification task. By focusing only on these uncertain or anomalous cases, the number of network events required for retraining is significantly reduced, leading to lower storage and processing costs. Additionally, this method simplifies the labeling process for network operators. The rejected instances can be efficiently labeled using misuse-based intrusion detection techniques. This is feasible because, by the time an event is publicly disclosed through a Common Vulnerabilities and Exposures (CVE) database, for example, the necessary information for proper classification is already available. As a result, the model can be updated with accurately labeled data, ensuring improved adaptability to emerging threats without the need for continuous, resource-intensive monitoring.

The key insight of the proposed approach is that model updates are facilitated by leveraging rejected instances, those initially classified with low confidence based on the reject option mechanism. These instances are later correctly labeled by the network operator after a predefined period and subsequently used to update the model. This strategy reduces the dependency on large-scale data labeling efforts while ensuring that updates incorporate only the most relevant and uncertain cases, improving model adaptability with minimal computational overhead. Figure 5 provides an overview of the proposed scheme, illustrating how rejected instances feed into the update process.

The following subsections describe each module in detail, outlining their respective roles in intrusion detection and model updates.

5.1 Intrusion Detection

The ideal scenario for an ML-based NIDS is one that maintains high classification accuracy despite unexpected changes in network traffic behavior. This capability is particularly crucial in production environments, where the dynamic nature of network traffic makes it challenging to rely on static models. However, conventional ML-based NIDS outputs a decision to all network events, including previously unseen ones, leading to a gradual decline in accuracy over time. As a result, these detection schemes often exhibit higher error rates during real-world deployment, as shifts in network behavior create discrepancies between the model's predictions and actual traffic patterns. This highlights the need for adaptive approaches capable of mitigating accuracy degradation while maintaining computational efficiency.

To address these challenges, the classification process is structured into two key steps. First, a pool of stream learning classifiers is employed to facilitate model updates. This approach enables the model to incrementally integrate new instances, allowing an initially outdated model to operate effectively in a production environment. By continuously adapting to evolving network traffic, this method reduces the computational overhead associated with frequent retraining and large-scale updates. The second step ensures classification reliability through a reject option mechanism. Here, classifications from the ensemble are only accepted if their confidence exceeds a predefined threshold; otherwise, they are discarded. The final event label is determined through a majority voting mechanism among the accepted classifications. However, if no classifier in the ensemble confidently classifies an event, it is rejected, and an alert is suppressed. Notably, the confidence values utilized in this scheme are classifier-agnostic. For instance, the Multi-layer Perceptron (MLP) outputs probability values based on a *sigmoid* or *softmax*

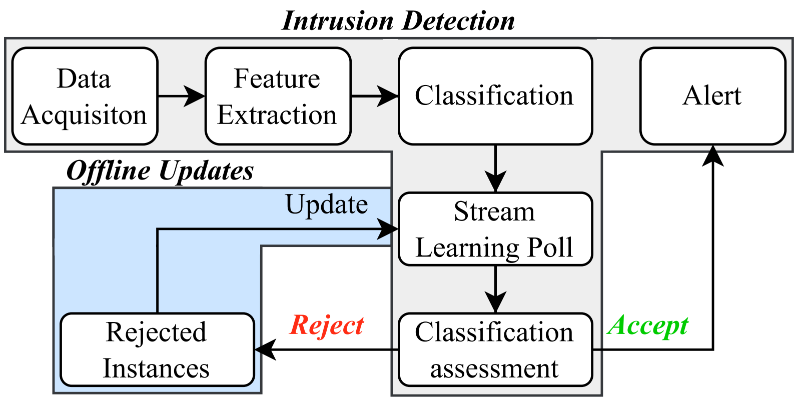


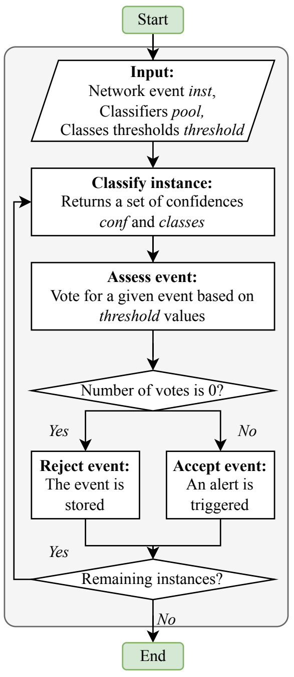
Figure 5. Proposed model

activation function. At the same time, a Random Forest classifier determines confidence scores based on the proportion of decision trees that assign a specific label to the event. This flexibility ensures that the proposed method can be seamlessly integrated with different learning models without relying on a specific classification framework.

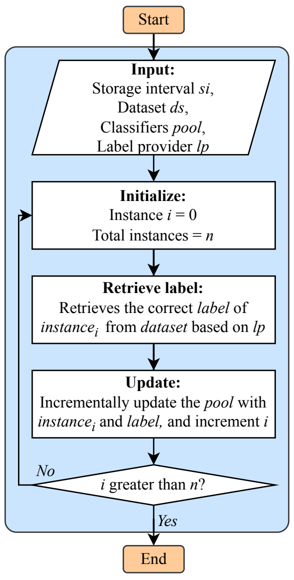
Figure 6a presents a flowchart that outlines the classification procedure in a structured manner. The process starts with an input instance (*inst*) consisting of multiple network flow features. Additionally, the system receives a *pool* of stream learning classifiers and predefined acceptance thresholds for both normal and attack classes, which are set individually for each classifier. Upon receiving an instance, all classifiers in the pool perform classification, producing their respective confidence values (*conf*). These confidence scores are then compared against the predefined acceptance thresholds (*threshold*). If the confidence score for a specific class exceeds the corresponding threshold, the *vote* procedure is triggered. This procedure follows a majority voting scheme to determine whether the event should be accepted or rejected. If the majority of classifiers accept the classification, an alert (*alert*) is generated and forwarded to the network operator for further action. However, if no classifier accepts the classification, or even the minority of the classifiers accepts, the event is rejected and stored for future model updates. This process ensures that only high-confidence classifications contribute to the system's decision-making, reducing false positives and improving adaptability over time.

The proposed classification scheme introduces a structured approach to handling network events by integrating stream learning, confidence-based rejection, and an adaptive model update mechanism. It leverages a pool of stream learning classifiers to enable incremental adaptation to evolving network traffic while reducing computational costs. The confidence-based reject option ensures that only high-confidence classifications are accepted, effectively minimizing false positives and mitigating the risks of misclassification. Additionally, rejected events, which represent uncertain or novel instances, are systematically stored and later used for incremental model updates. This strategy enhances the robustness of intrusion detection by maintaining classification accuracy over time, even in dynamic network environments where traditional batch learning methods struggle to adapt.

To define a rejection threshold, firstly the optimal points



(a) Intrusion Detection



(b) Offline Updates

Figure 6. Flowcharts illustrating the classification and updates schemes proposed.

between the average rejection rate and average accuracy rate for each classifier are selected. Secondly, the rejection threshold is selected according to the Pareto distribution, thus the optimal point between accuracy and rejection rates. Finally, the point is provided as input to the *assess event* (Figure 6a) algorithm and allows the configuration according to the needs of the network operator. Next, we detail how our proposed model deals with incremental model updates.

5.2 Offline Updates

In production environments, updating the NIDS is often impractical due to the lack of accurate labels, leaving the system vulnerable. To overcome this challenge, model updates are performed offline, using previously rejected instances (Section 5.2). This strategy reduces the number of instances requiring labeling over time (Section 4). The proposed method ensures that updates are based solely on rejected events that the outdated model in production could not reliably classify.

The rationale is that the model improves by incorporating only the previously misclassified instances. Consequently, the need for labeling, storage, and usage of instances for updates is minimized.

Furthermore, recurring updates are initiated only after a predefined time window has elapsed since the rejection of a given event (Figure 6b, *si Days Old Rejected Events*). This time window ensures sufficient time for the event to be accurately labeled using misuse-based techniques and tools. As a result, the correct event label may eventually become publicly available through cybersecurity attack databases, enhancing the reliability of the update and labeling process for production environments. This approach enables recurring updates to be applied automatically without the need for human intervention.

Figure 6b illustrates the update procedure, which is periodically triggered by the network operator (e.g., monthly). Rejected instances older than the predefined time window are retrieved from an event database that stores rejected events from the production environment. These events are then forwarded to the *Event Labeling* module, responsible for assigning the correct label to each event. Labeling is performed either with expert assistance or through using *misuse-based* techniques for autonomous labeling. Finally, the correctly labeled rejected instances serve as input for the recurring updates of the classifier pool deployed in the production environment.

Figure 6b presents the flowchart outlining the proposed update procedure. The process begins by receiving several inputs: a storage interval si, which indicates the number of days a rejected event should be stored before being considered for updates; a *dataset* containing events that are si days old; a classifier pool, which consists of multiple classifiers deployed in the production environment; and a label provider lp. The label provider is a crucial component responsible for supplying the correct label for each event in the dataset. The labeling can be accomplished through various methods, including expert assistance, misuse-based techniques, or unsupervised ML algorithms. Expert assistance involves human input to manually assign the correct label based on domain knowledge, while misuse-based techniques leverage predefined patterns of known attack behaviors to assign labels. Additionally, unsupervised ML algorithms can be employed to autonomously classify the events based on patterns observed in the data. Once labeled, the events are used to update the classifier pool, ensuring that the system continuously improves its detection capabilities by incorporating new, accurately labeled instances. The updates are carried out by requesting the event label for each instance in the dataset. Once the correct label is obtained, it is associated with the corresponding feature vector and incrementally incorporated as input for the model updates of each classifier in the pool. These updates refine the classifiers, ensuring they adapt to new data and improve their detection capabilities. After the model updates are completed, the updated classifier pool is forwarded to the *classification* module, where it replaces the outdated model currently deployed in the production environment (Figure 6a).

As a result, once all procedures are completed, the update task is greatly simplified, as only a smaller set of instances (the rejected ones) is used for updates. These rejected instances can be autonomously labeled using *misuse-based* techniques, expert assistance, or unsupervised ML algorithms, particularly since they are stored for extended periods, allowing their correct labels to become publicly available eventually. Furthermore, beyond stream learning classifiers, this approach significantly reduces the computational costs associated with storing and updating the underlying ML models. By combining stream learning classifiers with the reject option approach, our proposal becomes more practical for deployment in production environments, as it minimizes the number of instances required for updates and ensures more reliable classifications.

5.3 Discussion

Our proposed scheme significantly reduces the number of network events required for updates by strategically selecting relevant instances through classification with a reject option. This method ensures that only those instances that have the potential to improve the model are considered for updates, thereby optimizing the update process. Since events are stored for a predefined period before being used for updates, newly collected events can be labeled with greater ease, benefiting from the extended storage time. During this time, the events undergo analysis, and their correct labels become more likely to be available, either through expert assistance, misuse-based techniques, or public databases. This system of deferred labeling ensures that labeling is more accurate and that fewer instances need to be manually classified.

This approach also leads to a substantial reduction in computational costs, as only a subset of events—specifically the ones identified as potentially valuable for model updates—are used. The classifier pool is updated incrementally, with outdated models gradually improved as new, correctly labeled instances are added. As a result, the need for frequent, full-scale retraining of the model is minimized, allowing for more efficient use of resources. Furthermore, the scheme's reliance on high-confidence classifications, where only those events with strong, reliable predictions are accepted, helps maintain a stable and reasonable level of accuracy and model performance. This is especially beneficial when the models deployed in production are not always up-to-date, as it ensures that the system continues to operate effectively even when older models are in use.

In addition, our approach is designed to address the common challenges faced when updating ML-based NIDS, such as the difficulty in obtaining sufficient labeled data and the computational burden of frequent retraining. By focusing on rejected instances and labeling them in an incremental, resource-efficient manner, our scheme ensures that the model adapts over time without requiring significant manual intervention. Moreover, the incorporation of only those instances that have a high likelihood of improving the model means that the system remains robust, efficient, and adaptable to evolving network environments. Ultimately, this method supports long-term, reliable intrusion detection, making it more feasible for continuous use in production settings without compromising performance or accuracy.

6 Evaluation and Results

This section's goal is to evaluate the proposed approach, answering the following RQs:

- **RQ4**: *How does the proposed approach behave without recurring updates?*
- **RQ5**: *How does the proposed approach behave with recurring updates?*
- **RQ6**: *How does the proposed approach behave with delayed updates?*

6.1 Model construction

The proposed approach utilizes the same stream learning classifiers described in Section 4, relying on a *pool* of these classifiers. These classifiers were implemented using the *scikit-multiflow* $v0.5.3$ API, maintaining the same set of parameters. A random undersampling strategy was applied during the training phase to address the imbalanced nature of the dataset. This strategy, which performs undersampling without replacement, ensures a balanced distribution of *normal* and *attack* instances.

6.2 Intrusion Detection

The research question *RQ4* focuses on evaluating the improvement in accuracy achieved by the proposed classification scheme. The ensemble of stream learning classifiers was trained using January data and evaluated with February data. The trade-off between the rejection rate and the model's accuracy was assessed based on the class-related threshold approach [Fumera *et al.*, 2000]. Each class is associated with a specific acceptance threshold for each classifier (Figure 6a). The confidence values for each classifier were obtained through its *predict_proba* function, provided by the underlying API. The objective is to determine whether a classified event can be used for updates, thereby improving classification accuracy.

We evaluated the behavior of the proposal when no updates were applied to observe how accuracy and rejection rates evolve and how the implemented classification assessment scheme influences these metrics. Based on the set of most optimal operating points for each classifier, a 10% rejection rate was selected for the HT and Oza, and 15% for the ARF, representing the best trade-off between accuracy and error rates. It is important to note that the 10% and 15% rejection points can be adjusted according to the network operator's requirements. Higher rejection rates may result in improved accuracy, but they also increase the number of rejected events. On the other hand, if the rejection rate is set too low, accuracy may decrease over time. Since the proposed scheme relies on an ensemble of stream learning classifiers with a majority voting mechanism, an instance is rejected if two or more classifiers do not accept the classification (Figure 6a).

Figures 7a and 7b illustrate the accuracy and rejection rates over one year *without recurring updates*, respectively. The results demonstrate that the proposed scheme effectively maintains accuracy throughout the year while rejecting only a

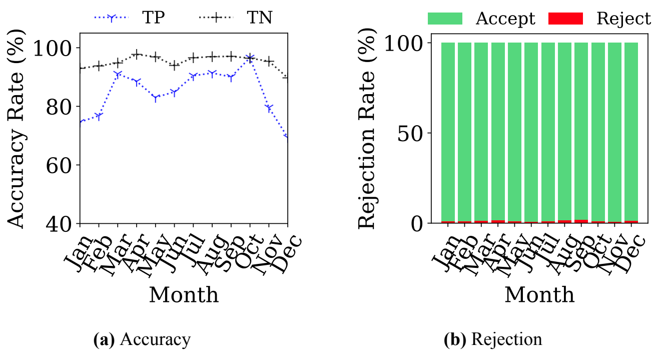


Figure 7. Proposed method's accuracy and rejection rates through a year of data *without* updates

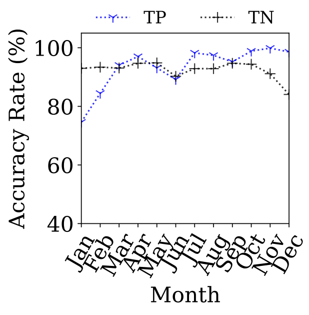
small fraction of the evaluated instances. Notably, the system's accuracy remains stable even in the absence of updates to the underlying classifiers. The proposed method achieved an average TP rate of 84.7% and an average TN rate of 95.19%, representing an difference of -1.97% and -0.82%, respectively, compared to the traditional approach without the reject-option rationale (Figure 7a vs. Figure 1b). The rejection rate averaged 1.09% across all evaluated events. The results indicate that the proposed approach successfully maintains accuracy over time.

While no significant improvement in classification performance was observed, the reject-option technique is crucial in ensuring stability, even in the absence of recurring updates. This suggests that the selective rejection of uncertain classifications helps mitigate the impact of outdated models, contributing to consistent performance throughout the year. Finally, in this research question the proposal was not evaluated under recurring updates. Instead, it mainly evaluated the proposal ability to correctly classify events even when some instances are rejected.

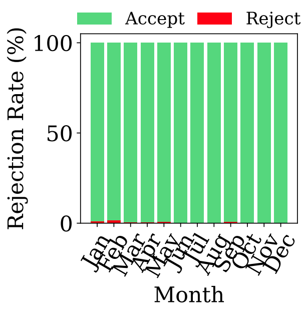
6.3 Offline updates

In response to *RQ5*, we evaluate the proposed method under conditions where recurrent updates are applied using previously rejected instances. The same rejection operation points as in the previous experiment were maintained. Rejected instances were stored and incrementally used for monthly updates, with a predefined delay of 30 days from their rejection (Figure 6b, variable si). For instance, on February 28^{th}, the underlying models were updated using rejected instances collected from January 1^{st} to 31^{th}. This approach ensures that instances are rejected and incorporated into model updates one month later. Furthermore, considering the labeling technique employed, the storage interval can be adjusted based on the network operator's requirements.

Figures 8a and 8b present the accuracy and rejection rates over one year when recurrent updates are applied using 30-day-old rejected instances. The results demonstrate that the proposed scheme, when updated recurrently, maintains its accuracy over time while rejecting only a small proportion of instances throughout the year. Compared to the previous experiment without updates, the model exhibited an increase of 93.86% and 91.81% in TP and TN rates, respectively (Figure 7a vs. Figure 8a). Additionally, the number of rejected instances decreased to 0.62%, indicating that the recurrent



(a) Accuracy



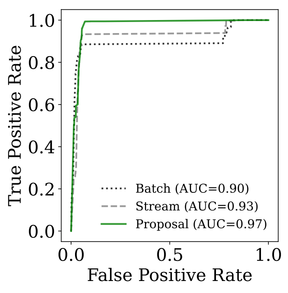
(b) Rejection

Figure 8. Proposed method's accuracy and rejection rates through a year of data *with* updates

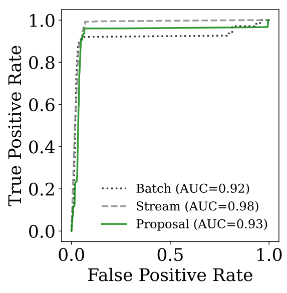
updates improved classification reliability while reducing the need for rejection (Figure 7b vs. Figure 8b). Finally, it is important to state that the TP and TN metrics are not entirely effective when comparing the proposed technique and the traditional stream and batch learning approaches. However, as the proposed scheme maintains its accuracy over time, even rejecting instances, it may be suitable for real scenarios.

Additionally, we analyzed the accuracy performance of the proposed scheme and compared it to traditional techniques, including batch and stream learning with and without periodic updates. Figure 9a presents the ROC curves, illustrating the comparison between our proposed method the batch and stream learning approaches without recurrent updates. It is important to note that the models were evaluated as the same rationale depicted in Section 4, RQ2, Figure 3. The results highlight an improvement of 0.07 in the AUC compared to the non-updated batch learning model and an increase of 0.05 compared to the updated batch learning version. These findings suggest that our approach maintains performance stability over time and enhances classification effectiveness relative to conventional learning strategies, mainly when updates are not conducted. Furthermore, Figure 9b presents the ROC curve and its analysis compared to the recurring updates scenario.

These findings emphasize the effectiveness of our proposed scheme in maintaining and even improving classification performance over time, leveraging rejected instances for updates while minimizing the impact of outdated models. The results demonstrate that the proposed approach effectively adapts to network behavior changes while maintaining classification accuracy, even when instances are rejected. By leveraging rejected instances for periodic updates, the method ensures model adaptability without requiring frequent full retraining, confirming the reliability of the proposed detection scheme. This technique also uses the incremental aspect of stream learning classifiers, supporting that the computational costs were reduced. Compared to traditional techniques, including state-of-the-art approaches, our scheme achieves comparable performance while reducing computational overhead and reliance on extensive labeled data. These findings highlight the practicality and effectiveness of the proposed method for real-world deployment in dynamic network environments.



(a) Without Updates



(b) With Updates

Figure 9. Comparison of the proposed method with traditional techniques, considering the non-updated and updated scenarios.

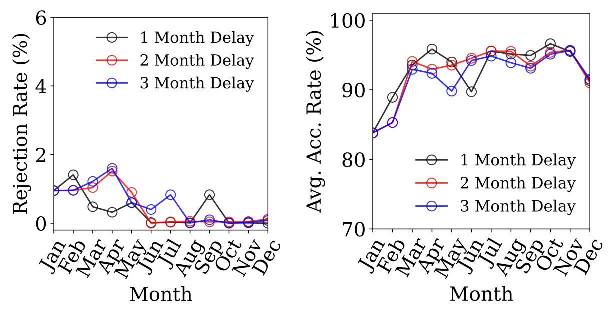
6.4 Delayed Updates

The investigation of *RQ6* aims to determine the impact of delayed updates on the performance of our proposed approach, specifically assessing how long updates can be postponed without degrading accuracy or increasing rejection rates. As demonstrated in previous experiments (Section 6.2), our method effectively maintains accuracy over an extended period, even in the absence of recurrent updates (Figure 7). However, this analysis further explores the effects of increasing the update delay on classification performance.

The analysis of update delays aims to determine the extent to which postponing updates affects classification performance. By varying the delay intervals, we evaluate how long rejected instances can be stored before being used for updates without negatively impacting accuracy and rejection rates. Figure 10 presents key performance metrics, including accuracy, the ROC curve, and rejection rates, under different update delay scenarios. The results indicate that while shorter delays generally lead to more adaptive models, our approach remains effective even when updates are deferred. This highlights the robustness of our scheme in handling evolving network conditions, providing valuable insights into balancing update frequency, computational efficiency, and classification reliability. These findings help define optimal update strategies for real-world intrusion detection systems, ensuring high detection performance with minimal computational overhead.

The results show that even with update delays of up to three months, there is minimal impact on both accuracy and rejection rates. For example, the average rejection rate for updates delayed in 30, 60, and 90 days remained stable at 0.38%, 0.47%, and 0.56%, respectively (Figure 10a). Furthermore, the difference in AUC for delays between 30 and 90 days was 32.1%, suggesting that the system maintains reliable performance even with extended delay intervals.

These findings emphasize the flexibility of the proposed approach, which allows longer update cycles without sacrificing detection capabilities. Network operators can store rejected instances for extended periods, accommodating label providers that may require additional time to supply accurate labels. This flexibility allows for easier and more efficient model updates, as rejected instances can be stored until the necessary labels are available. Importantly, this strategy does not result in significant negative impacts on accuracy or rejection rates, further enhancing the feasibility of periodic updates and reducing the need for frequent interventions. In addition, this supports the reliability of the proposed model under dy-



(a) Rejection rate **(b)** Accuracy rate

Figure 10. Rejection and accuracy rates when varying the update delay

namic network behavior, even when periodic updates are not performed.

7 Conclusion

Developing a reliable and easily updatable NIDS presents a significant challenge. Most existing approaches in the literature assume a stationary network setting, which leads to the need for periodic updates—a task that is often overlooked or inadequately addressed. The problem of model updates is exacerbated by the need for large amounts of properly labeled and evaluated network data, which is not always readily available. This paper proposes a novel NIDS scheme that simplifies the update process while maintaining high accuracy over a year of data, leveraging stream learning and reject option techniques. Stream learning facilitates the update process by allowing new instances to be incrementally incorporated into the models. The reject option, on the other hand, serves as a classification evaluator, ensuring that only the most reliable classifications are accepted based on a pre-defined acceptance threshold. This combination ensures that updates are streamlined and classification performance remains robust. Even without model updates, the proposed approach maintains consistent accuracy over a year of data. It improves true positive rates by up to 9.76% when updates are applied. Moreover, the proposed scheme demonstrates the capability to operate for up to three months without human intervention or recurrent updates, without experiencing any degradation in classification accuracy. Furthermore, the system's classification accuracy remains largely unaffected by the non-stationary nature of the network, addressing a critical challenge in NIDS development. For future works, memory consumption and federated learning scenarios should be evaluated, since the incremental nature of stream learning-based classifiers while preserving privacy in a distributed environment.

Acknowledgements

This work was partially supported by the Brazilian National Council for Scientific and Technological Development (CNPq), under grants 406603/2025-1, 307706/2025-7, 407879/2023-4, and 302937/2023-4, and by the Araucária Foundation, the Secretariat of Science, Technology and Higher Education (SETI).

Declarations

Authors' Contributions

Pedro Horchulhack contributed to the conceptualization, methodology, software, visualization, and writing of the original draft of this study. Eduardo Viegas contributed to the supervision, conceptualization, methodology, resources, visualization, writing of the original draft, as well as the review and editing of this study. Finally, Altair Santin contributed to the supervision, conceptualization, methodology, resources, and writing–review & editing of this study. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

Data can be provided on request.

References

Abbasi, A., Javed, A. R., Chakraborty, C., Nebhen, J., Zehra, W., and Jalil, Z. (2021). Elstream: An ensemble learning approach for concept drift detection in dynamic social big data stream learning. *IEEE Access*, 9:66408–66419. DOI: 10.1109/ACCESS.2021.3076264.

Abdulboriy, A. and Shin, J. S. (2024). An incremental majority voting approach for intrusion detection system based on machine learning. *IEEE Access*, 12:18972–18986. DOI: 10.1109/access.2024.3361041.

Ahmad, Z., Shahid Khan, A., Wai Shiang, C., Abdullah, J., and Ahmad, F. (2021). Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Transactions on Emerging Telecommunications Technologies*, 32(1):e4150. DOI: 10.1002/ett.4150.

Apruzzese, G., Laskov, P., Montes De Oca, E., Mallouli, W., Brdalo Rapa, L., Grammatopoulos, A. V., and Di Franco, F. (2023). The role of machine learning in cybersecurity. *Digital Threats: Research and Practice*, 4(1):1–38. DOI: 10.1145/3545574.

Apruzzese, G., Pajola, L., and Conti, M. (2022). The cross-evaluation of machine learning-based network intrusion detection systems. *IEEE Transactions on Network and Service Management*, 19(4):5152–5169. DOI: 10.1109/tnsm.2022.3157344.

Bahri, M., Bifet, A., Gama, J., Gomes, H. M., and Maniu, S. (2021). Data stream analysis: Foundations, major tasks and tools. *WIREs Data Mining and Knowledge Discovery*, 11(3). DOI: 10.1002/widm.1405.

Bouke, M. A. and Abdullah, A. (2023). An empirical study of pattern leakage impact during data preprocessing on machine learning-based intrusion detection models reliability. *Expert Systems with Applications*, 230:120715. DOI: 10.1016/j.eswa.2023.120715.

Du, J., Yang, K., Hu, Y., and Jiang, L. (2023). Nids-cnnlstm: Network intrusion detection classification model based on deep learning. *IEEE Access*, 11:24808–24821. DOI: 10.1109/access.2023.3254915.

Fumera, G., Roli, F., and Giacinto, G. (2000). Reject option with multiple thresholds. *Pattern recognition*, 33(12):2099–2101. DOI: 10.1016/s0031-3203(00)00059-5.

Kalita, D. J., Singh, V. P., and Kumar, V. (2023). A novel adaptive optimization framework for svm hyper-parameters tuning in non-stationary environment: A case study on intrusion detection system. *Expert Systems with Applications*, 213:119189. DOI: 10.1016/j.eswa.2022.119189.

Kandhro, I. A., Alanazi, S. M., Ali, F., Kehar, A., Fatima, K., Uddin, M., and Karuppayah, S. (2023). Detection of real-time malicious intrusions and attacks in iot empowered cybersecurity infrastructures. *IEEE Access*, 11:9136–9148. DOI: 10.1109/access.2023.3238664.

Kilincer, I. F., Ertam, F., and Sengur, A. (2021). Machine learning methods for cyber security intrusion detection: Datasets and comparative study. *Computer Networks*, 188:107840. DOI: 10.1016/j.comnet.2021.107840.

Liao, H.-J., Richard Lin, C.-H., Lin, Y.-C., and Tung, K.-Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24. DOI: 10.1016/j.jnca.2012.09.004.

Liu, L., Engelen, G., Lynar, T., Essam, D., and Joosen, W. (2022). Error prevalence in nids datasets: A case study on cic-ids-2017 and cse-cic-ids-2018. In *2022 IEEE Conference on Communications and Network Security (CNS)*, page 254–262, Austin, TX, USA. IEEE. DOI: 10.1109/CNS56114.2022.9947235.

Luxemburk, J. and Čejka, T. (2023). Fine-grained tls services classification with reject option. *Computer Networks*, 220:109467. DOI: 10.1016/j.comnet.2022.109467.

MAWI (2021). MAWI Working Group Traffic Archive - Samplepoint F. Available at:`https://mawi.wide.ad.jp/mawi/`.

Molina-Coronado, B., Mori, U., Mendiburu, A., and Miguel-Alonso, J. (2020). Survey of network intrusion detection methods from the perspective of the knowledge discovery in databases process. *IEEE Transactions on Network and Service Management*, 17(4):2451–2479. DOI: 10.1109/TNSM.2020.3016246.

Moore, A. W. and Zuev, D. (2005). Internet traffic classification using bayesian analysis techniques. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 50–60. DOI: 10.1145/1071690.1064220.

Nie, Z., Basumallik, S., Banerjee, P., and Srivastava, A. K. (2024). Intrusion detection in cyber-physical grid using incremental ml with adaptive moment estimation. *IEEE Transactions on Industrial Cyber-Physical Systems*, 2:206–219. DOI: 10.1109/ticps.2024.3413607.

Nisioti, A., Mylonas, A., Yoo, P. D., and Katos, V. (2018). From intrusion detection to attacker attribution: A comprehensive survey of unsupervised methods. *IEEE Communications Surveys & Tutorials*, 20(4):3369–3388. DOI: 10.1109/COMST.2018.2854724.

Olanrewaju-George, B. and Pranggono, B. (2025). Federated learning-based intrusion detection system for the internet of things using unsupervised and supervised deep learning models. *Cyber Security and Applications*, 3:100068. DOI: 10.1016/j.csa.2024.100068.

Paya, A., Arroni, S., García-Díaz, V., and Gómez, A. (2024). Apollon: A robust defense system against adversarial machine learning attacks in intrusion detection systems. *Computers &; Security*, 136:103546. DOI: 10.1016/j.cose.2023.103546.

Pereira, S. S. L. and Maia, J. E. B. (2022). Weakly supervised video anomaly detection combining deep features with shallow neural networks. *Journal of the Brazilian Computer Society*, 28(1):69–79. DOI: 10.5753/jbcs.2022.2194.

Saba, T., Rehman, A., Sadad, T., Kolivand, H., and Bahaj, S. A. (2022). Anomaly-based intrusion detection system for iot networks through deep learning model. *Computers and Electrical Engineering*, 99:107810. DOI: 10.1016/j.compeleceng.2022.107810.

Sarhan, M., Layeghy, S., and Portmann, M. (2021). Towards a standard feature set for network intrusion detection system datasets. *Mobile Networks and Applications*, 27(1):357–370. DOI: 10.1007/s11036-021-01843-0.

Sharma, B., Sharma, L., Lal, C., and Roy, S. (2023). Anomaly based network intrusion detection for iot attacks using deep learning technique. *Computers and Electrical Engineering*, 107:108626. DOI: 10.1016/j.compeleceng.2023.108626.

Talukder, M. A., Hasan, K. F., Islam, M. M., Uddin, M. A., Akhter, A., Yousuf, M. A., Alharbi, F., and Moni, M. A. (2023). A dependable hybrid machine learning model for network intrusion detection. *Journal of Information Security and Applications*, 72:103405. DOI: 10.1016/j.jisa.2022.103405.

Verkerken, M., D'hooge, L., Wauters, T., Volckaert, B., and De Turck, F. (2021). Towards model generalization for intrusion detection: Unsupervised machine learning techniques. *Journal of Network and Systems Management*, 30(1). DOI: 10.1007/s10922-021-09615-7.

Viegas, E., Santin, A., Bessani, A., and Neves, N. (2019). Bigflow: Real-time and reliable anomaly-based intrusion detection for high-speed networks. *Future Generation Computer Systems*, 93:473–485. DOI: 10.1016/j.future.2018.09.051.

Viegas, E. K., Santin, A. O., and Oliveira, L. S. (2017). Toward a reliable anomaly-based intrusion detection in real-world environments. *Computer Networks*, 127:200–216. DOI: 10.1016/j.comnet.2017.08.013.

Wang, K., Lu, J., Liu, A., and Zhang, G. (2024). Ts-dm: A time segmentation-based data stream learning method for concept drift adaptation. *IEEE Transactions on Cybernetics*, 54(10):6000–6011. DOI: 10.1109/TCYB.2024.3429459.

Xu, X., Zhang, X., Zhang, Q., Wang, Y., Adebisi, B., Ohtsuki, T., Sari, H., and Gui, G. (2024). Advancing malware detection in network traffic with self-paced class incremental learning. *IEEE Internet of Things Journal*, 11(12):21816–21826. DOI: 10.1109/jiot.2024.3376635.