Network Intrusion Detection System Based on Reinforcement Learning Technique Optimization

Sukkarin Ruensukont[1 [0009−0008−7815−9907]],
Karin Sumonkayothin[1 [0000−0001−6098−6228]],
Prarinya Siritanawan[2 [0000−0002−9023−3208]],
Narit Hnoohom[1 [0000−0003−3714−8482]], Setthawhut Saennam[3], and
Razvan Beuran[4 [0000−0002−4109−3763]]

[1] Department of Computer Engineering, Faculty of Engineering, Mahidol University, Nakhon Pathom 73170, Thailand
[2] Graduate School of Science and Technology, Shinshu University, Nagano City, Nagano 380-8553, Japan
[3] MFEC Public Company Limited, Bangkok 10900, Thailand
[4] Japan Advanced Institute of Science and Technology, Nomi, Ishikawa 923-1211, Japan

Abstract. With the increasing role of Machine Learning (ML) and Deep Learning (DL) in various domains, their application in enhancing Network Intrusion Detection Systems (NIDS) has gained significant attention. Traditional NIDS approaches often rely on correlation-based detection, which may lead to misleading or fake correlations, failing to align with real-world use cases. Addressing this issue requires additional features, new datasets, and the development of new solutions. However, the rapid advancements in ML and DL pose challenges for timely deployment, as training, testing, and evaluating new models against existing solutions can be time-consuming. The large size of real-world datasets also contributes to high computational costs and extended training times, limiting the practical use of ML-based NIDS in dynamic environments.

To tackle these challenges, this paper contributes to the field of NIDS in three key aspects: employing Reinforcement Learning (RL) to accelerate and optimize the model tuning process; introducing an efficient data preprocessing pipeline specifically designed for NIDS, which enhances data quality and feature representation; and proposing a novel sampling strategy that determines an optimal dataset size both in terms of total records and class-level balance. By integrating model tuning with the proposed method on dataset sampling, this research uses a smaller sampling size of 3,898 records and achieves a higher F1 score of 93.20, compared to the state-of-the-art statistical sampling method on the same NIDS dataset.

Keywords: Sampling · Real World · Optimization · Network Intrusion Detection System · Reinforcement Learning

1 Introduction

Network Intrusion Detection Systems (NIDS) have a significant role in cybersecurity. They monitor network traffic to detect malicious packets. These systems analyze network packets in real-time, identifying anomalies and patterns associated with cyber threats. Halimaa [1] used Support Vector Machine and Naïve Bayes, and Roshan [20] used Deep Learning (DL) to assist in developing a NIDS. Effective NIDS must maintain high accuracy while minimizing false positives to ensure reliable security. However, given the complexity and volume of network traffic, optimizing NIDS remains a challenging task. Existing research often focuses on training NIDS models using full-size datasets, which is time-consuming, especially with real-world data that is both large in size and high in dimensionality. We have explored the use of RL in NIDS through various studies, as detailed by Dang and Vo [5], Han et al. [8], Yang et al. [26], Li et al. [11], Ren et al. [19], Benaddi et al. [3], Lopez-Martin et al. [12], Dong et al. [6], Sethi et al. [21], Hsu and Matsuoka [9], Sethi et al. [22], Mouyart et al. [14], and Ren et al. [18]. While these works provide valuable perspectives on enhancing RL techniques, they often overlook the practical challenges of applying RL in real-world settings—particularly the impact of large dataset sizes on training time and computational demands. Although comprehensive

learning improves detection capability, the associated resource costs make real-time deployment difficult in operational environments.

To improve the practicality of deploying ML and DL models in NIDS, making them more adaptable to real-world constraints, we propose the three key contributions to NIDS:

Leveraging RL to accelerate hyperparameter optimization and reduce training time Wang et al. [25] and Powell et al. [16] contributed on optimization of the large scale problems. The iterative search process tends to be extremely time-consuming, making it impractical for real-time applications. In contrast, RL has demonstrated superior performance in solving optimization problems. Given the real-time requirements and the large volume of data involved in Network Intrusion Detection, RL is particularly well-suited for our case. RL has been effectively applied to optimize NIDS across three main areas: feature selection, hyperparameter tuning, and algorithmic improvements. For feature selection, Robinson et al. [17] utilized correlation-based and information gain-based methods to reduce dimensionality and enhance detection of minority attacks, achieving better accuracy and lower false positives on datasets like CIC-IDS2017 and NSL-KDD. In terms of hyperparameter tuning, Masum et al. [13] employed Bayesian optimization to automate parameter selection, significantly boosting model performance on the NSL-KDD dataset. At the algorithmic level, Vembu and Dhanapal [24] proposed using the Whale Optimization Algorithm to fine-tune CNNs, outperforming conventional models such as DNN, RF, and DT in both detection accuracy and efficiency, especially in Wireless Sensor Network environments. Building on these findings, our work begins by applying RL specifically for hyperparameter optimization to accelerate model training while maintaining high performance, laying the groundwork for further enhancements in NIDS.

Tailored data preprocessing pipeline to enhance data quality for NIDS taks Our propose data preprocessing approach stands out from previous work by applying well-known techniques in a way that is specifically adapted to the security context of NIDS, an area where such methods are rarely utilized. For example, we use hashing mechanisms to identify and track duplicate or equivalent network events. This is particularly valuable in security settings where identical patterns may occur across different sessions but must be recognized as the same attack signature. We handling port numbers that are numerically close but semantically distinct. Potentially grouping ports like 22 and 53 together due to their proximity. However, in a security context, these represent entirely different services and threat profiles.

Designing an efficient sampling strategy to determine optimal dataset sizes—both in overall volume and per class distribution to balance training efficiency and model performance Sampling strategy differs from existing approaches in several important ways. While sampling techniques are generally well known, their application within the context of NIDS remains limited. Most existing NIDS research still relies on full datasets, which are often very large and high-dimensional, making model training time-consuming and resource-intensive.

Although there is extensive research on sampling methods in general, studies specifically focusing on NIDS are relatively limited. We highlight related work with an emphasis on practical and real-world applications. Previous work on sampling for NIDS by Alikhanov [2] studied the effect of traffic sampling on machine learning-based NIDS approaches, focusing on sampling at the flow level, which occurs before feature extraction. However, we perform sampling at the feature level, allowing better control and avoiding the need to reprocess raw traffic data. Sampling at the flow level presents challenges: it is difficult to determine the optimal sampling ratio beforehand, and since full feature extraction is required regardless, this effectively doubles the preprocessing effort.

Another study of sampling on NIDS by Kabir et al. [10] applied sampling on the KDD 99 dataset, which is another widely used dataset for NIDS. They used a statistical approach to determine the sample size, then clustered the data based on similarity and selected samples only from dense clusters, excluding data points that were distant from these clusters even if they were not true outliers.

Another relevant study by Han et al. [8] focused on accelerating hyperparameter tuning on the CIC-IDS2017 dataset using Proximal Policy Optimization (PPO). While their work successfully addressed the time-consuming nature of tuning, they did not report the actual tuning time, which limits direct comparability. Moreover, their tuning process was performed entirely on the full dataset, making it computationally expensive.

In this work, we introduce a novel sampling strategy that addresses these limitations. Unlike Kabir et al. [10]'s approach, which trained both the tuned and final models solely on sampled subsets, our method

preserves the class distribution and overall data representation in the sampled subsets. Importantly, the optimized hyperparameters obtained from these representative samples are subsequently applied to train the final model on the full dataset, ensuring both computational efficiency during tuning and comprehensive learning in the final model. This dual-phase approach enables us to achieve substantially higher performance with significantly reduced sampling sizes, while also avoiding the heavy computational cost of full-size tuning thereby balancing tuning efficiency, data representativeness, and predictive accuracy.

Moreover, common sampling methods do not address the class imbalance often seen in NIDS, resulting in rare attack types being underrepresented or entirely missing. Our method overcomes these limitations by preserving class distributions during sampling and ensuring sufficient representation of minority classes.

Another key distinction is in the timing and purpose of sampling. We apply sampling only during the hyperparameter tuning phase, not during the final training. This means our model is ultimately trained on the full dataset, ensuring maximum performance and reliability. In contrast, other works that sample before feature extraction or model tuning cannot guarantee that the tuned parameters will perform well on the full data. Finally, our approach avoids the practical drawbacks of low-level sampling, which may require access to network equipment or flow generators.

We propose a method for training NIDS that leverages three key techniques: RL for efficient hyperparameter optimization, tailored data preprocessing specifically designed for NIDS, and a sampling technique for NIDS datasets. Together, these contributions enable fast model training that keeps pace with emerging attacks and effectively handles the vast volume of network data.

2 Proposed Method

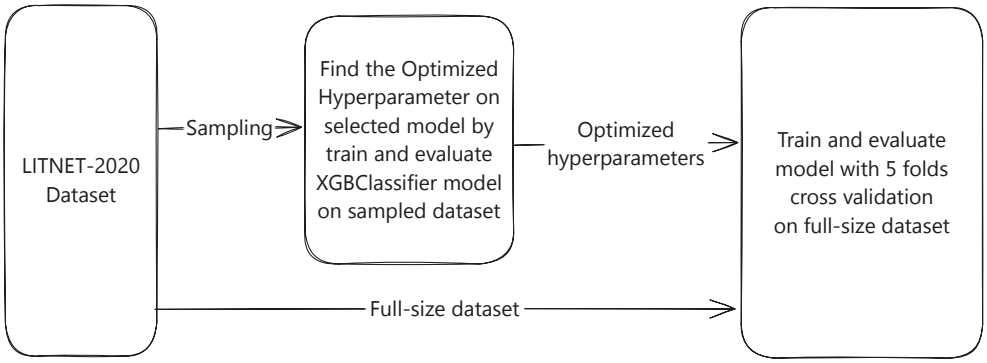

Fig. 1: Reinforcement Learning Method on Different Sampling Sizes

The rapid advancements in ML and DL pose challenges for timely deployment, as training, testing, and evaluating new models against existing solutions can be time-consuming. The large size of real-world datasets further contributes to high computational costs and extended training times, limiting the practical use of ML-based NIDS in dynamic environments. To address these challenges, Fig. 1 shows the proposed method consisting of two key strategies to improve model training efficiency for NIDS. First, we employ RL techniques for hyperparameter optimization to accelerate model training. Second, we introduce a structured sampling strategy to effectively manage large datasets, selecting an optimal dataset size that balances training efficiency and model performance.

By integrating RL-based hyperparameter tuning and an optimized dataset sampling approach, this develop the more practical ML/DL models for NIDS. These improvements enhance timely deployment and better handling of real-world data.

2.1 Dataset Preparation

We propose a specific method for preparing NIDS datasets to align with the real-world characteristics we aim to represent. This method consists of 9 steps that must be completed before utilizing the dataset.

- **Removing Redundant Features Using Correlation Matrix:** We analyze the correlation matrix to examine the relationships between features and remove one feature from each pair with a high positive or negative correlation. For example, features such as *start time* and *end time* often exhibit high correlation. This dimensionality reduction aligns with our objective of handling large-scale real-world data. However, reducing dimensions does not always lead to a decrease in model training time, as in the case of Decision Trees, where the conditions in tree nodes remain unchanged.
- **Add Duration Feature, Then Remove Start And End Time:** Several NIDS datasets, such as LITNET-2020 [4], specify the start and end times of traffic flows. However, in the context of security, the exact start and end times are often not relevant. Therefore, these features can be removed. Instead, *duration* is a crucial factor that helps distinguish between different attack classes. We compute the duration feature as follows:

$$\text{Duration} = \text{End Time} - \text{Start Time} \tag{1}$$

  This newly added feature provides meaningful information while reducing redundant data.
- **Adding Source File Label:** In the case that dataset files are separated per class. A new column, 'source_file', is introduced to retain the original file information for each data point. This enables us to trace back the source of each record and analyze its relationships with it's neighbors.
- **Removing Attack Identification Features:** Certain features specifically related to attack identification are removed to prevent data leakage. This ensures that the model generalizes well rather than relying on predefined attack characteristics.
- **Convert Non-Numeric Data to Numeric Data:** Many features in NIDS datasets are not in numeric form, such as protocol names (e.g., TCP, HTTP, DNS), IP addresses in both IPv4 format (xxx.xxx.xxx.xxx) and IPv6 format, TCP flags represented as strings, MAC addresses, and other non-numeric values. These values are first hashed into strings. Then, we use the `int` function to convert the string into a base-10 number and finally into a decimal number. The result is that if the original string value is the same, the hash value will also be the same, and a consistent numeric float value will be obtained. The conversion from non-numeric to numeric does not apply to the class label.
- **Removing Features with Identical Values:** Features where all values are identical across all records are removed. These features do not contribute to the model's learning process and cannot help in distinguishing between class labels.
- **Normalization with MinMaxScaler:** To standardize feature values, MinMaxScaler is applied to all features except for the *source port*, *destination port*, and *class label*. The normalization process is defined by equation (4):

$$v' = \frac{v - \min(v)}{\max(v) - \min(v)} \tag{2}$$

  Where:
  - v is the value of the feature before normalize.
  - $\min(v)$ is the minimum value of target feature.
  - $\max(v)$ is the maximum value of target feature.
  - v' is the value after normalized in a range [0, 1]

  This scaling process transforms all numeric features to a range between 0 and 1. Standardization helps prevent any single feature from dominating due to differing magnitudes and allows models to converge more efficiently.
- **Handling Port Features as Categorical Data:** The *source port* and *destination port* are already numerical. If we apply MinMaxScaler normalization, the values of well-known ports (0-1024) will be very similar after normalization, making it difficult for the model to differentiate between ports. In the context of NIDS, these well-known ports have significant differences. For example, port 22 is well-known for SSH, and port 53 is well-known for DNS. Therefore, we handle port numbers as categorical data using the One-hot encoder. However, since the port number range is from [0, 65535], using a One-hot

encoder would result in 65,536 new features. Knowing that the port number is represented in 16 bits, we instead apply a Binary One-hot encoder, which converts the port number into 16 new features. After the conversion, the original *source port* and *destination port* features are removed. This method enables the model to better distinguish between well-known ports during the training process. This distinction is important because port numbers represent discrete categories rather than continuous values.

– **Convert labels to numbers:** NIDS datasets often have class or label values as strings. For example, LITNET-2020 [4] has the class value in *attack_t* as 'none' for benign and 'icmp_smf' or 'udp_f' for the names of Attack classes. We will convert the benign class value to 0 and convert the values of other malicious classes to numbers 1, 2, . . . , n, where n is the total number of classes. During this conversion step, we map the class numbers to the class names and store this mapping in a file, so that the class name can be identified from the converted class number later.

2.2 Sampling Strategy

Instead of using the full-size dataset, which can be computationally expensive and time-consuming, the study introduces a sampling strategy to determine an optimal dataset size that can provide sufficiently good hyperparameters within a limited time. This approach balances the trade-off between time efficiency and model performance in hyperparameter tuning. The obtained hyperparameters are then used to train the model on the full-size dataset. This makes the development and training of ML and DL models more practical for real-world applications.

Table 1: Comparison of Sampling Methods with Key Characteristics

Sampling Method	Unbiased	Easy to Implement	Subgroup Representation	Cost-Effective	Generalizable
Systematic Sampling	✗	✓	✗	✓	✓
Cluster Sampling	✗	✓	✗	✓	✗
Convenience Sampling	✗	✓	✗	✓	✗
Quota Sampling	✗	✓	✓	✓	✗
Snowball Sampling	✗	✓	✗	✓	✗
Simple Random Sampling	✓	✓	✗	✗	✓
Stratified Sampling	✓	✗	✓	✗	✓
Statistical & Selected Cluster [10]	✗	✗	✓	✗	✗
Random & Preserved Min per Class	✓	✓	✓	✓	✓

We have compared various sampling methods as summarized in Table 1 The first five rows present common sampling techniques with their distinct advantages and drawbacks. Systematic sampling is easy to implement and cost-effective but may introduce bias if the population contains a hidden pattern. Cluster sampling is efficient for large or geographically spread populations but tends to be biased and less generalizable. Convenience sampling is the simplest and cheapest method but suffers from high sampling bias and poor representativeness. Quota sampling improves subgroup representation compared to convenience sampling, yet it is still subjective and not fully generalizable. Snowball sampling works well for accessing hidden or hard-to-reach populations but lacks unbiasedness and generalizability.

The last four methods in Table 1 provide more rigorous and balanced sampling approaches. Simple random sampling ensures unbiasedness and generalizability but can be costly and may not guarantee balanced subgroup representation. Stratified sampling enhances subgroup representation and generalizability by dividing the population into strata, although it is more complex and expensive. Statistical & selected cluster sampling by Kabir et al. [10] attempts to control bias through selective cluster choice but remains difficult to implement, costly, and not fully generalizable. Our proposed method, Random & Preserved Minimum Sampling per Class, effectively combines true randomness with guaranteed minimum subgroup representation, making it easy to implement, cost-effective, unbiased, and generalizable. This method also resolves the issues of class imbalance by ensuring a minimum sample from each class, addressing both data fairness and the bias caused by unequal class distributions. It is particularly beneficial in situations like Network Intrusion Detection (NID), where certain classes (e.g., attacks) are underrepresented.

Random sampling results in a reduction in the number of records for each class according to the sampling fraction. In NIDS, it is common for the number of Benign records to be significantly higher than Malicious

records. Thus, applying fraction-based sampling does not significantly impact the number of Benign records but greatly affects the number of Malicious records, which are already scarce.

To address this issue, we propose a method to *preserve the number of attack records per class*. First, we determine the desired sampling size as:

$$\text{Sampling size} = \text{Full-size dataset length} \times \text{Fraction} \tag{3}$$

Next, we distribute this sampling size among all classes by computing the expected number of records per class (E_c):

$$E_c = \frac{\text{Sampling size}}{\text{Number of classes}} \tag{4}$$

If any class has an initial count lower than this expected value, we preserve the number of attack records per class to ensure that Malicious records remain sufficient for the RL process to optimize hyperparameters. Meanwhile, the remaining records are filled with Benign records to match the desired sampling size. If a class has an initial count exceeding the expected sampling size per class, we apply random sampling to obtain the required number of records.

$$N_c = \min\left(O_c, E_c\right) \tag{5}$$

where N_c is the final number of records selected for class c, O_c represents the number of records in class c in the original dataset, and E_c is the expected number of records per class after sampling.

2.3 Fine-Tuning Hyperparameters with RL Optimization on Sampled Datasets

The next step is to determine the optimal sampling size of the sampled dataset that achieves the best F1 score when training the model using RL Optimization. The obtained hyperparameters will then be used for further applications. In this stage, we start by selecting an appropriate model, which should align with the characteristics of the NIDS dataset, which consists of multiclass labels and categorical features. These characteristics influence model selection, as certain algorithms handle categorical data and multiclass classification more effectively.

After selecting the model, in this stage, we choose the hyperparameters to be tuned, identifying whether they are of type float, integer, or categorical, as well as defining their possible value ranges. We then apply a RL optimization technique to fine-tune the selected hyperparameters, aiming to maximize the F1-score. For RL optimization, as illustrated in Fig. 2, we employ a Markov Decision Process (MDP), in which the environment, decision process, reward function, and initial state are defined as follows.

Markov Decision Process: A Markov Decision Process is used to aid decision-making. An MDP consists of states, actions, transitions, and rewards. Then finding a policy that maximizes the expected cumulative reward, which, in this case, is the F1 score, by selecting actions that lead to the best long-term outcomes. The agent aims to choose a policy $\pi : S \rightarrow A$ that maximizes the expected return $J(\pi)$.

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t) \right] \tag{6}$$

where:

- $J(\pi)$: Expected return (cumulative expected reward under policy π).
- $\mathbb{E}_\pi[\cdot]$: Expectation over the trajectories generated by policy π.
- $\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t)$: Sum of discounted rewards over time.
- $\gamma \in [0, 1]$: Discount factor, determining the importance of future rewards.
- $R(S_t, A_t)$: Reward function as defined in equation (2), representing the expected immediate reward when taking action A_t in state S_t.
- S_t : State at time step t.
- A_t : Action taken at time step t following policy π.

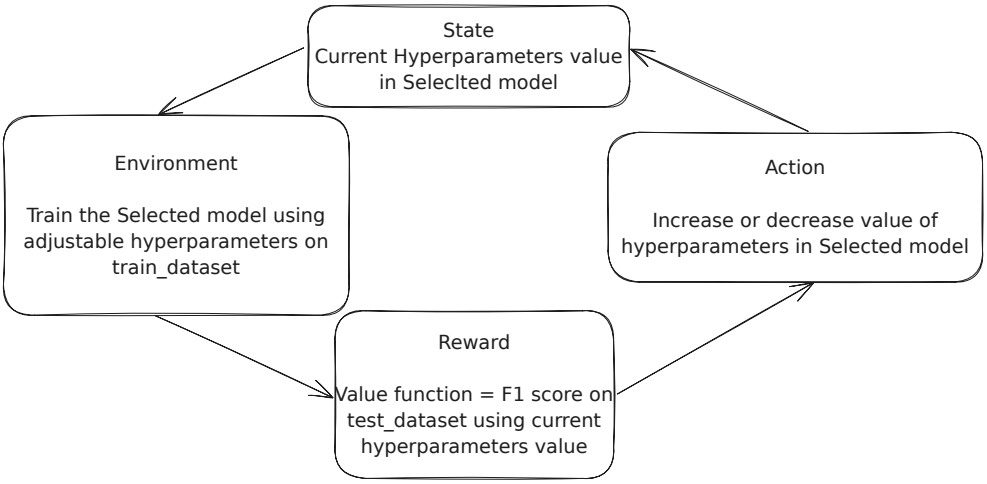

Fig. 2: Reinforcement Learning Technique Optimization Process

The Decision Process: At each time step t, the agent is in a state $s_t \in S$. The agent then takes an action $a_t \in A$, which leads to a new state $s_{t+1} \in S$ according to the transition probability $P(s_{t+1}|s_t, a_t)$. The agent receives a reward $r_t = R(s_t, a_t, s_{t+1})$ for the transition.

Reward function We define reward function after taking action in each state as follow.

$$R(s, a) = \mathbb{E}\left[r_t \mid S_t = s, A_t = a\right] \tag{7}$$

- $R(s, a)$: The reward function, which gives the expected reward when in state s and taking action a.
- $\mathbb{E}$: Expected value from action to reward.
- r_t : The reward received at time step t. The reward is calculated from F1 score of model and hyperparameter on test dataset from action taken in s state
- S_t : The state of the system at time step t. This is current model and hyperparameter's value.
- A_t : The action taken at time step t. The action either change the model or modify hyperparameter value.

Initial State and Hyperparameter Range: Define the initial state and the range of each hyperparameter, then start RL Optimization. If the agent takes an action that would cause a hyperparameter value to exceed its defined range, the value remains unchanged. This ensures that all hyperparameters stay within their specified limits throughout the optimization process.

Reinforcement Learning Process for Hyperparameter Tuning: At each time step t, the agent is in state s_t. The agent then selects an action a_t, which involves adjusting one or many of the hyperparameters by `neighbor_step` (initially set to 1). The environment, represented by the intrusion detection system, processes the action, updates the state to s_{t+1}, and calculates the reward r_t, which reflects the F1 score achieved by the model after training with the new hyperparameter values. The agent receives this feedback and updates its knowledge, continuing the process to optimize the hyperparameters over time.

3 Experimental Setup

3.1 Dataset Selection and Preparation

We selected LITNET-2020 [4] as the main dataset for our experiment because it is the very recent NIDS dataset, created in 2020, compared to popular datasets like CIC-IDS2017 [23] (created in 2017) and KDD-99 [15] (created in 1999). Additionally, LITNET-2020 has the largest number of records, with a total of

35,196,460 records, consisting of 1 benign and 12 malicious classes, for a total of 13 classes. It was collected over a period of 9 months, which aligns with the approach we propose for training models with large datasets that represent real-world scenarios.

LITNET-2020 does not include headers by default, so we had to manually add the headers, which were sourced from GitHub-Grigaliunas [7], where the LITNET-2020 dataset is published. We then preprocessed the dataset according to the steps outlined in Section 2.3, Dataset Preparation.

In addition to LITNET-2020, we also conducted experiments on CIC-IDS2017, which is a popular NIDS dataset, and on KDD-99 to compare the results with the state-of-the-art sampling approach for NIDS by Kabir [10], which was tested on KDD-99. We preprocessed both datasets in the same way, following the procedures outlined in Section 2.3. In CIC-IDS2017, there are occurrences of positive and negative infinite values, so we replaced positive infinity with the maximum value of the respective feature and negative infinity with the minimum value of the respective feature.

LITNET-2020, CIC-IDS2017, and KDD-99 datasets have been preprocessed such that all feature values are in float format and the labels are numerical, making them ready for model training.

Fake Correlation Consideration As previously mentioned in the Introduction, the issue of fake correlation and its mitigation must be addressed during the dataset acquisition stage. In traditional network intrusion detection—whether volume-based or signature-based approaches—before the adoption of AI techniques, detection was carried out by defining and identifying specific characteristics unique to each type of attack.

Upon examining the LITNET-2020 dataset, along with other widely used intrusion detection datasets such as CIC-IDS 2017 and KDD-99, we observed that many of the attack classes defined within these datasets lack sufficient features or details to conclusively identify the nature of the attacks. Although this work does not delve into the specifics of these limitations, our observations are grounded in well-known methodologies and practical experience in the field of network intrusion detection.

Training machine learning models to classify attack types based on such incomplete features essentially results in learning correlations between features and labels. However, if the features are not sufficient to accurately define the classes, any such correlation—regardless of its statistical significance—can be considered a fake correlation.

Therefore, this study does not aim to improve classification accuracy by refining class-feature definitions or by modifying the dataset. Instead, acknowledging the practical constraints of real-world applications, we adopt existing, widely accepted NIDS datasets, LITNET-2020, CIC-IDS 2017 and KDD-99 [4,15,23], treating them as sufficiently usable in their current form. The focus of this work is to explore sampling strategies that enhance model training efficiency, rather than to address dataset accuracy or redefine the ground truth.

3.2 Experimental Method

There are two steps in the experimental method. First, we use RL to find the optimal hyperparameter for each sampling rate. Then, we use only the selected hyperparameter to train and evaluate a new model from scratch using cross-validation on the full-size dataset.

Step 1: Reinforcement learning for optimal hyperparameter from each sampling rate

Sampling: After completing the preprocessing of the full-size dataset, we perform sampling based on a fixed number of records per class, as determined by Equation 5. The sampling fractions used are as follows: 0.1, 0.01, 0.001, 0.0001, 0.00009, 0.00008, 0.00007, 0.00006, 0.00005, 0.00004, 0.00003, 0.00002, and 0.00001, respectively. Each fraction differs by a factor of 10, except in the range of 0.0001 to 0.00001, where additional finer-grained samplings were performed at 0.0001, 0.00009,..., 0.00002 and 0.00001. The reason for this finer subdivision is that we observed a significant change in the F1-score between the fractions 0.0001 and 0.00001. Thus, we introduced more granular fractions in this range to analyze the variations in greater detail. The sampling method used was random sampling with `random_state` set to 42 to ensure consistent experimental results across different runs.

The number of records for each sampling size, except for the finer-grained range between 0.0001 and 0.00001, as well as the distribution of different classes, can be observed in Table 2.

Table 2: Sampling Data Distribution (Number of records) of LITNET-2020

	Full-size dataset	**Fraction 0.1**	**Fraction 0.01**	**Fraction 0.001**	**Fraction 0.0001**	**Fraction 0.00001**
Sampling Size	35196460	3519646	351964	35196	3519	351
Number of Classes	13	13	13	13	13	13
Records in Class 0	32087753	2785414	149882	8428	270	27
Records in Class 1	59479	59479	27874	2787	270	27
Records in Class 2	11628	11628	11628	2787	270	27
Records in Class 3	93583	93583	27874	2707	270	27
Records in Class 4	1580016	278742	27874	2787	270	27
Records in Class 5	22959	22959	22959	2787	270	27
Records in Class 6	52417	52417	27874	2787	270	27
Records in Class 7	24291	24291	24291	2787	270	27
Records in Class 8	1255702	278742	27874	2787	270	27
Records in Class 9	747	747	747	747	270	27
Records in Class 10	1176	1176	1176	1176	270	27
Records in Class 11	6232	6232	6232	2787	270	27
Records in Class 12	477	477	477	477	270	27

Train-Test Splitting: The next step involves performing a stratified random split of each sample into 70% for training and 30% for testing, ensuring the class distribution remains consistent with the original dataset in both subsets.

Model Selection: XGBClassifier was chosen because it balanced a high F1 score of 99.81 with an efficient training time of just 11 minutes, outperforming other models. SVM, despite achieving a high F1 score of 90.12, required an impractical 49 hours for training. Logistic Regression struggled with multiclass classification, while CatBoost, though accurate, had significantly longer training times. K-Nearest Neighbor (KNN) was inefficient on large datasets, and LightGBM underperformed compared to XGBClassifier. Given these factors, XGBClassifier proved to be the most suitable model for our experiment.

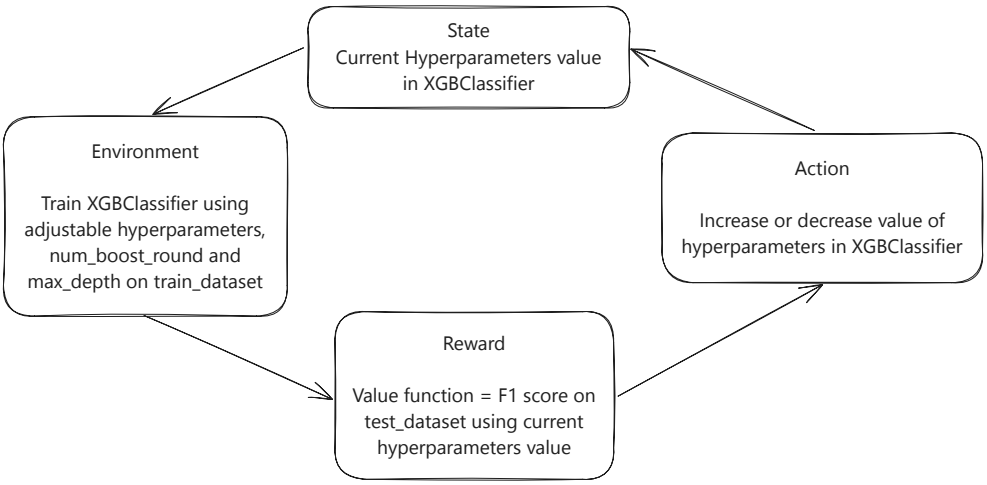

Fig. 3: Reinforcement Learning Technique Optimization Process for XGBClassifier

Fine-Tuning Hyperparameters with RL Optimization on Sampled Datasets: RL Technique Optimization is used to fine-tune the XGBClassifier model as illustrated in Fig. 3. The process begins by defining an initial state, and in each iteration, an action is performed to adjust various parameters, leading to the next state. The RL process can be constrained using different limitations, such as a time limit, a fixed number of iterations, or a restriction on the number of visited states. In this case, the training limit is set based on time. We define the Environment, State, Action, and Reward function for the RL optimization technique as shown in Table 3.

At this point, we have obtained the hyperparameters num_boost_round and max_depth for each sampling size. In the next step, we will evaluate these values on the full-size dataset.

Table 3: Details of Reinforcement Learning Setup

Environment	Use the XGBClassifier model to train on the training dataset, then evaluate the value function on the test dataset.
State Definition	Each state consists of different values for the hyperparameters: `num_boost_round`, ranging from 2 to 5000, and `max_depth`, ranging from 1 to 100. The total number of possible states is: $(5000 - 2 + 1) \times (100 - 1 + 1) = 4999 \times 100 = 499{,}900$ possible states The initial state is: `num_boost_round` = 10 and `max_depth` = 2.
Agent Actions	The agent can perform four possible actions: – Increase `num_boost_round` by `neighbor_add_step`. – Decrease `num_boost_round` by `neighbor_add_step`. – Increase `max_depth` by `neighbor_add_step`. – Decrease `max_depth` by `neighbor_add_step`. For example, if the current state is [10, 2], the neighboring states are: [11, 2], [9, 2], [10, 3], and [10, 1]. The value function is evaluated for each of these. If the current `neighbor_add_step` does not lead to a new state, we increase it by 1 to allow reaching further states.
Reward Function	The reward is the F1 score obtained by training the XGBClassifier with the given `num_boost_round` and `max_depth` on the training dataset and evaluating on the test dataset.

Step 2: Performance evaluation on model trained by different hyperparameter from different sampling size in step 1 This step performs Stratified K-Fold cross-validation, which ensures balanced class distributions across folds, on the full-size dataset using the values of num_boost_round and max_depth obtained through Reinforcement Learning on each sampling size in the previous step as hyperparameter values for the XGBClassifier model. In this process, we set $K = 5$. The procedure is outlined in Algorithm 1.

The evaluation strategy using both the **macro F1 score** and **per-class recall** ensures that the model is robust and performs well across all classes, including those that are underrepresented. This comprehensive evaluation is particularly essential for real-world cybersecurity, where the volume of attacks is often much lower than that of benign instances. It is crucial to accurately measure the model's ability to detect each attack class, ensuring that even rare or less frequent attack types are properly identified and not overlooked.

Algorithm 1 K-Fold validation on full-size dataset

Input: Preprocessed datasets from CSV files
Output: Average F1 score (weight=macro) and recall per class on full-size dataset

procedure LOAD AND PREPARE DATA
Load CSV files into dataframes
Concatenate all dataframes into a single dataset
Split dataset into features (X) and labels (y)
end procedure

procedure TRAIN AND TEST WITH CROSS-VALIDATION
for each fold in 5-fold cross-validation **do**
    Use 4 folds as training set and 1 fold as testing set
    Train XGBClassifier model using hyperparameter value of num_boost_round and max_depth from RL Technique Optimization
    Predict on testing set
    Compute F1 score (Weight=macro) and recall for each class
end for
Calculate mean F1 score and recall per class across folds
end procedure

procedure OUTPUT RESULTS
Print average F1 score and recall values
Print mean recall per class
Report evaluation time
end procedure

4 Experiment Results

4.1 Evaluation Metrics:

Macro F1 score: The average F1 score across all classes, with equal weight for each class, regardless of the number of records in each class. This helps address situations where the benign class is much larger than the others.

Mean Recall per Class: Recall values were computed for each class over all folds and averaged to measure the model's detection capability for each class.

4.2 Optimized Hyperparameters and Their Evaluation on Both the Sampling Datasets and the Full-size Dataset

To evaluate model performance, 5-fold cross-validation was conducted on the full-size dataset, using hyperparameter from the RL on sampling dataset. The results is shown in Table 4 with a sampling fraction of 0.1, or 3,519,646 records, the values obtained were `num_cat_boost` = 38 and `max_depth` = 12. The F1 score obtained from the sampling and the full-size dataset, after performing cross-validation, were similar.

With a sampling fraction of 0.01, or 351,964 records, the values obtained were `num_cat_boost` = 50 and `max_depth` = 11. The higher `num_cat_boost` value compared to the previous case was due to the smaller sampling size, which reduced the training time per iteration, allowing for more improvement within the same amount of time. Even though the F1 score did not show statistically significant differences, it was considered better than with a sampling fraction of 0.1.

With a sampling fraction of 0.001, or 35,196 records, the values obtained were `num_cat_boost` = 310 and `max_depth` = 4. It can be observed that as the sampling size decreases, more iterations can be performed, allowing for higher values of `num_cat_boost` and `max_depth`. The F1 score also improved.

Table 4: Optimized hyperparameters value and F1 score on Sampling Size and Full-Size Dataset

Sampling	S1	S2	S3	F1	F2	SD
0.1	38	12	0.9992	0.9970	15	0.0007
0.01	50	11	0.9997	0.9986	22	0.0008
0.001	310	4	0.9999	0.9998	139	0.0002
0.0001	310	4	0.9999	0.9998	139	0.0002
0.00001	110	4	0.9403	0.9979	50	0.0005

Column Descriptions:

Sampling = Sampling Fraction, **S1** = Sampling num_boost_round, **S2** = Sampling max_depth, **S3** = Sampling F1 score, **F1** = Full-Size Dataset Cross-Validation F1 score, **F2** = Full-Size Dataset Cross-Validation Time (Minutes).

With a sampling fraction of 0.0001, or 3,519 records, the values obtained were `num_cat_boost` = 310 and `max_depth` = 4. It can be observed that a smaller sampling size allowed for faster training. However, with such a limited number of samples, it was not possible to train a model with a higher F1 score than before.

With a sampling fraction of 0.00001, or 351 records, the values obtained were `num_cat_boost` = 110 and `max_depth` = 4. As the dataset size became very small, the F1 score began to decline.

In Table 4, we can also observe that the Standard Deviation (SD) of the F1 score across each fold is similar, as the SD values are low.

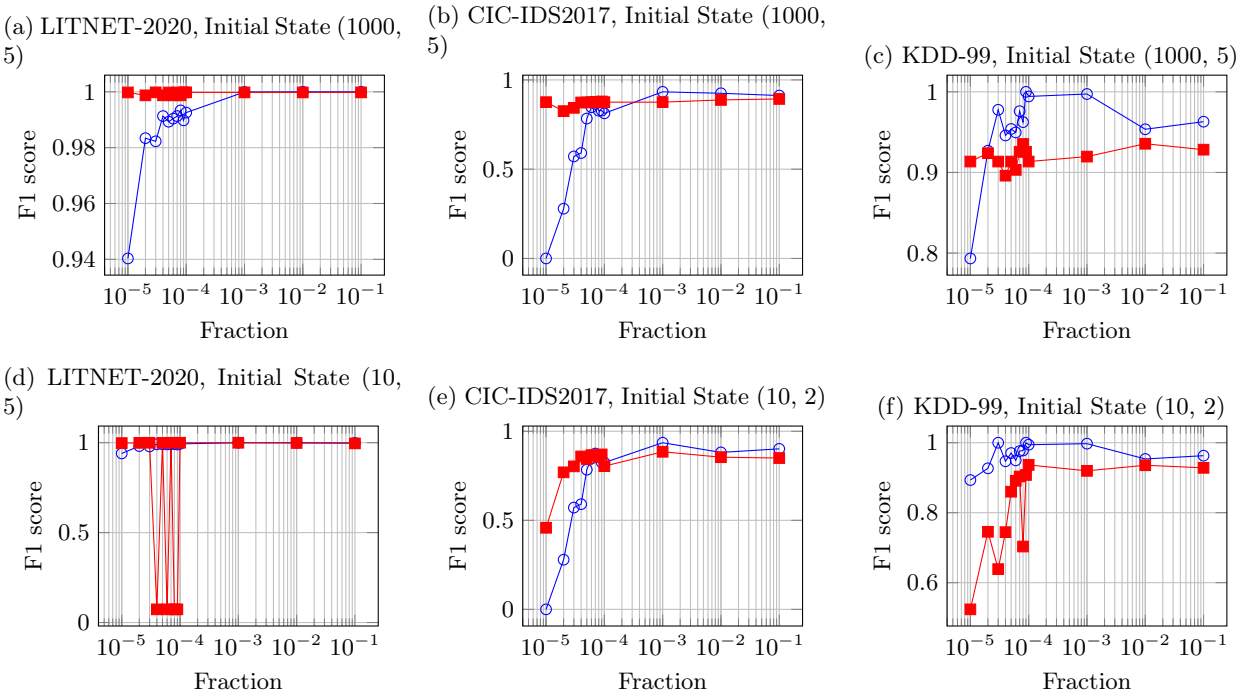

Fig. 4: Comparison of F1 score on Cross Validation Between Sampling (blue circle) vs Full-Size (red rectangle) of Different Datasets and Initial State

4.3 Optimized Sampling Size

From Table 4, the sampling size that achieved the highest F1 score, both in the sample and during 5-fold cross-validation on the full-size dataset, was 35,196 records and 3,519 records, respectively. It can be

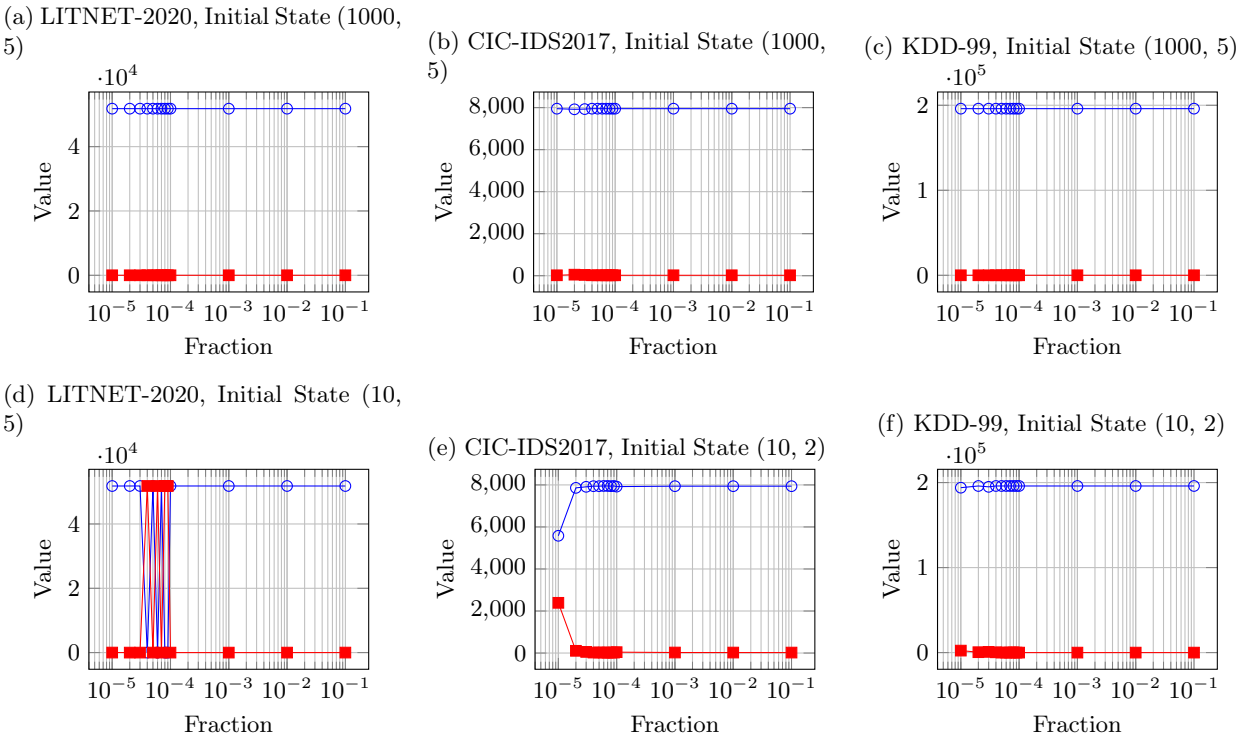

Fig. 5: Comparison of True Positive (blue circle) and False Negative (red rectangle) of Malicious Classes on Different Datasets and Initial State

seen that a dataset size of approximately 30,000 records is sufficient for training a high-performing model. Moreover, with a smaller dataset, the model training process can be completed more quickly. For the dataset with approximately 3,000 records, since the LITNET-2020 dataset is generated, the data is relatively clean. Generally, datasets larger than 10,000 records are preferred. Therefore, using a dataset size of approximately 30,000 records is highly recommended.

4.4 F1 score on LITNET-2020 Compared to CIC-IDS2017 and KDD-99

From Fig. 4, where the Default Initial state is num_boost_round = 1000 and max_depth = 5, we can observe that the F1 score of the full-size dataset (red box plot) shows a similar trend across all three datasets: LITNET-2020, CIC-IDS2017, and KDD-99. The F1 score approaches 1 at its best. When examining the F1 score from sampling at different Fraction sizes, we find that the F1 score increases significantly between Fraction = 0.00001 and 0.0001. After that, the F1 score remains mostly unchanged even as the Fraction increases.

Additionally, we tested a case where the Initial State is very low, with num_boost_round = 10 and max_depth = 2, to observe the progression of using RL to adjust hyperparameters in each iteration. We found that the F1 score for Sampling on LITNET-2020 is consistently good across all Fractions. For CIC-IDS2017 and KDD-99, the F1 score improves between Fractions of 0.00001 and 0.0001. The F1 score for the full-Size dataset, using hyperparameters trained with different Fraction sizes, fluctuates between Fraction = 0.00001 and 0.0001 on LITNET-2020, and remains below 0.9 between Fraction = 0.00001 and 0.0001 on CIC-IDS2017 and KDD-99. In other words, we achieve a good F1 score on the full-size dataset for Fraction values starting from 0.0001 upwards in all three datasets.

4.5 True Positive and False Negative on LITNET-2020 Compared to CIC-IDS2017 and KDD-99

From Fig. 5, where the Default Initial state is num_boost_round = 1000 and max_depth = 5, we can observe that the True Positive values for the full-size dataset (blue circle graph) show a consistent trend across all three datasets: LITNET-2020, CIC-IDS2017, and KDD-99. The True Positive values are high and remain steady across all Fraction ranges, except for CIC-IDS2017, where the True Positive value is lower than the others at Fraction = 0.00001. As for the False Negative values of the full-size dataset (red square graph), they are consistently low across all Fraction ranges, except for CIC-IDS2017, where the False Negative value is higher than the others at Fraction = 0.00001.

Additionally, we tested a case with a very low Initial State (num_boost_round = 10 and max_depth = 2) to observe the development of using RL to adjust hyperparameters. It shows that the True Positive and False Negative values for Sampling on LITNET-2020 fluctuate between Fraction = 0.00001 and 0.0001 before achieving high True Positive and low False Negative values at Fraction = 0.0001 and beyond. For CIC-IDS2017, the True Positive and False Negative values improve at Fractions greater than 0.00001. For KDD-99, the True Positive and False Negative values are consistently good across all Fractions.

4.6 Discussion on the performance of the sampling strategies for NIDS

Although no prior work has directly addressed the specific issue explored in this study, the most closely related work is by Kabir et al. [10], who applied a statistical sampling approach to the KDD-99 dataset and evaluated the results for each class individually (single-class evaluation). However, their method imposed strict constraints on sample selection, which resulted in reduced diversity of the training and testing samples. To enable a fair comparison, we recompiled the experimental results of [10] by selecting five classes (four malicious and one benign) and recomputing the average F1 score across these classes. Their statistical sampling approach achieved an optimal training set size of 494,021 records, resulting in an F1 score of 0.8159 on the test set. In contrast, under the same conditions, our proposed sampling method achieved F1 scores of 0.9356 and 0.9196 using 48,980 and 4,895 records, respectively. Unlike previous method, which applies strict constraints on sample selection, our approach does not impose such constraints and can train models that achieve higher F1 scores on the test set using the NIDS model trained from smaller sample sizes, demonstrating its potential as a more competitive sampling strategy.

However, the overall performance of the proposed method still falls short of state-of-the-art RL based NIDS methods, such as Han et al. [8], which achieved an F1 score of 0.9655 on the CIC-IDS2017 dataset, while our method achieved an F1 score of 0.8935. While the NIDS method can benefit from the sampling strategy proposed in this paper, this suggests that further improvements on the performance are needed to meet the standard requirements.

5 Conclusion

On the LITNET-2020 dataset, we achieved the best F1 score of 0.9998 by tuning the hyperparameters using a sampling size of 0.1%, which corresponds to 35,196 records. Even with a smaller sampling size of 0.01% (3,519 records), we were still able to tune the same hyperparameters. However, with an even smaller sampling size of 0.001% (351 records), the tuned hyperparameters resulted in an F1 score of 0.9979, where the performance started to drop compared to the first two sampling sizes.

The results from testing on two other well-known network intrusion datasets revealed a consistent pattern: both the F1 score and the true positive rate increased as the sampling size grew. This pattern was observed consistently across all three datasets. For CIC-IDS2017, we achieved the best F1 score of 0.8935 by tuning the hyperparameters using a sampling size of 0.1%, which corresponds to 2,820 records. For KDD-99, we achieved the best F1 score of 0.9356 by tuning the hyperparameters using a sampling size of 1%, or 48,980 records

Our proposed dual-phase approach leverages reinforcement learning optimization for hyperparameter tuning on efficiently sampled subsets, followed by cross-validation and final training on the full dataset. This strategy preserves class distribution and overall data representation in the sampled subsets while enabling optimized hyperparameters to be applied to the full dataset, thus avoiding the substantial computational

overhead of full-size tuning. The XGBClassifier, a robust and efficient gradient boosting algorithm well-suited for large-scale network environments, is employed, alongside tailored preprocessing techniques specifically designed for NIDS datasets. As a result, our method achieves markedly higher performance with significantly reduced sampling sizes, providing a practical and scalable solution for intrusion detection model optimization.

In future work, we plan to train the final model on sampled datasets of varying sizes to further explore the trade-off between training time and model performance. Additionally, we intend to conduct experiments on high-dimensional datasets that closely resemble real-world scenarios involving vast and complex data.

References

1. A., A.H., et al.: Machine learning-based intrusion detection system. Journal of Cybersecurity Research (2020)
2. Alikhanov, J., Jang, R., Abuhamad, M., Mohaisen, D., Nyang, D., Noh, Y.: Investigating the effect of traffic sampling on machine learning-based network intrusion detection approaches. IEEE Access **10**, 5801–5823 (2022). https://doi.org/10.1109/ACCESS.2021.3137318
3. Benaddi, H., Ibrahimi, K., Benslimane, A., Jouhari, M., Qadir, J.: Robust enhancement of intrusion detection systems using deep reinforcement learning and stochastic game. IEEE Transactions on Vehicular Technology **71**(10), 11089–11102 (2022)
4. Damasevicius, R., Venckauskas, A., Grigaliunas, S., Toldinas, J., Morkevicius, N., Aleliunas, T., Smuikys, P.: Litnet-2020: An annotated real-world network flow dataset for network intrusion detection. Electronics **9**(5) (2020). https://doi.org/10.3390/electronics9050800, https://www.mdpi.com/2079-9292/9/5/800
5. Dang, Q.V., Vo, T.H.: Studying the reinforcement learning techniques for the problem of intrusion detection. International Journal of Cybersecurity (2021)
6. Dong, S., Xia, Y., Peng, T.: Network abnormal traffic detection model based on semi-supervised deep reinforcement learning. IEEE Transactions on Network and Service Management **18**(4), 4197–4212 (2021)
7. Grigaliunas, F.N.o.I.: Litnet-2020: An annotated real-world network flow dataset for network intrusion detection. https://github.com/Grigaliunas/electronics9050800 (2020), accessed: 2025-03-24
8. Han, H., et al.: Reinforcement learning in cybersecurity. Symmetry (2021), https://www.mdpi.com/2073-8994/14/1/161
9. Hsu, Y.F., Matsuoka, M.: A deep reinforcement learning approach for anomaly network intrusion detection system. In: 2020 IEEE 9th International Conference on Cloud Networking (CloudNet). pp. 1–6. IEEE (2020)
10. Kabir, E., Hu, J., Wang, H., Zhuo, G.: A novel statistical technique for intrusion detection systems. Future Generation Computer Systems **79**, 303–318 (2018). https://doi.org/https://doi.org/10.1016/j.future.2017.01.029, https://www.sciencedirect.com/science/article/pii/S0167739X17301371
11. Li, Z., Huang, C., Deng, S., Qiu, W., Gao, X.: A soft actor-critic reinforcement learning algorithm for network intrusion detection. Computers & Security **135**, 103502 (2023)
12. Lopez-Martin, M., Sanchez-Esguevillas, A., Arribas, J.I., Carro, B.: Network intrusion detection based on extended rbf neural network with offline reinforcement learning. IEEE Access **9**, 153153–153170 (2021)
13. Masum, M., Shahriar, H., Haddad, H., Hossain Faruk, M.J., Valero, M., Khan, M., Rahman, M., Adnan, M., Cuzzocrea, A.: Bayesian hyperparameter optimization for deep neural network-based network intrusion detection (07 2022). https://doi.org/10.48550/arXiv.2207.09902, arXiv preprint
14. Mouyart, M., Machado, G.M., Jun, J.Y.: A multi-agent intrusion detection system optimized by a deep reinforcement learning approach with a dataset enlarged using a generative model to reduce the bias effect. Journal of Sensor and Actuator Networks **12**(5), 68 (2023)
15. Organizers, T.K.C..: Kdd cup 1999 data. The Third International Knowledge Discovery and Data Mining Tools Competition (1999), https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, accessed: 2025-03-21
16. Powell, B.K.M., Machalek, D., Quah, T.: Real-time optimization using reinforcement learning. Computers and Chemical Engineering **143**, 107077 (2020). https://doi.org/https://doi.org/10.1016/j.compchemeng.2020.107077, https://www.sciencedirect.com/science/article/pii/S0098135420305500
17. Rejimol Robinson, R.R., Anagha Madhav, K.P., Thomas, C.: Improved minority attack detection in intrusion detection system using efficient feature selection algorithms. Expert Systems **41**(7), 1–20 (2024), https://research.ebsco.com/linkprocessor/plink?id=9da677a7-ab90-3108-b348-9f7d9777d131
18. Ren, K., Zeng, Y., Cao, Z., Zhang, Y.: Id-rdrl: a deep reinforcement learning-based feature selection intrusion detection model. Scientific Reports **12**(1), 15370 (2022)
19. Ren, K., Zeng, Y., Zhong, Y., Sheng, B., Zhang, Y.: Mafsids: a reinforcement learning-based intrusion detection model for multi-agent feature selection networks. Journal of Big Data **10**(1), 137 (2023)
20. Roshan, K., et al.: A novel deep learning-based model to defend network intrusion detection systems against adversarial attacks. IEEE Transactions on Information Forensics and Security (2022)

21. Sethi, K., Madhav, Y.V., Kumar, R., Bera, P.: Attention based multi-agent intrusion detection systems using reinforcement learning. Journal of Information Security and Applications **61**, 102923 (2021)
22. Sethi, K., Rupesh, E.S., Kumar, R., Bera, P., Madhav, Y.V.: A context-aware robust intrusion detection system: a reinforcement learning-based approach. International Journal of Information Security **19**, 657–678 (2020)
23. Sharafaldin, I., Lashkari, A.H., Ghorbani, A.A.: Toward generating a new intrusion detection dataset and intrusion traffic characterization. 4th International Conference on Information Systems Security and Privacy (ICISSP), Portugal, January 2018 (2018), https://www.unb.ca/cic/datasets/ids-2017.html, accessed: 2025-03-21
24. Vembu, G., Rd, R.: Optimized deep learning-based intrusion detection for wireless sensor networks. International Journal of Communication Systems **36** (06 2022). https://doi.org/10.1002/dac.5254
25. Wang, L., Pan, Z., Wang, J.: A review of reinforcement learning based intelligent optimization for manufacturing scheduling. Complex System Modeling and Simulation **1**(4), 257–270 (2021). https://doi.org/10.23919/CSMS.2021.0027
26. Yang, W., Acuto, A., Zhou, Y., Wojtczak, D.: A survey for deep reinforcement learning based network intrusion detection. arXiv:2410.07612v1 [cs.CR] (2024), 25 Sep. 2024